Automating the Conversion of Natural Language Fiction to Multi-Modal 3D Animated Virtual Environments

A thesis submitted in fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

of

RHODES UNIVERSITY

by

KEVIN ROBERT GLASS

August 2008

Abstract

Popular fiction books describe rich visual environments that contain characters, objects, and behaviour. This research develops automated processes for converting text sourced from fiction books into animated virtual environments and multi-modal films. This involves the analysis of unrestricted natural language fiction to identify appropriate visual descriptions, and the interpretation of the identified descriptions for constructing animated 3D virtual environments.

The goal of the text analysis stage is the creation of annotated fiction text, which identifies visual descriptions in a structured manner. A hierarchical rule-based learning system is created that induces patterns from example annotations provided by a human, and uses these for the creation of additional annotations. Patterns are expressed as tree structures that abstract the input text on different levels according to structural (token, sentence) and syntactic (parts-of-speech, syntactic function) categories. Patterns are generalized using pair-wise merging, where dissimilar sub-trees are replaced with wild-cards. The result is a small set of generalized patterns that are able to create correct annotations. A set of generalized patterns represents a model of an annotator's mental process regarding a particular annotation category.

Annotated text is interpreted automatically for constructing detailed scene descriptions. This includes identifying which scenes to visualize, and identifying the contents and behaviour in each scene. Entity behaviour in a 3D virtual environment is formulated using time-based constraints that are automatically derived from annotations. Constraints are expressed as non-linear symbolic functions that restrict the trajectories of a pair of entities over a continuous interval of time. Solutions to these constraints specify precise behaviour. We create an innovative quantified constraint optimizer for locating sound solutions, which uses interval arithmetic for treating time and space as contiguous quantities. This optimization method uses a technique of constraint relaxation and tightening that allows solution approximations to be located where constraint systems are inconsistent (an ability not previously explored in interval-based quantified constraint solving).

3D virtual environments are populated by automatically selecting geometric models or procedural geometry-creation methods from a library. 3D models are animated according to trajectories derived from constraint solutions. The final animated film is sequenced using a range of modalities including animated 3D graphics, textual subtitles, audio narrations, and foleys.

Hierarchical rule-based learning is evaluated over a range of annotation categories. Models are induced for different categories of annotation without modifying the core learning algorithms, and these models are shown to be applicable to different types of books. Models are induced automatically with accuracies ranging between 51.4% and 90.4%, depending on the category. We show that models are refined if further examples are provided, and this supports a boot-strapping process for training the learning mechanism.

The task of interpreting annotated fiction text and populating 3D virtual environments is successfully automated using our described techniques. Detailed scene descriptions are created accurately, where between 83% and 96% of the automatically generated descriptions require no manual modification (depending on the type of description). The interval-based quantified constraint optimizer fully automates the behaviour specification process. Sample animated multi-modal 3D films are created using extracts from fiction books that are unrestricted in terms of complexity or subject matter (unlike existing text-to-graphics systems). These examples demonstrate that: behaviour is visualized that corresponds to the descriptions in the original text; appropriate geometry is selected (or created) for visualizing entities in each scene; sequences of scenes are created for a film-like presentation of the story; and that multiple modalities are combined to create a coherent multi-modal representation of the fiction text.

This research demonstrates that visual descriptions in fiction text can be automatically identified, and that these descriptions can be converted into corresponding animated virtual environments. Unlike existing text-to-graphics systems, we describe techniques that function over unrestricted natural language text and perform the conversion process without the need for manually constructed repositories of world knowledge. This enables the rapid production of animated 3D virtual environments, allowing the human designer to focus on creative aspects.

Acknowledgments

This research would have been impossible without the intellectual and creative drive of Shaun Bangay, who tirelessly reviewed and critically assessed the work from its start. I would like to express my sincere gratitude for the many hours of discussion, meticulous proof-reading, extreme programming, and motivating guidance during this process.

Thanks to all my friends who have engaged in stimulating conversation and provided motivational support. Particular gratitude goes to Matthew Gibb, who has provided valued proof-reading services, provided advice from a non-technical perspective, and who has been a driving factor in the production of this work.

Most importantly I thank my parents and family for their unconditional support.

This work was undertaken in the Distributed Multimedia Centre of Excellence at Rhodes University. This organization provided generous sponsorship of equipment, travel subsidy, and the test corpora (Penn Treebank and Lancaster/Oslo-Bergen corpus) used in this research. I am also grateful to the Connexor organization for their provision of the syntactic parser used for this research. My gratitude is also expressed towards the creators of open-source tools and freeware that made large portions of this research possible.

The financial assistance from the Henderson Scholarship and National Research Foundation (Scarce Skills Scholarship) towards this research is hereby acknowledged. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to Rhodes University or the donor.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Creating an animated film from a fiction book is a task that requires a number of repetitive activities. These include reading and comprehending the original text for creating detailed descriptions of the film, planning the arrangement and behaviour of entities in each scene, constructing geometry for representing entities visually, and quantifying behaviour for models in the virtual environment. This research reduces the manual effort required for the film-creation task by replacing repetitive activities with automated processes.

Fiction books are popular sources of inspiration for the creation of films because they contain rich visual descriptions regarding background scenery, the layout of people and objects in a scene, and their interactions. A well known example is J.R.R. Tolkien's *The Lord of the Rings* that was adapted to a series of live-action films[1]. This film series is notable in the use of computer graphics, making possible the visualization of fantastical scenes that would have required large quantities of effort and expense to reproduce in reality. In spite of this, the creation of this film series required extensive human effort in adapting the original book to a suitable screen-play, in constructing the geometric models, and in editing and sequencing the final films (as evident by the list of credits following any film in the series).

Many tasks in the process of transforming a fiction book into an animated film have the potential to be automated using computer technology. Technologies already exist that reduce the effort in the creation of animated graphics, including key-frame animation, inverse-kinematics, motion-capture, and fluid and cloth simulations. We speculate that additional tasks stand to benefit from automation, including the analysis and interpretation of language and the population of corresponding virtual environments.

We use the term *fiction-to-animation* to collectively describe the task of converting fiction text to corresponding virtual environments. The term *fiction* refers specifically to text sourced from a fiction book. The term *animation* refers to the creation of moving three-dimensional graphics in a virtual environment, and the creation of other modalities including audio. We use the term *visualize* to describe the creation of graphics that correspond to the original text, but this term also refers to the creation of content in other modalities.

[1]Written between 1937 and 1945, film premiers between 2001 and 2003.

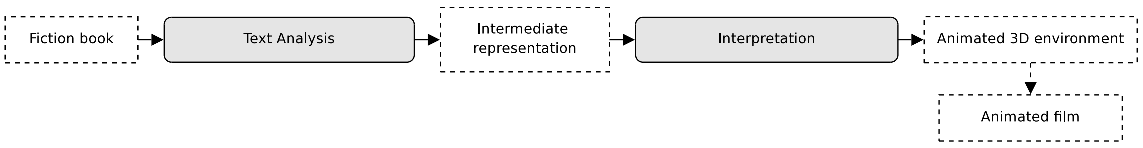


Figure 1.1: Generalization of the film-creation activities when using fiction text as a source.

1.1 Background

The process of creating an animated film is guided by a human director's discretion and creativity, but this process often includes two generic stages, namely the development of a *screen-play* and the development of *story-boards*. The *screen-play* is a document that explicitly describes (in a structured format) the scenes that comprise a film and the character interactions within each scene (Hauge, 1988). S*tory-boards* are constructed from an original screen-play, serving as a plan for background scenery, positioning of actors and objects in each scene, as well as specifying how scenes are sequenced to make up a film (Cantor and Valencia, 2004). The subsequent construction and filming of scenes follows from the story-boards.

In the context of the fiction-to-animation task, the creation of a screen-play and story-boards requires repetitive manual effort. The construction of a screen-play involves a detailed analysis of the original text to derive a structured intermediate representation of the story. This process calls for frequent re-examination and multiple readings of the original text. The task of creating story-boards from the screen-play involves creative interpretation of the described scenes, with regards to specifying layout and behaviour of objects and characters in a setting. If the final presentation is an animated 3D film, then the production includes extensive repetitive effort in constructing 3D models for each character, setting up 3D virtual environments, and explicitly defining motion and articulation in each scene.

We generalize the fiction-to-animation task in terms of two major activities, the *text-analysis* activity and the *interpretation* activity, illustrated in Figure 1.1. The task of creating a screen-play is an example of a *text-analysis* activity, while the tasks of creating story-boards and constructing the 3D environments represent *interpretation* activities. The screen-play forms the link between these two activities, and is an example of an *intermediate representation* of the original story. We choose an intermediate representation that is expressed in a format more suitable for computer-based representation (as opposed to a screen-play), and automate the text analysis task using natural language processing technology. This intermediate representation also allows for automated planning of virtual environments, from which multi-modal animated 3D films are produced.

The methods we select for automating each task are characterized by a central theme, namely the use of *knowledge-poor* techniques for performing the conversion process. Knowledge-poor techniques are characterized by the absence of computer encoded world-knowledge in the form of a special purpose, manually constructed knowledge-database. The use of a knowledge-base limits the capability of an automated system to the detail provided by the encoded information. Populating such a database requires extensive human effort, and we believe that no existing knowledge-base caters for the range of concepts potentially posed by fiction text. We believe that the small amount of knowledge required can be provided by a human without losing the benefit of automation.

1.2 Problem statement

We investigate the automatic conversion of text sourced from a fiction book into a corresponding multi-modal, animated 3D presentation. We rephrase this problem in terms of the following hypothesis:

> The process of converting a fiction book into an animated 3D film can be automated.

To show that automation is supported in the fiction-to-animation process, evidence of automation is required in the text analysis and interpretation activities illustrated in Figure 1.1. Therefore, the problems to be investigated in this research are as follows:

Problem 1 (text analysis): *Can a suitable intermediate representation be generated from a fiction book?*
This problem requires the identification and categorization of visual descriptions in fiction text and their representation in a structured manner.

Problem 2 (interpretation): *Can we create virtual environments that correspond to the intermediate representation?*
This problem is concerned with interpreting the intermediate representation for producing corresponding multi-modal presentations that reflect the content of the original fiction text.

The worst-case scenario is that the human manually creates an intermediate representation from an original fiction book, and manually transforms this representation into an animated film. This corresponds to the current film-creation process described in Section 1.1. We remove the need for manual repetitive tasks by automating majority of the text-analysis and interpretation processes.

The above problems are defined only in the presence of an intermediate representation that is structured enough for automatic creation and interpretation by a computer.

Intermediate representation

We use *annotated fiction text* as an intermediate representation for the fiction-to-animation task. Descriptions of visual information in the fiction text are marked up in different categories. We refer to marked up descriptions as *semantic annotations*, because they identify semantic information regarding the visual scenes in the story. Figure 1.2 presents an example of original fiction text that is annotated with semantic annotations (using XML[2]), and we highlight the following advantages of this choice of intermediate representation with reference to this example:

- Annotated fiction text identifies visual properties described in text, as required by problem 1. For example, the avatars that appear in the scene, the nature of the setting, as well as spatial relations between entities are marked up in the example in Figure 1.2.

[2]Extensible Markup Language

<avatar>Anne</avatar> didn't very much like a big brown **<object>cow</object>** who **<transition type="INSIDE" subject="cow">came</transition>** up **<relation type="near" subject="cow" object="her">close<relation>** and stared at her, but it **<transition type="OUTSIDE" subject="it">went</transition>** away when **<avatar>Daddy</avatar>** told it to.

Figure 1.2: Example annotated fiction text, from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

- Annotations identify different categories of interpretation activities to be performed, including specifying the appearance of the setting, what avatars require visual depiction, and how they are to be placed or moved in a scene. This satisfies the requirement posed by problem 2 above.
- The structured format of the annotations is conducive to automatic creation and interpretation while remaining closely associated with the original fiction text.

We refine the two fiction-to-animation problems in the following sections, where refinements are based on our choice of annotated text as an intermediate representation.

1.2.1 Text analysis

The text analysis problem is concerned with creating semantic annotations over original fiction text.

Manually creating annotations requires decisions to be made by a human annotator that are based on experience with natural language and personal discretion. As a result, we do not expect identical annotations to be created by two different humans. We believe that two factors influence the manual creation of annotations, namely linguistic indicators and world-knowledge. *Linguistic indicators*, including the structure of the text and the function of individual words, are used to convey linguistic concepts to the human. Meaning is derived by combining these concepts with internal *world-knowledge* to formulate decisions about how to create annotations. Exactly how natural language is interpreted and combined with world knowledge is as yet uncertain, and many conflicting theories exist regarding this process (Schank, 1972; Minsky, 1975; Mandler and Johnson, 1977; Eysenck and Keane, 2000).

Linguistic indicators within natural language are described by commonly accepted theories of language structure (such as the *sentence* or the *phrase*) and syntax (such as *verb* or *noun*). World-knowledge cannot be defined in such concrete terms, particularly because each human's world-knowledge and decision process is determined by individual experiences. We do not attempt to create a generic world-knowledge representation, but phrase the problem to allow individual human experience to be captured in the automated process.

We phrase the text analysis problem in terms of the following two sub-problems:

1. Automatically deriving linguistic indicators that identify structural and syntactic properties of fiction text.
2. Creating semantic annotations using a process that conforms to rules derived from human created examples (where rules are phrased in terms of structural and syntactic properties of text).

1.2.2 Interpretation

The interpretation problem encompasses all activities required to translate the intermediate representation into an animated 3D environment. We identify three activities in this process.

The first activity requires a review of the intermediate representation, and the formulation of a high-level plan regarding the contents and layout of each scene. This corresponds to the creation of an initial draft of a story-board, which uses artistic and creative discretion on the part of a human director.

As the story-boards are progressively refined, human experience and creativity are used to plan the layout of each scene, and to ensure that various entities are placed correctly according to either explicit textual descriptions or common-sense constraints (such as gravity).

The final detailed plan is subsequently implemented in a 3D virtual environment, involving a number of repetitive tasks, such as 3D object modeling, placing and posing models, and key-frame definition (tasks vary according to the modeling facilities used). These tasks are complicated by current technology that is restricted to two-dimensional interfaces for designing 3D virtual worlds. This requires multiple views to place objects correctly, as well as a number of interfaces for defining motion, actions, and poses.

The interpretation problem is therefore phrased in terms of three sub-problems:

1. Interpreting semantic annotations to identify which scenes to visualize, the contents of each scene, and the behaviour of entities in each scene.

2. Planning the exact behaviour in a scene in an automatic manner. This problem is characterized by the inclusion of a temporal dimension because the concept of behaviour implies time-based activity.

3. Automating the population of a 3D virtual environment with appropriate visual icons, visualizing the described behaviour, and constructing coherent multi-modal representations of the fiction text.

1.3 Strategy

We use a collection of automated knowledge-poor techniques for automating the fiction-to-animation process. The following sections describe our strategies for performing text analysis and interpretation.

1.3.1 Automation of text-analysis

Automating text analysis requires a process that is able to *learn* about how to create annotations in a similar manner that would be used by one specific human annotator. We automate text analysis using machine-learning, which models a human's thought processes regarding the annotation task using examples created by that human. An automatically generated model is used to produce further annotations.

```
{They/PRP had/VBD it/PRP on/IN the/DT top/NN of/IN a/DT hill/NN ,/@COPY in/IN a/DT sloping/JJ
field/NN that/WDT looked/VBD down/RP into/IN a/DT sunny/JJ valley/NN ./@COPY} {Anne/NNP did/VBD
n't/RB very/RB much/JJ like/IN a/DT big/JJ brown/JJ cow/NN who/WP came/VBD up/RP close/VB and/CC
stared/VBD at/IN her/PRP ,/@COPY but/CC it/PRP went/VBD away/RB when/WRB Daddy/NNP told/VBD
it/PRP to/TO ./@COPY}
```

Figure 1.3: Example fiction text with surface annotations (tokens, sentences and parts-of-speech), from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

We propose the following mechanisms for achieving a machine learning approach in automating the text-analysis problem:

1. Structural and syntactic properties of text are automatically identified in the form of *surface annotations.* Structural properties include tokens, sentences, and quotes. Syntactic properties include parts-of-speech, phrases, and syntactic function (example surface annotations are provided in Figure 1.3). Surface annotations are created using general natural language processing tools ranging from automatic tokenizers and sentence splitters to parts-of-speech taggers and syntactic parsers.

2. We develop a hierarchical rule-based learning mechanism for automating the creation of semantic annotations. This mechanism induces patterns from manually annotated fiction text (supplemented with surface annotations) and uses these patterns for the creation of new annotations.

1.3.2 Automation of interpretation

We automate the interpretation of annotated fiction text in three stages: by interpreting the annotations to formalize which scenes to portray and identify the contents and behaviour in each scene; by calculating values that visually reflect this behaviour in a virtual environment; and by populating 3D environments with visual geometry.

1. Annotations are interpreted automatically to specify scene detail in a structured manner using knowledge-poor techniques. The set of scenes to visualize is derived by segmenting the text into fragments that describe a single physical location (using annotations that identify physical settings in fiction text). The entities that occur in each scene are identified using annotations that indicate references to avatars and objects, and are instantiated visually by selecting geometric models from a library. Behaviour of entities in each scene is expressed using structured time-quantified constraints (derived from annotations) that describe the spatial relationships between entities in a scene over intervals of time.

2. We plan behaviour through the creation of symbolic analytical constraints that describe the behaviour of entities in a virtual environment (examples of which are provided in Figure 1.4). The solutions to these constraints consist of precise numerical values that represent behaviour in a virtual environment. We represent the time aspect using interval arithmetic, allowing behaviour to be specified over contiguous intervals of time. We find solutions to constraints using an interval based quantified constraint optimizer. This approach is beneficial because

- Entity M has a trajectory defined as $\mathbf{r}^M(t) = (1-t)\mathbf{p}_0^M + t\mathbf{p}_1^M$
- Entity N has a trajectory defined as $\mathbf{r}^N(t) = \mathbf{p}_0^N$

Example system of constraints over the two trajectories:
M *near* N: $||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N + \alpha)^2 < 0 \, \forall t \in [t_{start}, t_{end}]$
M *noCollide* N: $||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N)^2 > 0 \, \forall t \in [t_{start}, t_{end}]$
...

Figure 1.4: Example set of symbolic analytical constraints that specify behaviour in a virtual environment.

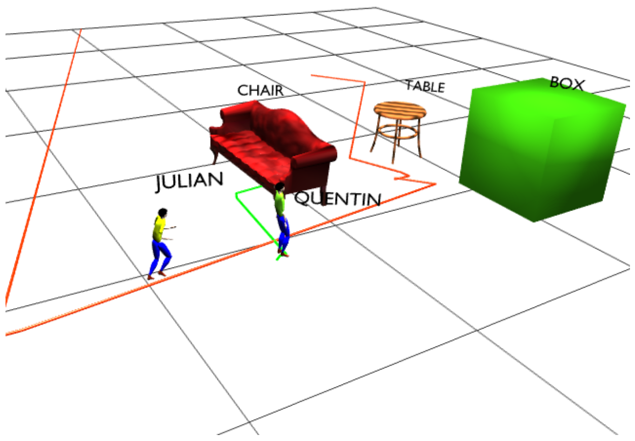


Figure 1.5: Example behaviour visualized in a virtual environment.

it is capable of finding approximate solutions even where automatically generated constraints are inconsistent.

3. We populate a 3D virtual environment by automatically instantiating models for each entity in a 3D environment, and generating geometry for background scenery. Position and motion are automatically assigned to a model in a scene using the quantified trajectories produced from the constraint optimization process (illustrated in Figure 1.5).

The result of these processes are multi-modal animated 3D virtual environments, from which films are rendered. Example snap-shots from the films created using our automated processes are illustrated in Figure 1.6.

1.4 Overview

This dissertation describes the methods we use for automating fiction-to-animation in terms of the corresponding problems defined in Section 1.2. These problems and the manner in which they are related are illustrated in Figure 1.7. The strategies we use for solving each problem are also illustrated, as are points for human intervention in the process.

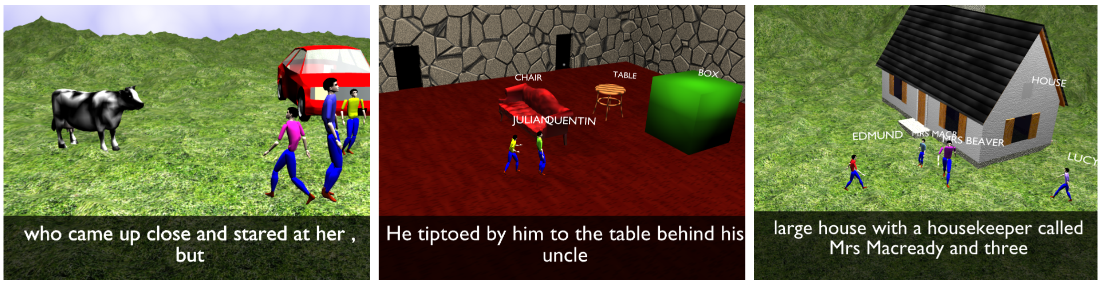

CHAIR
TABLE
BOX
JULIAN
QUENTIN
HOUSE
EDMUND
MRS MACR
MRS BEAVER
LUCY
who came up close and stared at her , but
He tiptoed by him to the table behind his uncle
large house with a housekeeper called Mrs Macready and three

Figure 1.6: Multi-modal presentations produced using our strategy for interpreting annotated fiction text.

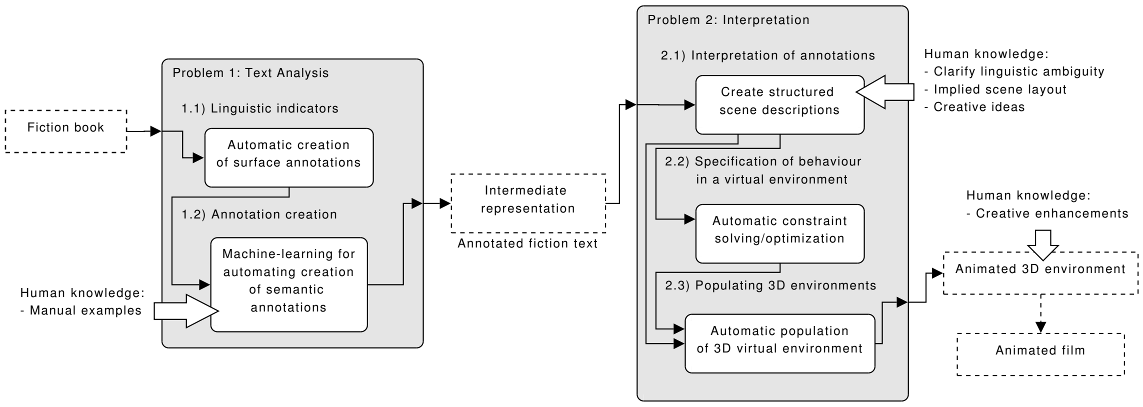

Fiction book
Problem 1: Text Analysis
1.1) Linguistic indicators
Automatic creation of surface annotations
1.2) Annotation creation
Machine-learning for automating creation of semantic annotations
Human knowledge:
- Manual examples
Intermediate representation
Annotated fiction text
Problem 2: Interpretation
2.1) Interpretation of annotations
Create structured scene descriptions
2.2) Specification of behaviour in a virtual environment
Automatic constraint solving/optimization
2.3) Populating 3D environments
Automatic population of 3D virtual environment
Human knowledge:
- Clarify linguistic ambiguity
- Implied scene layout
- Creative ideas
Human knowledge:
- Creative enhancements
Animated 3D environment
Animated film

Figure 1.7: Illustration of the fiction-to-animation problems with proposed solution strategies.

This exposition is structured according to the identified sub-problems as follows:

- Chapter 2 reviews related text-to-graphics research, highlighting the unique characteristics of the fiction-to-animation problem. It also motivates the strategies shown in Figure 1.7 for automating the fiction-to-animation process.
- Chapter 3 describes our strategies for automating the creation of surface annotations over fiction text (relating to problem 1.1 in Figure 1.7).
- Chapter 4 develops the hierarchical rule-based learning approach for creating semantic annotations in fiction text. This system uses surface annotations resulting from methods developed in Chapter 3, and automates the creation of the intermediate representation (illustrated as problem 1.2 in Figure 1.7).
- Chapter 5 develops the method for finding solutions to systems of time-based symbolic constraints. This mechanism is used in the derivation of precise values that represent behaviour in virtual environments, corresponding to problem 2.2 in Figure 1.7. The capabilities of this mechanism influence the manner in which constraint systems are derived from annotated text, explaining why we present this mechanism before the automated derivation of constraint systems (problem 2.1).
- Chapter 6 describes techniques for deriving scene detail from annotations, corresponding to problem 2.1 in Figure 1.7. This includes our strategy for creating structured descriptions

of behaviour from annotated text, their conversion into time-based analytical constraints, and the manner in which the solution-finding process (described in Chapter 5) is used for quantifying behaviour. Chapter 6 also develops strategies for instantiating 3D virtual environments containing geometric models and background scenery (corresponding to problem 2.3 in Figure 1.7). The creation of multi-modal animated films is also covered in this chapter.

- Chapter 7 summarizes the overall strategy for automating the fiction-to-animation task, and presents the conclusions and contributions resulting from this research.

Chapter 2

Techniques for converting text to graphics

2.1 Introduction

The problem of automatically converting natural language to graphical representations has been examined from a number of different perspectives in related research. Techniques vary according to the style and complexity of language used as input, and according to the type of output required. This chapter positions the fiction-to-animation problem in relation to existing text-to-graphics research in these respects.

2.1.1 Categorization of text-to-graphics systems

A *text-to-graphics* system is an automated computer program that takes a sequence of textual symbols as input and produces corresponding graphical representations. Text-to-graphics systems are categorized according to the level of complexity of the input text in relation to its resemblance to natural language. At the lowest level are systems for which input text is structured formally according to well defined grammars (an example of which is VRML[1]) allowing unambiguous interpretation by a computer. At the highest level are systems that interpret actual natural language to create corresponding graphics, but require complex techniques for resolving ambiguities inherent in natural language. Between these levels of input are systems that take input that is *language-similar*, where input bears resemblance to natural language but is structured for interpretation by a computer.

Low level text-to-graphics systems are of little interest in this research because the input lacks resemblance to fiction text. Instead, we examine two categories of text-to-graphics systems: *language-similar* methods, because these are precursors to high-level systems; and high-level *language-to-graphics* systems that successfully transform natural language input into graphical representations.

Systems within each category are further distinguished according to the type of output they produce. A range of different options exists for visualizing text, including modifications to existing

[1]Virtual Reality Modeling Language: `http://www.w3.org/MarkUp/VRML/`

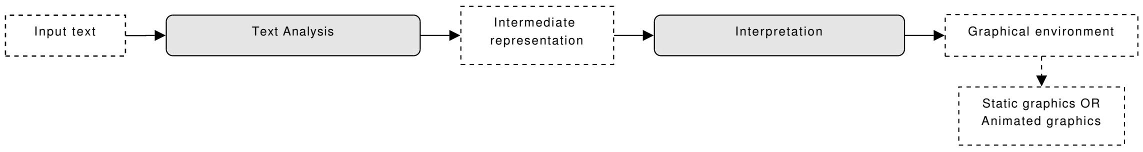

Input text
Text Analysis
Intermediate representation
Interpretation
Graphical environment
Static graphics OR Animated graphics

Figure 2.1: Generic text-to-graphics process.

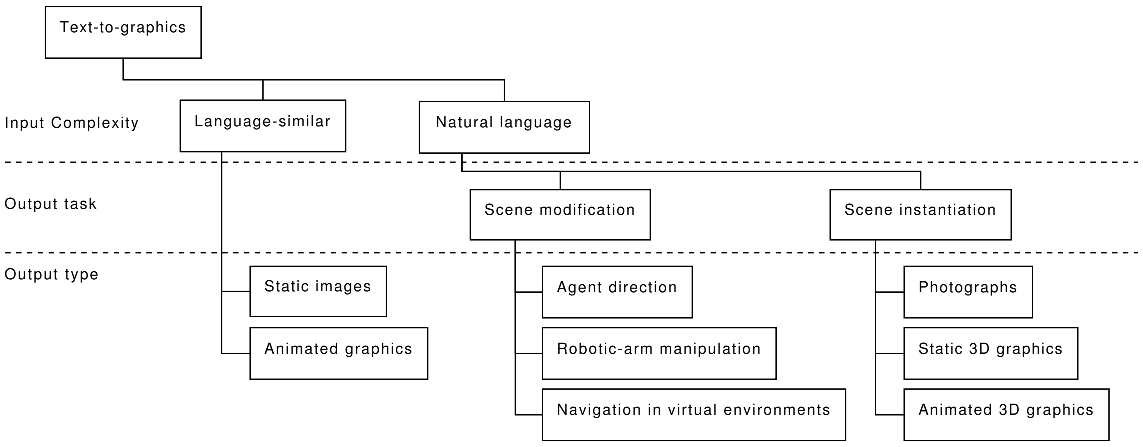


Figure 2.2: Overview of related text-to-graphics systems.

graphical environments in response to commands, the creation of line drawings and 3D images, and the creation of multi-modal 3D animated graphics.

This review highlights the fact that all text-to-graphics systems exhibit similarities with regards to the structure of the process used for conversion. This generic text-to-graphics process is summarized in Figure 2.1. A text-to-graphics system requires a text-analysis process that converts the input text into some computer-readable intermediate representation. This representation is then interpreted to construct or modify a graphical environment, requiring a reasoning process regarding the layout of a graphical environment. Static images or animated graphics are rendered from the final graphical environment. The design of our fiction-to-animation system conforms to this generic process, while differing in the techniques used to accomplish each task.

2.1.2 Overview

This review categorizes existing text-to-graphics systems according to the level of the input required by each system, as summarized in Figure 2.2. Section 2.2 describes language-similar systems, while Section 2.3 describes systems that take natural language as input. Systems are further categorized according to the type of graphical output produced. Thereafter, the fiction-to-animation system is discussed in relation to the previously described text-to-graphics systems (Section 2.4).

2.2 Language-similar graphics-generation systems

Systems that produce graphics automatically from *language-similar* instructions benefit from the ability to parse input text in a reliable manner. Various constituents of the input text are recognized using a formally defined grammar. In spite of this, commands that resemble natural language introduce ambiguity even though the structure of the input is strictly defined. Language-similar input text is used primarily in early techniques for text-based graphics production. Due to limitations in hardware, graphics in the form of line drawings were produced at the time of development of many of these systems. Even with this restriction in output quality, systems exist that create both static images as well as animated graphics from the language-similar input.

2.2.1 Output as a static image

Early language-similar graphics-generation systems produce static line drawings in response to *commands* issued by a human user. Examples of such systems include the following:

- The ENV I system (Boberg, 1972) accepts input such as:
  `(CREATE A BRICK); (PUT PHOBJ1 ON PHOBJ2); (MAKE PHOBJ1 LONGER)`
- The CLOWNS *microworld* (Simmons, 1975) accepts instructions such as:
  `(BOAT ABOVE WATER); (ATTACH BOAT WATER); (DOCK ABOVE WATER)`
- The NALIG system (Adorni et al., 1984) requires input nearer to genuine natural language in the form of:
  `<subject> <preposition> <object>`
- The PUT system (Clay and Wilhelms, 1996) accepts commands of the form:
  `put {“table” on “floor”}`

In each of the above cases the system provides a finite vocabulary and grammar for specifying commands in the task of controlling the positioning of objects in a scene. The restriction of the grammar reduces a large amount of ambiguity in the interpretation of the commands because the function of each token in the input is explicitly defined for the computer. However, the use of natural language terms still presents cases of ambiguity. For example, ambiguous terms such as “LONGER” in the ENV I example (Boberg, 1972), or “on” in the PUT system (Clay and Wilhelms, 1996), are interpreted differently depending on the objects involved.

All the above systems have the common mechanism of mapping units of input directly to modules in which hand-coded decision making processes are executed for resolving ambiguity. For example, PUT (Clay and Wilhelms, 1996) interprets commands using *image schemas*, which are documents that define how relations are realized in a 3D environment with respect to properties of the involved entities. For each preposition such as “on” or “at”, a schema encodes world-knowledge regarding how to handle different cases, for example in placing an item “on the wall” as opposed to “on the table”. Equivalent reasoning modules are used in NALIG in the form of hand coded *rules* (Adorni et al., 1984), or *theorems* in ENV I (Boberg, 1972).

Each of the above systems has the objective of providing an alternative interface for arranging a graphical environment. In particular, ENV I (Boberg, 1972) is concerned with creating and arranging cubes and wedges in a 3D environment, while NALIG (Adorni et al., 1984) and PUT (Clay and Wilhelms, 1996) are concerned with arranging more general 3D objects annotated with spatial properties (such as which surface of a geometric object represents the “top”). In contrast, the

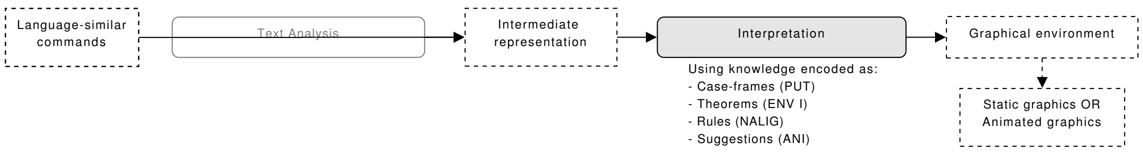


Figure 2.3: Language-similar input for the generic text-to-graphics process.

CLOWNS microworld (Simmons, 1975) is only concerned with arranging a graphical environment containing a finite set of pre-defined objects.

In all cases, the input is a *command* that explicitly describes how a scene must be modified. This is different to a *narrative,* which is a collection of descriptions (only some of which refer to visual aspects), and which is potentially characterized by a temporal aspect.

2.2.2 Output as animated graphics

Early systems exist that convert sequences of language-similar instructions to animated graphics, including the ANI system (Kahn, 1979) and the primitives-based story visualization system (Narayanan et al., 1995). Sequences of language-similar instructions are provided that specify how the graphical environments change:

- The ANI system (Kahn, 1979) takes commands as follows:
  `(CONVEY (wants cinderella (meets cinderella prince)))`
  `(CONVEY (prevents stepmother (meets cinderella prince)))`
  ("cinderella" and "stepmother" are entities declared previously, and keywords such as "meets" and "prevents" are associated with specific routines)

- The primitives-based story visualization system (Narayanan et al., 1995) takes input in which verbs are manually replaced with one of Schank's fourteen primitive actions (Schank, 1973). The sentence "John entered the restaurant" is expressed as:
  `John PTRANS restaurant`

Graphical models are animated by invoking each primitive action in sequential order, although reasoning is used in the ANI system regarding how to order the visualizations. For instance, the two CONVEY commands in the above example are executed simultaneously rather than sequentially.

As with language systems that produce static graphics, the restriction on the input language allows for explicit mapping of natural language keywords to modules that interpret the instructions graphically. Simple animated graphics are the result in both systems, for example consisting of applying motion to two-dimensional symbols in ANI.

All language-similar graphics generation systems map units of input to modules in which hand-coded decision making processes are executed for resolving ambiguity. Because of the structured form of the input, text analysis is not required. World-knowledge is encoded in various forms for the interpretation of the intermediate representation, as indicated in Figure 2.3. The mapping from input symbols to interpretation modules is done directly, a luxury not available when true natural language is used as input.

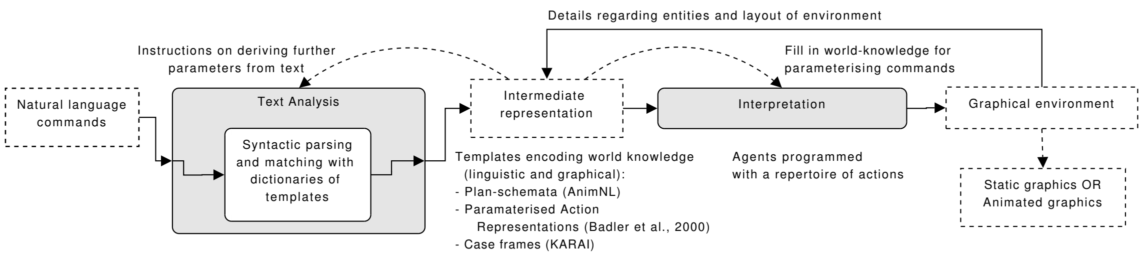

Details regarding entities and layout of environment
Instructions on deriving further parameters from text
Fill in world-knowledge for parameterising commands
Natural language commands
Text Analysis
Syntactic parsing and matching with dictionaries of templates
Intermediate representation
Interpretation
Graphical environment
Templates encoding world knowledge (linguistic and graphical):
- Plan-schemata (AnimNL)
- Paramaterised Action Representations (Badler et al., 2000)
- Case frames (KARAI)
Agents programmed with a repertoire of actions
Static graphics OR Animated graphics

Figure 2.4: Scene modification using natural language input illustrated with respect to the text-to-graphics process.

2.3 Natural language graphics-generation systems

Natural language poses a difficult problem in terms of the text-to-graphics problem because commands or narratives are expressed in a variety of ways, with varying degrees of ambiguity. In some systems *commands* are provided as input in order to modify a pre-constructed graphical environment or assign behaviours to entities within the environment. Alternatively, natural language *narrative* describing a fictitious environment is used as input, from which a graphical representation must be created and possibly animated if the input narrative describes actions and events.

2.3.1 Output as a modification on a pre-existing environment

One of the first examples of language-based manipulation of a 3D environment is the SHRDLU system (Winograd, 1972), in which English commands are used to control a virtual robot-arm in a 3D environment. SHRDLU is significant in that it is one of the earliest systems to use grammatically correct English as input to a graphics-based system, requiring complex syntactic analysis of the input text. Part of the success of the ambiguity resolution is the restriction to a limited domain, which contains only a finite number of entities and permissible actions.

Natural language is often used as the modality for communication between a human operator and entities within a pre-built virtual environment. In particular, techniques exist that allow for natural language commands to be issued to *agents* in a virtual environment (Webber et al., 1995; Badler et al., 2000; Bindiganavale et al., 2000; Shinyama et al., 2000). Example commands handled by these systems include:

`Walk around the room.` (Badler et al., 2000)

`Chicken, push the sphere from the left.` (Shinyama et al., 2000)

Agents are designed to perform a certain range of actions, and the task of the system is to determine which action is being described in the input command, and to determine which entities in the scene are to take part in the action. This process is simplified by the fact that the environments are pre-constructed, and so data is available regarding every entity in the scene for resolving ambiguity.

Most systems make use of an intermediate template representation that encodes both knowledge on how to extract relevant details from the input text, as well as interpret the information for visualizing actions in the graphical environment, illustrated in Figure 2.4. Information derived using syntactic parsing is used to determine which template (schema (Webber et al., 1995), PAR (Badler et al., 2000) or case frame (Shinyama et al., 2000)) is applicable to a certain input word or phrase. The chosen template contains instructions regarding how to derive additional information from the text to parametrize the described action, as well as how to visualize the action graphically (for instance, by sending the appropriate command to an agent).

Natural language is also used as an interface for navigation in virtual environments, specifically controlling the camera providing the view (Bersot et al., 1998). More distantly related is virtual-storytelling, in which input text is interpreted in terms of emotional content, using this information to automatically assign the correct facial expressions to a story-telling avatar (Piesk and Trogemann, 1997).

Input to systems in this category is provided in the form of *commands* that are executed immediately in order to update the graphical environment. The aspect of time is handled by implication, in that no reasoning is performed regarding when an action should occur or how long it should take to execute. Animation occurs in response to an input command, after which the issuer of the command must wait until the action is completed. Ambiguity is avoided because commands are directed to entities that already exist in a scene. This is indicated in Figure 2.4 by the feedback loop between the graphical environment and the intermediate representation. This allows the interpretation to be guided based on the current state of the environment. A more complex formulation of the text-to-graphics problem is the case in which both the environment and the entities must be *created*.

2.3.2 Output as instantiated graphics

A number of alternatives exist for creating graphical representations from natural language input, including producing output as a sequence of photographs, constructing a static 3D environment, or constructing an animated 3D environment from which animated films are rendered.

2.3.2.1 Output as sequences of photographs

An alternative to visualizing natural language text using 3D environments is the use of photographs correctly matched to concepts in input text (Joshi et al., 2004; Zhu et al., 2007). These approaches extract keywords from input text that are used as queries into a database of annotated images or an Internet-based image search engine. One advantage of the text-to-picture approach is that semantic interpretation of the input text is avoided, rather making use of keywords to visualize the text. This allows for large portions of unrestricted text to be visualized as sequences of corresponding images. The problem with this approach is that images (especially returned from an image search engine) tend to be poorly annotated, and the lack of syntactic reasoning potentially results in inappropriate images for a particular keyword (Glass et al., 2007).

2.3.2.2 Output as instantiated 3D environments

Unlike systems that respond to individual natural language commands, some systems create entirely new graphical environments from a narrative describing a scene. This requires both recognition and creation of new entities, as well as interpretation of the entire discourse to determine the

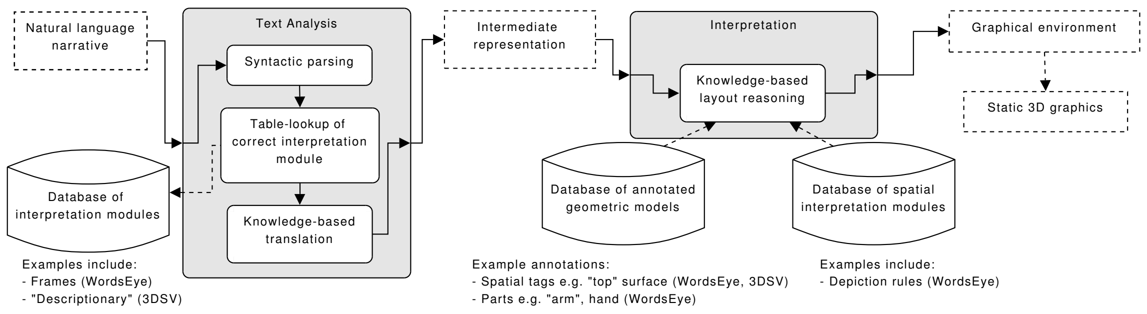

Natural language narrative
Text Analysis
Syntactic parsing
Table-lookup of correct interpretation module
Knowledge-based translation
Database of interpretation modules
Examples include:
- Frames (WordsEye)
- "Descriptionary" (3DSV)
Intermediate representation
Interpretation
Knowledge-based layout reasoning
Database of annotated geometric models
Example annotations:
- Spatial tags e.g. "top" surface (WordsEye, 3DSV)
- Parts e.g. "arm", hand (WordsEye)
Database of spatial interpretation modules
Examples include:
- Depiction rules (WordsEye)
Graphical environment
Static 3D graphics

Figure 2.5: Scene instantiation from natural language narrative illustrated in terms of the generic text-to-graphics process.

global layout of the environment (rather than implement instructions one at a time). Examples of such narratives include:

`John uses the crossbow. He rides the horse by the store. The store is under the large willow.` (Coyne and Sproat, 2001)

`A brick wall is right of a room. A red rug is in the room. A wooden table is on the rug.` (Zeng et al., 2005)

As illustrated in Figure 2.5, a computer-readable intermediate representation must be created from the input text. This is done using a database of interpretation modules, which are documents that contain instructions regarding how to interpret the text in certain scenarios. Surface annotations of the input text are used to perform a look-up in a database that contains different categories of interpretation modules. For instance, words annotated as *nouns* result in a look-up in the database for a module containing instructions that instantiate an entity in the 3D scene (Yamada et al., 1992; Zeng et al., 2003, 2005; Seversky and Yin, 2006). Different categories of interpretation modules exist for different types of semantics, including categories for handling nouns, spatial prepositions, and verbs (Coyne and Sproat, 2001). Each category has a number of defined modules, handling instances that potentially occur within that category. For example, the token "on" is recognised as a preposition by the parser, and consequently a preposition module is invoked that determines how to locate the subject entity and the reference entity of the relation from the text.

The intermediate representation created by the text analysis modules is a translation of the input language into computer-readable format. For example, WordsEye (Coyne and Sproat, 2001) creates a representation in the following format (ellipsis indicates unquoted portions):

```
(...
 ("node5" (:ENTITY:3D-OBJECTS ("cat-vp2842")))
 ("node6" (:STATIVE-RELATION "on":FIGURE "node5":GROUND "node7"))
 ("node7" (:ENTITY:3D-OBJECTS ("pool_table-vp8359"...)))...)
```

The above representation instantiates two entities, and specifies an "ON" relation between them. This representation must be interpreted in the visual context, which is done using world knowledge associated with geometric models and using a number of hand-coded spatial reasoning modules.

Geometric models are annotated with detailed *spatial tags,* information that describes the various regions semantically associated with an object. For instance, tags such as "front" and "behind" are described for each model, as well as more semantically relevant terms such as "at" and "in". For instance, the spatial tag "in" would be defined to indicate the top surface of a geometric model of a "bed", while it would be interpreted as full containment for an model representing a "bedroom". Other data is also associated with models including parts, skeletons and function properties (Coyne and Sproat, 2001).

Spatial tags are used to position objects in the scene. The "ON" example quoted above is realized by aligning the "top" surface of the "pool_table" object with the "bottom" surface of the "cat" object (Coyne and Sproat, 2001; Zeng et al., 2005; Seversky and Yin, 2006). Some scene layout instructions require further world knowledge with regards to spatial representation, such as the "He rides the horse" example mentioned previously. Additional world knowledge is required to determine that "rides" must be translated into an "on" relation when referring to a "horse", but must be interpreted as an "in" relation if the entity were riding a "car". WordsEye makes use of reasoning modules called *depiction-rules* to perform such reasoning (Coyne and Sproat, 2001).

Alternatives to knowledge-guided interpretation exist for the task of performing scene layout. The Spatial Representation Interpreter (Sprint) system (Yamada et al., 1992) formulates the layout problem as a set of constraints, the solutions to which specify the layout of a scene. An alternative to explicitly annotated object models is presented by Seversky and Yin (2006), in which 3D objects are automatically assigned spatial tags using an automatic process that creates voxel representations of the geometric models, and then decides which voxels represent the canonical spatial categories (for example "top" or "bottom").

The defining feature of the above-mentioned systems is the requirement for three categories of world knowledge. World knowledge must be encoded into the linguistic interpretation modules so that the intermediate representation is parametrized to the required level. In addition, world knowledge must be encoded with each geometric model in the database, so that various relations are realized correctly. World knowledge is also required for reasoning about different types of spatial relationship. The success of these systems depends on a sufficient quantity of encoded world knowledge, which requires a large amount of manual effort to create.

2.3.2.3 Output as animated graphics

The creation of animated graphics from natural language is the most complex language-to-graphics application in terms of the requirement to derive temporal information from text. Not only must ambiguities in text be resolved, but concepts such as *events* or *actions* must be recognised within the text and translated into time-based graphical visualizations. Example story narratives provided as input to such systems include:

- CarSim (Johansson et al., 2005) (equivalent Swedish text):
  `The bus was on its way from Kandahar towards the capital Kabul when it left the road while overtaking and overturned, said general Salim Kahn ...`
- Confucius (Ma, 2006) (each entered separately):
  `John put a cup on the table.`
  `John left the gym.`

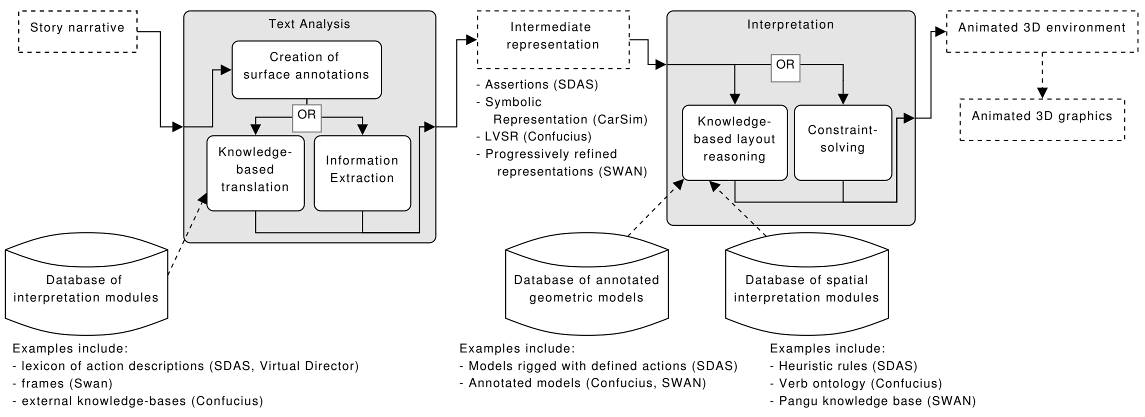

Story narrative
Text Analysis
Creation of surface annotations
OR
Knowledge-based translation
Information Extraction
Database of interpretation modules
Examples include:
- lexicon of action descriptions (SDAS, Virtual Director)
- frames (Swan)
- external knowledge-bases (Confucius)
Intermediate representation
- Assertions (SDAS)
- Symbolic Representation (CarSim)
- LVSR (Confucius)
- Progressively refined representations (SWAN)
Interpretation
OR
Knowledge-based layout reasoning
Constraint-solving
Database of annotated geometric models
Examples include:
- Models rigged with defined actions (SDAS)
- Annotated models (Confucius, SWAN)
Database of spatial interpretation modules
Examples include:
- Heuristic rules (SDAS)
- Verb ontology (Confucius)
- Pangu knowledge base (SWAN)
Animated 3D environment
Animated 3D graphics

Figure 2.6: Creation of animated graphics from natural language narrative illustrated in terms of the generic text-to-graphics process.

- SWAN (Lu and Zhang, 2002) (equivalent Chinese text):
  `The new queen killed the beautiful princess Snow White with a poisonous apple. A prince made Snow White alive again. The prince married princess Snow White in a church.`

Each of the above examples are converted into corresponding animated 3D environments. The CARSIM system (Johansson et al., 2005) creates animations specific to the "car accident" domain, and so the variety of objects and actions are limited accordingly. CONFUCIUS (Ma, 2006) animates an action described in a single sentence, while SWAN (Lu and Zhang, 2002) animates a story expressed as simplified Chinese. Other systems not listed above include the Story Driven Animation System (SDAS) (Takashima et al., 1987) that converts simple children's stories written in Japanese into animated graphics (in which the primary focus is selecting the correct *action* with which to animate a model), and the Virtual Director system (Mukerjee et al., 2000) that makes use of agents that are programmed to perform certain actions and interactions described in a limited "urban parks" domain.

We divide methods for the construction of 3D animations from story narratives into two distinct groups with respect to the text-analysis task, as indicated in Figure 2.6. Knowledge-based text analysis for creating animations is similar to WORDSEYE (Coyne and Sproat, 2001) in that a database of interpretation modules is used to convert input text into a semantic representation. This database is extended to contain categories for actions or events. Systems that make use of hand-coded interpretation modules include CONFUCIUS (Ma, 2006), SWAN (Lu and Zhang, 2002), SDAS (Takashima et al., 1987) and the Virtual Director (Mukerjee et al., 2000). An alternative to the knowledge-based method is the use of information extraction techniques to extract fragments of text that describe visual aspects of a scene, an example of which is the CARSIM system (Nugues et al., 2003; Johansson et al., 2005). The result in both cases is an intermediate representation that expresses the contents of the input text in a computer-readable format, with the added ability to specify time-based events (such as the LVSR representation proposed by Ma and McKevitt (2003)).

Spatial layout reasoning is performed using one of two methods in the creation of animations from the intermediate representation, namely knowledge-based reasoning, and constraint-based planning. As with WordsEye (Coyne and Sproat, 2001), knowledge-based reasoning requires geometric models that are annotated with spatial tags and other types of semantic information (Takashima et al., 1987; Lu and Zhang, 2002; Ma, 2006), as well as modules for interpreting different kinds of relations and actions (Ma and McKevitt, 2004b). The alternative is the use of constraints to specify the layout and motion of entities, the solutions to which are used to construct the graphical environment. Example systems that use this method include CarSim (Egges et al., 2001) and the Virtual Director (Mukerjee et al., 2000).

The above-mentioned techniques result in animated graphics, which implies that the aspect of time is relevant in all cases. The Story Driven Animation System (SDAS) (Takashima et al., 1987) and the Virtual Director (Mukerjee et al., 2000) arrange the visualization of actions according to the order in which the token (from which actions are derived) appears in the text, and each action is manually associated a finite portion of time for execution. Alternatively, specific tokens are used to determine the order of events (for example, "before", "after" or "during") using reasoning that requires data from an external knowledge-base (Lu and Zhang, 2002; Johansson et al., 2005; Ma, 2006).

Of the existing text-to-animation systems described in this section, Confucius, Swan and CarSim produce results most similar to those required by a fiction-to-animation system, in that each produces animated 3D graphics from natural language input. We believe that Confucius and Swan are impressive in their ability to create animated graphics with highly articulated models, while CarSim is impressive in its ability to process large quantities of unsimplified text with minimal use of a manually created knowledge-base.

2.4 Fiction-to-animation in context

Related text-to-graphics systems are summarized in Table 2.1 according to the type of input and output. Language complexity refers to the level of restriction of the input text (in terms of sentence length, sentence complexity, or subject domain). Output type distinguishes between systems that modify pre-existing environments, or create new graphical environments. We include our fiction-to-animation system in this table for comparison. Few systems *create* 3D animations (rather than modify) from *natural language*, and of these only two systems create multi-modal animated graphics. The fiction-to-animation system is the only example in which unrestricted natural language is converted to multi-modal animated graphics.

2.4.1 Trends in text-to-graphics conversion

Common trends exist in text-to-graphics research, characterized according to the method used for text analysis and layout interpretation.

A common trend for text analysis is the use of a semantics based approach, which is concerned with the task of "understanding" the text to the maximum extent possible. This requires syntactic parsing as well as encoded world-knowledge (Coyne and Sproat, 2001; Lu and Zhang, 2002; Ma, 2006). A contrasting approach is to avoid semantic understanding and rather *extract* only items from the input text that are required for creating graphical representations (Johansson et al., 2005).

	Input					Output						
	Content		Complexity			Type		Mode				
	Commands	Narrative	Language-similar	Restricted natural language	Unrestricted natural language	Scene modification	Scene creation	Line drawings	Photographs	3D graphical scenes	Animated 3D graphics	Multi-modal animated graphics
Env I (Boberg, 1972)	✓		✓				✓	✓				
Shrdlu (Winograd, 1972)	✓			✓		✓		✓		✓		
Clowns (Simmons, 1975)	✓		✓			✓		✓				
Ani (Kahn, 1979)		✓	✓				✓	✓				
Nalig (Adorni et al., 1984)	✓		✓				✓	✓				
SDAS (Takashima et al., 1987)		✓		✓			✓	✓			✓	
Sprint (Yamada et al., 1992)		✓		✓			✓	✓				
Primitives-based (Narayanan et al., 1995)		✓	✓				✓	✓				
AnimNL (Webber et al., 1995)	✓			✓		✓					✓	
Put (Clay and Wilhelms, 1996)	✓		✓				✓			✓		
PAR (Badler et al., 2000)	✓			✓		✓					✓	
Karai (Shinyama et al., 2000)	✓			✓		✓					✓	
Virtual Director (Mukerjee et al., 2000)		✓		✓			✓				✓	
WordsEye (Coyne and Sproat, 2001)		✓		✓			✓			✓		
Swan (Lu and Zhang, 2002)		✓		✓			✓					✓
3DSV (Zeng et al., 2003)		✓		✓			✓			✓		
Story Picturing Engine (Joshi et al., 2004)		✓			✓				✓			
CarSim (Johansson et al., 2005)		✓		✓			✓				✓	
Voxel-based (Seversky and Yin, 2006)		✓		✓			✓			✓		
Confucius (Ma, 2006)		✓		✓			✓					✓
Text-to-Picture (Zhu et al., 2007)		✓			✓				✓			
Fiction-to-animation		✓			✓		✓					✓

Table 2.1: Categorization of related text-to-graphics systems according to input and output.

We observe that systems using the latter approach have less stringent restrictions regarding the complexity of the input text.

Spatial reasoning is performed using either one of two methods. The most popular method for positioning objects in a scene is the use of detailed world knowledge in the form of geometric models annotated with spatial and functional characteristics (Coyne and Sproat, 2001; Zeng et al., 2003; Ma, 2006). This includes knowledge in the form of interpretation mechanisms for applying specific types of relations or actions to models. An alternative to this approach is the formulation of constraints that restrict the layout of a scene or trajectory of an object. Solutions to these constraints, obtained either as a result of numerical methods (Yamada et al., 1992) or discrete solving techniques (Johansson et al., 2005), are then transformed into actual graphical representations.

Text-to-graphics systems that require the use of a knowledge-base tend to use custom-built versions that provide the required data specific to the purpose of the individual system (Coyne and Sproat, 2001; Lu and Zhang, 2002; Ma, 2006). No single formulation exists for the structure or content of a generalized knowledge base for the text-to-graphics problem.

Existing text-to-graphics techniques aim to automate the entire conversion process, and there is a strong requirement for external knowledge because natural language is inherently ambiguous. Rather than include a human element in the conversion process to resolve ambiguity, complex reasoning systems are implemented to retain full automation. Because of the labour required in the construction of a knowledge-base, existing systems generally limit the input in some respect. For instance, WordsEye is limited to short sentences, Confucius allows only a single sentence to be used as input at a time, while Swan limits the grammar complexity. CarSim does not limit the complexity of the input text, but does restrict the subject of the text to a single domain. Only text-to-picture systems do not limit the text complexity because no knowledge-based reasoning is required.

2.4.2 Motivation for the design of the fiction-to-animation system

Enhanced representations for fiction books range from digital visualizations of physical books (Chu et al., 2003) and providing ambient sounds effects (Back et al., 1999), to virtual interactive representations (Billinghurst et al., 2001). In all of these approaches the non-textual representations are generated manually, while interaction with the book is the primary focus. Alternatively, we address the problem of converting text as it is found in popular fiction books into corresponding animated graphics, a problem that is an extension to the many text-to-graphics systems presented in previous sections. Text sourced from popular fiction is an example of a natural language narrative without any restrictions on complexity or domain.

We design the fiction-to-animation system to include the same generic stages that exist in other text-to-graphics systems. The automated process consists of a text analysis component and an interpretation component, both illustrated in Figure 2.7. Each component is constructed using a set of sub-components that collectively solve the text analysis and interpretation problems.

Text analysis

The text analysis component is designed to automate the creation of semantic annotations in a manner that adapts to individual human thought processes. We adopt a machine learning approach

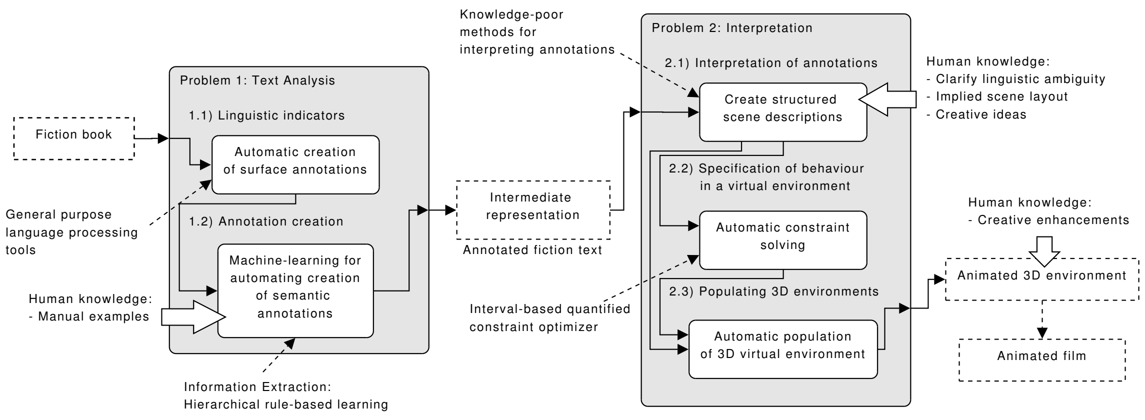

Fiction book
Problem 1: Text Analysis
1.1) Linguistic indicators
Automatic creation of surface annotations
General purpose language processing tools
1.2) Annotation creation
Machine-learning for automating creation of semantic annotations
Human knowledge: - Manual examples
Information Extraction: Hierarchical rule-based learning
Knowledge-poor methods for interpreting annotations
Intermediate representation
Annotated fiction text
Interval-based quantified constraint optimizer
Problem 2: Interpretation
2.1) Interpretation of annotations
Create structured scene descriptions
2.2) Specification of behaviour in a virtual environment
Automatic constraint solving
2.3) Populating 3D environments
Automatic population of 3D virtual environment
Human knowledge: - Clarify linguistic ambiguity - Implied scene layout - Creative ideas
Human knowledge: - Creative enhancements
Animated 3D environment
Animated film

Figure 2.7: Fiction-to-animation process in terms of the generic text-to-graphics process.

to accomplish this, motivated by the use of similar technology in the CarSim system (Johansson et al., 2005). This approach induces models (from human created examples) regarding how to identify portions of text as belonging in a particular category. This is in contrast to the majority of text-to-graphics systems that attempt deep "understanding" of the text through the use of complex knowledge-bases. The benefit of the machine-learning approach is that the complexity of input text is not limited to the ability of any one syntactic parser, nor is there a dependence on a custom-built knowledge base. This benefit is demonstrated by the CarSim system in its ability to handle large quantities of non-simplified language.

We believe that models induced by a machine learning mechanism are more descriptive if they are supplemented with linguistic information that describes the structure and syntactic features of the input text. We use the term *surface annotations* to describe this type of information, and obtain it using a suite of general purpose natural language processing tools. Existing text-to-graphics systems generally derive similar information in this manner (Coyne and Sproat, 2001; Lu and Zhang, 2002; Ma, 2006).

Intermediate representation

The fiction-to-animation system uses annotated fiction text as an intermediate representation. This is motivated by the machine-learning approach to text analysis, because text can be manually annotated by humans and used as example data for training. Intermediate representations used by related systems do not exhibit this property (Coyne and Sproat, 2001; Lu and Zhang, 2002; Ma, 2006).

Interpretation

The interpretation component is responsible for converting the annotated text into corresponding animated 3D virtual environments. We divide the interpretation component into three modules, each of which is guided by a constraint-based formulation of behaviour in the virtual environment. The first module is concerned with interpreting annotations to form spatial constraints that describe behaviour. The second module is concerned with finding solutions to these constraints. The third module uses these solutions to instantiate and populate the virtual environment.

The use of constraint-based formulations of behaviour is motivated by existing systems such as SPRINT (Yamada et al., 1992) and CARSIM (Johansson et al., 2005). Both these systems use constraints to specify behaviour without requiring detailed knowledge to be associated with individual object models (as opposed to systems like CONFUCIUS (Ma, 2006) and SWAN (Lu and Zhang, 2002)).

The manner in which constraints are formed from annotations is dependent on the capabilities of the constraint solving mechanism used to locate solutions. Behaviour implies a relation with time, and solving constraints with respect to time is a challenging task. One method is demonstrated by CARSIM (Johansson et al., 2005), which discretizes time into a sequence of instances, solving constraint systems at each discrete point. We use interval-arithmetic to represent time as contiguous intervals (preventing discretization), and develop a interval-based constraint optimization strategy that is able to find solutions to time-based constraints. The interval-based constraint optimizer represents the second module of the interpretation component.

The first module of the interpretation component is concerned with formulating structured descriptions of the scenes to be visualized. This includes identifying which scenes to portray, which entities to visualize in each scene, and deciding how to portray these entities visually. This modules also derives time-based constraints that describe entity behaviour from the annotated text that can be solved using the interval-based constraint optimizer. We use a suite of knowledge-poor methods for deriving scene descriptions and formulating constraints.

The third module of the interpretation component involves instantiating and populating virtual environments that correspond to the annotated fiction text. This involves methods for placing geometry in a virtual environment (where we prefer 3D geometry for comparability with state of the art text-to-graphics systems like SWAN (Lu and Zhang, 2002), CONFUCIUS (Ma, 2006) and WORDSEYE (Coyne and Sproat, 2001)), for constructing geometry the represents background scenery, and animating entities so that they perform the specified behaviour.

2.4.3 Fiction-to-animation in relation to other systems

Four existing text-to-graphics systems are notable with respect to their ability to create 3D graphics from natural language input, namely WORDSEYE (Coyne and Sproat, 2001), CONFUCIUS (Ma, 2006), CARSIM (Johansson et al., 2005) and SWAN (Lu and Zhang, 2002). Each of these systems exhibit strengths and weaknesses with respect to their capabilities in terms of input and output. We rate these systems according to the quantity, complexity, and domain freedom of the allowed input, and also according to the output quality.

Some systems restrict the quantity of input severely, such as CONFUCIUS, which handles only a single input sentence at a time. Systems such as WORDSEYE and SWAN do not place an upper limit on the number of sentences, but neither of these present examples that extend beyond 20 sentences. CARSIM displays the ability to handle unlimited sentence length in its use of a corpus of 200 newspaper reports. The fiction-to-animation system is most comparable to CARSIM in its ability to handle extracts of increased length.

Sentence complexity is explicitly restricted for the SWAN system. We observe that input examples provided for WORDSEYE and CONFUCIUS are contrived for illustration purposes. In contrast, CARSIM system uses examples sourced from public-domain literature. The fiction-to-animation

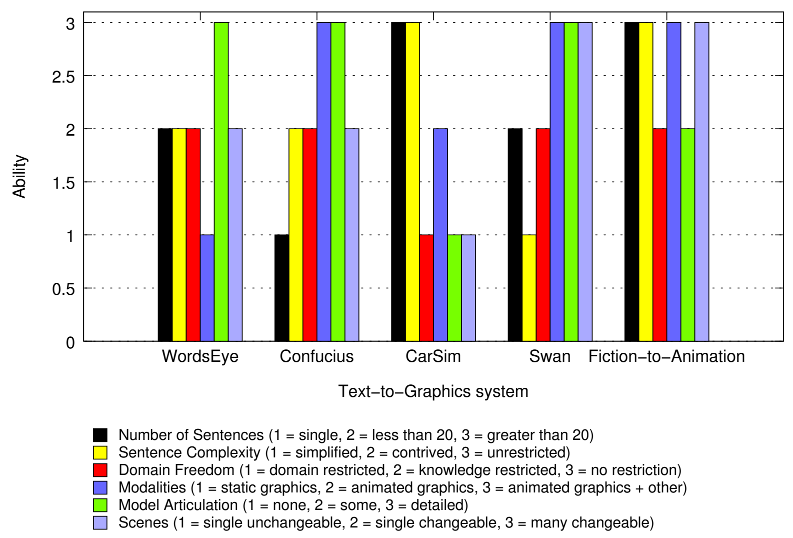


Figure 2.8: Summary of capabilities of related text-to-graphics systems.

system does not use contrived examples, but rather uses text sourced directly from popular fiction books.

CarSim is the only system that expressly restricts the subject domain of the input text (to that of “car accidents”), however, all other systems are limited in their ability to handle a general domain by the extent of the knowledge available to the system. WordsEye, Confucius and Swan are limited by the level of detail of the knowledge-base, a limitation that is reduced if the knowledge-bases are improved. The fiction-to-animation system is also limited in this respect.

Only Swan and Confucius have the ability to create multi-modal presentations of the input text. The fiction-to-animation system also exhibits this capability.

The level of articulation of models (that is, in animating poses and actions) is impressive in WordsEye, Confucius and Swan. This feature is not exhibited in CarSim, which is only concerned with translating models of cars or trucks, without model-level articulation. The fiction-to-animation system articulates models to a certain extent, but not the the level of detail exhibited in other systems.

The output produced by the CarSim system is always depicted in the same visual scene. WordsEye and Confucius have the ability to visualize only a single scene at a time, generally using a textured plane in which objects are placed. Swan has the ability to produce multiple scenes of this structure. The fiction-to-animation system, however, is able to produce a number of scenes from a single fragment of input text, where background geometry is procedurally generated according to the type of scene.

We rate the capabilities of each text-to-graphics system based on the above discussions (each capability is rated on a scale between 1 and 3). A comparative plot such as the one presented in Figure 2.8 indicates that the fiction-to-animation system is superior to existing systems, particularly in its ability to simultaneously handle large quantities of input text of unrestricted complexity, and produce multi-modal animations that occur in a number of scenes.

2.5 Conclusion

This chapter indicates that a variety of methods exist for converting text to graphics, each of which conforms to a common structure. We show that, especially in the case of natural language input, knowledge-bases are generally used to assist in interpreting the input text to aid in the construction of intermediate representations from which graphical outputs are produced. Alternative conversion processes exist, including the use of information extraction and constraint-based methods that reduce the reliance on knowledge-bases. The fiction-to-annotation process is based on the latter techniques, avoiding the requirement for a complex knowledge base.

The fiction-to-animation system is unique in the context of existing text-to-graphics research. It is the first to convert unrestricted text sourced from popular fiction books into multi-modal 3D animations. In addition, the combination of the various components, including an information extraction module for creating annotations and the interval-based quantified constraint optimizer, represent new methods in solving the text-to-graphics conversion problem.

This chapter contributes to the text-to-graphics domain in the following respects:

- The common structure that we derive from related techniques contributes to the understanding of the general text-to-graphics problem, and consists of two primary tasks: text analysis and interpretation. We use this structure for solving the fiction-to-animation problem.
- We identify a gap in text-to-graphics research regarding the conversion of fiction text to multi-modal animated 3D environments and films.
- We provide an innovative classification scheme for related text-to-graphics research, based on the input and output of each system (Table 2.1 on page 20). This contributes to the field by providing a succinct summary of existing technology in a domain where the approaches are significantly varied.

Chapter 3

Surface annotations for fiction text

This chapter describes automatic techniques for identifying the structural and syntactic properties of digitized natural language text (collectively named *surface annotations*). Structural properties indicate which portions of the input text are tokens, sentences and quotes. Techniques for deriving these properties are presented in Section 3.2. Syntactic properties indicate the function of structural units in conveying the meaning of the text, and include parts-of-speech, syntax and phrasing. Techniques for deriving syntactic properties are presented in Section 3.3. We demonstrate the use of structural and syntactic properties in the creation of semantic annotations over fiction text in Section 3.4. Conclusions regarding the automated techniques are presented in Section 3.5.

3.1 Introduction

3.1.1 Problem statement

We believe that human cognition of natural language is assisted by structural and syntactic properties of the text. We investigate the identification of these properties in fiction text in the following respects:

1. Digital fiction text is represented on a computer as a stream of characters, from which structure must be derived. Structural elements include tokens, sentences, and quotes.

2. Structures in digital fiction text can be classified according to their function or use. We investigate the derivation of these functional classifications, including the identification of parts-of-speech, syntax, and phrasing.

3. Tasks in the fiction-to-animation process use structural and syntactic properties to aid in creating semantic annotations. We investigate the levels of accuracy expected in the identification of these properties, so that any error produced is quantified for future experiments that use these properties.

We demonstrate the use of structural and syntactic properties in creating semantic annotations over fiction text, as a motivation for our belief that these properties assist in the cognition of natural language.

	Tokens	Number of sources	Range of Content
SUSANNE	130 000	64 sources from 4 genre categories (extracts from Brown Corpus).	Press reportage, letters, biographies, memoirs, technical writing, adventure and Western fiction (Sampson, 2007).
WSJ	1 288 623	2499 stories from the 1989 Wall Street Journal.	Press reportage (Marcus et al., 1994).
Brown	1 170 775	500 sources, approx. 2000 words each, from 15 genre categories.	Informative and imaginative prose (American English) (Francis and Kucera, 1979).
LOB	1 157 220	500 sources, approx. 2000 words each, from 15 genre categories.	Informative and imaginative prose (British English) (Johansson et al., 1978).

Table 3.1: Size and content of the four annotated corpora.

3.1.2 Problem formulation

Structural and syntactic properties of natural language assist in identifying portions of text that describe visual characteristics of a scene. For instance, tokens (structural property) in the input text that are classified as *nouns* (syntactic property) potentially identify tangible objects in a scene. This chapter identifies methods for identifying structural and syntactic properties of natural language, in the creation of what we term *surface annotations.*

Structural properties reflect the structure of the written language, and in this category we investigate automatic methods for tokenizing fiction text, identifying sentence boundaries and identifying quoted speech. Syntactic properties reflect the function of units of text in written language, and in this category we investigate methods for identifying the parts-of-speech of tokens, the syntactic function of tokens, and the identification and classification of phrases in the text.

Chapter 4 describes a machine learning approach for creating semantic annotations in fiction text. This machine learning approach combines human created example annotations with surface annotations to derive a model of the human's mental processes. This means that the surface annotations must be provided in an accurate manner so that consistent models are created by the learning system. We investigate the accuracy with which surface annotations are created over natural language text using automated processes.

The evaluation of surface annotations requires test data against which the automatically derived properties can be compared, described in the following section.

Evaluation data

Evaluation data for verifying the correctness of structural and syntactic properties of text is found in a resource known as an *annotated corpus.* This is a collection of natural language texts that are manually labeled to indicate structural or syntactic properties. We use four corpora in this chapter, namely the Wall Street Journal (WSJ) section of the Penn Treebank (Marcus et al., 1994), the Brown corpus section of the Penn Treebank (Francis and Kucera, 1979; Marcus et al., 1994), the Lancaster-Oslo/Bergen (LOB) corpus (Johansson et al., 1978, 1986), and the SUSANNE corpus (Sampson, 2007). These corpora are widely used in the field of natural language processing for evaluation purposes, and their size and contents are listed in Table 3.1.

Element	Gold Standard	Automatic Label
e_1	A	A
e_2	B	A
e_3	A	A
e_4	C	A
e_5	A	D

Table 3.2: Example of manual and automatic labeling.

The following section provides a brief explanation of popular metrics for performing evaluation of natural language processing tasks.

Success metrics

General natural language processing is concerned with assigning labels of a certain category to the original text, as is the case with the creation of surface annotations. The accuracy of an automated label-creation process is evaluated by checking the correctness of the automatically produced labels. Example labeling is illustrated in Table 3.2, where each element e_i is labeled as either A, B, C or D. The *gold standard* refers to the correct labeling of these elements, verified by hand. The automatic label is assigned to each element using an automatic process.

There are two metrics commonly used for evaluating automatic labeling mechanisms, namely *precision* and *recall*. Precision, as it relates to the example in Table 3.2, measures the accuracy of the labeler's output:

$$precision = \frac{number\,of\,labels\,correct}{number\,of\,labels\,automatically\,assigned}$$

The precision for labeling an element as "A" in the automated output in Table 3.2 is $\frac{2}{4}$, because 2 of the 4 assigned "A" labels correspond to the gold standard. Recall is a metric that measures the ability of the automatic process to correctly label each element:

$$recall = \frac{number\,of\,labels\,correct}{number\,of\,labels\,in\,gold\,standard}$$

The recall for labeling an element as "A" in the automated output in Table 3.2 is $\frac{2}{3}$ because two of the three elements labeled "A" in the gold standard are labeled "A" automatically.

In some cases, there is no difference between precision and recall. If only the overall success of the automated process is required (that is, not for any specific label type), then precision equals recall because the number of labels assigned is equal to the number of labels in the gold standard. In this case, both metrics measure the *accuracy* of the automated process. Accuracy is $\frac{2}{5}$ for the example in Table 3.2.

All the above metrics are presented as percentages in subsequent sections. The remainder of this chapter describes automated methods for identifying structural and syntactic properties of natural language text, and evaluates these methods over large natural language corpora using precision, recall and accuracy.

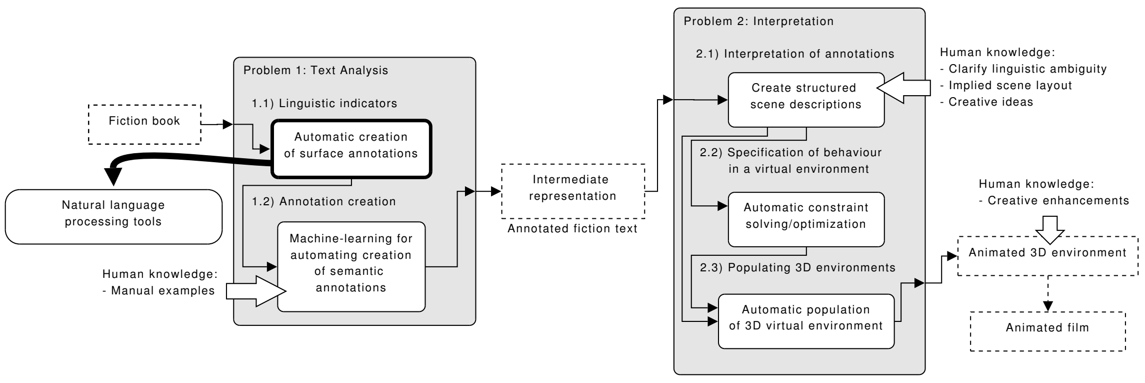

Problem 1: Text Analysis
1.1) Linguistic indicators
Fiction book
Automatic creation of surface annotations
Natural language processing tools
1.2) Annotation creation
Machine-learning for automating creation of semantic annotations
Human knowledge:
- Manual examples
Intermediate representation
Annotated fiction text
Problem 2: Interpretation
2.1) Interpretation of annotations
Create structured scene descriptions
Human knowledge:
- Clarify linguistic ambiguity
- Implied scene layout
- Creative ideas
2.2) Specification of behaviour in a virtual environment
Automatic constraint solving/optimization
2.3) Populating 3D environments
Automatic population of 3D virtual environment
Human knowledge:
- Creative enhancements
Animated 3D environment
Animated film

Figure 3.1: Context of the surface annotation creation process with respect to the fiction-to-animation problem.

3.1.3 Context

The research presented in this chapter examines the automation of the first part of the text-analysis problem in fiction-to-animation conversion (Figure 3.1). We describe a method for automatically identifying surface annotations over fiction text. The input to this process is unaltered digitized fiction text. Surface annotations that are created using automated techniques are used in the machine learning process for identifying semantic annotations described in Chapter 4.

3.2 Structural properties

The following sections present and evaluate techniques for identifying structural components of natural language text, including tokens, sentences, and quotes. We assume certain conventions are used in formatting the digitized input text, which are described in Appendix A.

3.2.1 Tokens

Tokenization is the process of separating sequences of characters into units that resemble words. The structure of English requires a white space between words, but white-space does not always separate words and punctuation marks. Examples include sentence-terminating full-stops (Grefenstette and Tapanainen, 1994; Mikheev, 2002), abbreviations, and acronyms. Punctuation such as *commas, brackets,* and *semi-colons* also exhibit this feature, as do possessive suffixes and contractions such as *'s, n't* and *'ll.*

Our tokenization strategy is based on the assumption that English words are separated using white-space. The input stream of characters is tokenized using white-space as a delimiter. Each token is compared against a lexicon containing 87 309 English words, sourced from the 12DICTS collection (Atkinson, 2003). If a token is a member of this lexicon, then it is recognized as an English word and is not tokenized any further. Tokens that do not occur in the 12DICTS lexicon are potentially acronyms or abbreviations. The token is checked for membership within a lexicon of abbreviations and acronyms, sourced from the ANNIE component of the GATE architecture (GATE, 2005). If a match is found, no further tokenization occurs. If a token is neither an English word nor an abbreviation, then the token has one or more punctuation marks attached.

	Tokens in Brown	Tokens found	Tokens correct	Precision	Recall
Brown Corpus	1170775	1180170	1165852	98.79%	99.58%

Table 3.3: Precision and recall of our tokenization technique over the Brown corpus.

The token is split into sub-tokens using punctuation marks, and apostrophe suffixes as delimiters (identified using manually constructed lists shown in Appendix A). The resulting sub-tokens are then reprocessed using the above methods until all tokens are classified.

We evaluate our approach using the Brown corpus component of the Penn Treebank (Marcus et al., 1994). A plain-text version of the corpus is created by re-binding punctuation to words. The plain-text version is tokenized and the result is compared with the original Brown corpus. Each token in the Brown corpus is matched with the automatically generated output.

In total, 99.58% of the automatically generated tokens match the original, as listed in Table 3.3. This means that less than 0.5% of the original tokens are incorrectly tokenized.

Our method is an alternative to existing tokenization technology that uses machine learning methods (Grover et al., 2000; Clark, 2003). The minimal gains in accuracy achieved using these methods are made at the cost of dramatically increasing the complexity of the tokenization process.

3.2.2 Sentences

Detecting how tokens group together to form sentences is known as *sentence boundary disambiguation* and is dependent on an accurate tokenization process (Mikheev, 2002). This is because periods used to indicate abbreviations must be correctly differentiated from periods used as sentence terminators (full-stops).

We use a custom built sentence boundary disambiguator based on the tokenization approach described in Section 3.2.1. The process identifies sentence boundaries based on three sentence terminating symbols, namely the full-stop, the exclamation mark, and the question mark, each of which is presumed to be correctly tokenized. If closing quotation marks fall after a sentence terminator then the last such token encountered becomes the sentence terminating token.

We evaluate this sentence boundary detection algorithm by measuring the accuracy with which sentence terminators are identified (Mikheev, 2002). We perform this evaluation using the Brown corpus because it contains sentence termination labels. A plain-text version is created by removing all sentence boundary information, and this version is passed through our sentence boundary disambiguator. Two plain-text versions are created: one using correct tokenization as found in the Brown corpus, and the other tokenized using the method described in Section 3.2.1. The former evaluates accuracy where no errors occur in the tokenization, while the latter tests the effect that tokenization errors have on the identification of sentence terminators.

The resulting accuracies of the implemented sentence boundary disambiguator are listed in Table 3.4, showing that the use of this method for identifying sentences produces a small degree of error. We explain the error in these results by the incorrect tokenization of the portion of the Brown corpus containing scientific documents, where acronyms and special symbols confound our tokenization method. We do not believe that fiction text exhibits this type of prose. Errors in tokenization negatively affect the identification of sentence boundaries, but recall is reduced by less than 1%.

	Sentences	Automatic	Correct	Precision	Recall
Correct Tokens	52108	53560	50166	93.66%	96.27%
Automatic Tokens	52108	56709	49947	88.08%	95.85%

Table 3.4: Accuracy of our sentence boundary detection process.

As with tokenization, more complex techniques based on machine learning produce more accurate results, at the cost of increased algorithmic complexity (Palmer and Hearst, 1994).

3.2.3 Quotes

One feature defining fiction text is the presence of quoted text, the function of which is to indicate the speech of characters in the scene. Quoted speech begins and ends with quotation marks, and we annotate portions of text between double inverted commas as quotes. We do not investigate the accuracy with which quoted text is identified because of the straight-forward nature of this technique.

3.3 Syntactic properties

This section describes methods for deriving syntactic properties of natural language text. We investigate three different categories: the parts-of-speech of a token; the syntactic function of a token; and the classification of phrases.

3.3.1 Parts-of-speech

A parts-of-speech tagger automatically assigns labels (tags) to tokens that indicate the word-class (including *noun*, *verb*, *adjective*). This section investigates a number of publicly available parts-of-speech taggers. The part-of-speech of a word is dependent on the context in which it is used. For instance, the word “man” can be placed into two word-classes: *noun*, in the sentence, “The man was walking”; and *verb* in, “He will man the lifeboat”. Autonomous parts-of-speech taggers employ a machine learning mechanism that formulates a model regarding how to classify a word given its context. Publicly available parts-of-speech taggers are distributed with customized, pre-trained models.

This investigation determines the levels of accuracy to be expected from taggers in their pre-trained form, over different types and genres of text. This work differs from previous evaluations in which each tagger is provided with the same training data before testing (Teufel et al., 1996). Given the availability of many different types of tagging systems, this section also investigates the result of combining these taggers into voting ensembles (Glass and Bangay, 2005, 2007b).

3.3.1.1 Parts-of-speech tagging techniques

We identify different classes of freely available parts-of-speech taggers. These are listed in Table 3.5, along with the reported accuracy of each[1]. The reported accuracies listed in Table 3.5 cannot

[1]At time of writing, the Association of Computational Linguistics list the POS Tagger, Stanford Tagger and SVMTool as the top three parts-of-speech taggers available. Source: `http://aclweb.org/aclwiki/` [accessed on 26 September 2007].

Name	Type	Test Corpus	Reported Accuracy
QTag	Probabilistic	Romanian corpus	98.39%
TreeTagger	Decision Tree	WSJ	96.36%
Brill Tagger	Rule-based	WSJ	97.20%
Stanford Tagger	Maximum Entropy	WSJ	97.24%
SVMTool	Support Vector Machine	WSJ	97.20%
POS Tagger	Bidirectional Perceptron	WSJ	97.33%

Table 3.5: Freely available parts-of-speech taggers, and the accuracy reported for each.

be fairly compared because taggers are evaluated using different corpora. The following sections provide an overview of each tagging mechanism.

Probabilistic tagger

Conventional parts-of-speech taggers, otherwise known as *n-gram tagger*s, are probabilistic taggers that examine the previous $n - 1$ words of the current word to establish its word-class. The QTag tool[2] (Tufis and Mason, 1998) implements a window of three words (*trigram*), where the probability of each possible tag for a current word is combined with the likelihood that the tag is preceded by the two previously assigned tags. The tag with the highest probability is selected. The initial probabilities are calculated from a training corpus. QTag has only undergone one definitive evaluation over a Romanian corpus consisting of approximately 250 000 words. Accuracies of between 95.63% and 98.39% are reported (Tufis and Mason, 1998). The version used in our testing is trained for English.

Decision tree-based tagger

The TreeTagger[3] (Schmid, 1994) is a probabilistic tagger that uses a binary decision tree to estimate the probability of a tag being appropriate for a specific word. The TreeTagger was tested on 100 000 words from the Wall Street Journal corpus (sourced from different portions to those used for training), achieving an accuracy of 96.36%.

Rule-based tagger

The Brill tagger[4] uses a rule-based approach (Brill, 1994), where a set of rules for determining word tags is created as follows (during training): tags are randomly assigned to the corpus of words, after which *transition rules* are learned by correcting the falsely identified word-tags. During the tagging process, these rules are applied to identify the correct word tag. The Brill tagger was trained on 600 000 words from the Wall Street Journal corpus, and tested using a separate portion of the same corpus (containing 150 000 words), achieving an accuracy of 97.2%. The rule-based tagger is the only non-statistically based tagger in this collection.

[2]QTag: `http://www.english.bham.ac.uk/staff/omason/software/qtag.html` [accessed on 25 September 2007]
[3]TreeTagger: `http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/` [accessed on 25 September 2007]
[4]Brill Tagger: `http://www.cs.jhu.edu/~brill/RBT1_14.tar.Z` [accessed on 25 September 2007]

Maximum entropy tagger

A *maximum entropy* approach (Toutanova and Manning, 2000; Toutanova et al., 2003) is used for the tagging tool developed at Stanford University[5]. For a given word and its context, every tag in the tag-set is assigned a probability based on data derived during training. The probability of a tag sequence is calculated for a sequence of words, resulting in a probability distribution. The tag associated with the distribution exhibiting the highest entropy, or information gain (Schwartz, 1963), is chosen. This tagger was tested on the Wall Street Journal corpus with reported accuracy rates of up to 97.24%.

Support vector machine tagger

A learning technique based on support vector machines[6] (SVM) is also used for parts-of-speech tagging (Giménez and Marquez, 2003). A dictionary is extracted from a training corpus with all possible tags for each word. Each word tagged as τ in the training corpus provides a positive example for tag τ and a negative example for all other tags. A binary SVM is trained for a specific tag τ using these positive and negative examples. When deciding which tag to assign to a word, the most confident tag according to the predictions of all the binary SVMs is selected. A centered window of seven tokens is used, which is larger than the common window of three tokens in a *trigram* tagger. The accuracy of this tagger is reported to be 97.2% over the Wall Street Journal corpus.

Bidirectional sequence classification tagger

Shen et al. (2007) propose a learning and inference technique called *guided learning* to cater for problems inherent in left-to-right classification schemes (such as the maximum entropy and probabilistic approaches). A bidirectional approach is used but this introduces a new problem of selecting the order in which to conduct the inference process. Guided learning integrates classification of tokens and inference direction into a single learning task. This tagger[7] is tested on the Wall Street Journal corpus yielding accuracy rates of up to 97.33%.

Combination of parts-of-speech taggers

Ensembles of parts-of-speech tagging systems are constructed based on the premise that, although each tagger uses the same contextual information regarding the current word to define its tag, each one makes use of it in a different manner. The combination of parts-of-speech taggers is demonstrated by van Halteren et al. (1998) where combination techniques range from simple voting, to the use of second level learners (machine learning techniques that learn which tag to select from the options presented by the ensemble) (Brill and Wu, 1998; Màrquez et al., 1999; van Halteren et al., 2001).

Related work in combining taggers ensures that the component taggers are all trained using identical training data (Teufel et al., 1996; van Halteren et al., 2001). While providing comparable results, this does not provide a useful indication of "off-the-shelf" value of each parts-of-speech

[5]Stanford tagger: `http://nlp.stanford.edu/software/tagger.shtml` [accessed on 25 September 2007]
[6]SVMTool: `http://www.lsi.upc.edu/~nlp/SVMTool/` [accessed on 25 September 2007]
[7]POS Tagger: `http://www.cis.upenn.edu/~xtag/spinal/` [accessed on 25 September 2007]

	Tokens	Tokens (excluding punctuation)	Full Tag-set	Tag-set (excluding punctuation)
SUSANNE	50 325	42 889	353	-
WSJ	1 288 623	1 114 957	48	36
Brown	1 170 775	1 015 425	48	36
LOB	1 157 220	997 906	153	141

Table 3.6: Number of tokens and tag-set size of the four test corpora.

tagger within the ensemble. This study differs from existing research by avoiding the pre-training of each tagger, under the assumption that the trained tagger provided with each tool is the best trained model for that tool.

3.3.1.2 Tagger accuracy

We evaluate publicly available parts-of-speech taggers with the aim of answering the following questions:

- *Which parts-of-speech tagger produces the most accurate tags over varied natural language sources?*
  Majority of parts-of-speech tagging techniques are trained using the Wall Street Journal corpus. We believe that this corpus is not representative of the style of language used in fiction. This validation is performed over other corpora including the Brown, LOB and SUSANNE corpora, all of which contain extracts from fiction books.
- *Is a higher accuracy achieved by combining parts-of-speech taggers into a voting scheme?*
  The use of a voting system potentially improves the accuracy of the produced tags, and we investigate which combinations of tagging tools, as well as which types of voting methods produce the best accuracy.

We use the corpora listed in Table 3.6 to answer the above questions. Each corpus uses a different set of tags to indicate parts-of-speech. For example, the Penn tag-set used for the WSJ corpus indicates a *preposition* using the symbol PRP while the tag-set used by the LOB corpus indicates a *preposition* using the tag PP. The WSJ uses 48 different tag categories (36 excluding punctuation), while the LOB corpus uses a set of 153 tags, and the SUSANNE 353. All of the tagging tools described in Section 3.3.1.1 use the Penn tag-set, making direct comparison over corpora like the LOB or SUSANNE impossible.

We use an independent coarse tag-set for evaluation to which other tag-sets are mapped. This tag-set is a simplified version of the Penn tag-set (Marcus et al., 1994). Mapping between tag-sets is not always direct (Atwell et al., 1994; Teufel, 1995; Atwell et al., 2000; Déjean, 2000). For instance, mappings of type 1:n occur, where a single tag type in one scheme maps to n tag types in the coarse tag-set. Avoiding these cases is impossible because of the definitions of the tag-sets, and where such mappings are identified, any sentence in the corpus containing such a tag is removed from the test corpus (which explains why the SUSANNE corpus is reported as having 50 325 tokens in Table 3.6, as against its actual 130 000 tokens). The coarse tag-set and the mappings we use from tag-sets of different corpora are listed in Appendix B.

	Reported	WSJ	LOB	Brown	SUSANNE
Statistical (QTag)	98.39	70.40	72.87	72.51	76.077
Tree (TreeTagger)	96.36	96.94	91.67	94.45	***91.15***
Rule-based (Brill Tagger)	97.20	93.10	88.67	92.55	88.45
Maximum Entropy (Stanford)	97.24	91.53	80.21	89.89	85.92
Support Vector Machine (SVMTool)	97.20	***98.17***	***92.17***	***95.10***	90.19
Bidirectional (POS)	97.33	83.36	82.59	84.74	83.02

Table 3.7: Accuracy results of various taggers over the different corpora.

Validation

This experiment validates the accuracy of each of the parts-of-speech taggers listed in Section 3.3.1.1 over the different annotated corpora, with the purpose of answering the following questions:

- *Which parts-of-speech tagger produces the highest accuracy over majority of the corpora?*
- *What is the nature of the errors produced by the parts-of-speech taggers?*

We use the pre-trained models distributed with each tagger in this evaluation. Each corpus is stripped of tags, and the stripped corpus is tagged by each parts-of-speech tagger. A gold standard is created by mapping the original tags in each corpus to the coarse tag-set. The output of each tagger is also mapped to the coarse tag-set. Each tag in the automatically tagged corpus is compared to the corresponding tag in the gold standard, and we measure the percentage of correct tags with regard to the total number of tokens in the corpus (to calculate accuracy). We do not include punctuation in this evaluation.

The accuracy of each parts-of-speech tagger over each corpus is listed in Table 3.7. The accuracies observed do not correspond to those reported for each tagger in related work, and this is explained as follows:

- **Different test data:** majority of the taggers are trained and tested over the WSJ section of the Penn Treebank, specifically tested over only one fifth of this corpus. Results are expected to decrease as a result of larger test-beds, differences in style, genre, and origin of the language (British versus American).
- **Different tag-sets:** we expect that a reduced tag-set makes the tagging task simpler. In the case of the SVM tagger, a higher accuracy than reported in related work is observed because the coarse tag-set is a direct simplification of the Penn Tag-set. However, mapping from other tag-sets such as LOB and SUSANNE introduces error, which explains why all taggers produce lower accuracy scores over these corpora.

The figures in Table 3.7 reflect very high accuracies in spite of the above problems. The reduced accuracies over the LOB corpus are explained by the differences between the American and British lexicons. The SVM tagger produces the most accurate results over the greatest number of corpora.

The top three contributors of error for each tagger are listed in Table 3.8. Each tagger has difficulties with certain tags across the different corpora. For example, 15.79% of the total error encountered by QTag over the WSJ corpus is caused by the incorrect assignment of NN instead of NNP. Many of the errors occurring confuse similar classes of tag. For example, NNP and NN

	WSJ		LOB		Brown		SUSANNE	
QTag	NNP/NN	15.79%	IN/CD	7.71%	NNP/NN	9.21%	IN/CD	8.71%
	IN/CD	6.96%	PRP/NN	7.67%	IN/CD	7.54%	VBN/JJ	6.68%
	VBD/JJ	5.07%	NNP/NN	6.59%	PRP/NN	5.99%	PRP/NN	6.31%
Tree	NN/JJ	8.31%	IN/TO	13.56%	NNP/NN	8.12%	NN/NNP	23.67%
	JJ/NN	7.23%	VB/VBG	7.25%	NN/JJ	4.84%	MD/JJ	4.93%
	IN/RB	5.90%	NNP/JJ	3.95%	NN/NNP	4.82%	JJ/NNP	4.80%
Brill	IN/DT	8.69%	IN/TO	10.00%	IN/DT	8.92%	NN/NNP	21.28%
	VBD/VBN	7.83%	IN/DT	6.85%	VBD/VBN	7.67%	VBD/VBN	7.35%
	VB/NN	7.50%	RP/IN	5.90%	VB/NN	6.57%	RP/IN	6.74%
Stanford	VBD/VBN	14.98%	NN/@COPY@	9.88%	VBD/VBN	14.02%	VBD/VBN	16.23%
	VB/NN	10.56%	IN/@COPY@	6.39%	VB/NN	8.86%	NN/NNP	14.81%
	JJ/RB	6.29%	VBD/VBN	6.21%	JJ/RB	5.44%	VB/NN	6.92%
SVM	NN/JJ	9.71%	IN/TO	14.43%	NN/NNP	8.34%	NN/NNP	26.40%
	NN/NNP	6.78%	VB/VBG	7.49%	NN/JJ	7.66%	JJ/NNP	5.51%
	JJ/NN	5.60%	NN/JJ	4.58%	JJ/NN	4.64%	VB/VBG	4.61%
POS	NNP/NN	38.67%	NNP/NN	21.14%	NNP/NN	27.02%	NNP/NN	23.14%
	VBD/VBN	7.24%	VBD/VBN	7.50%	VBD/VBN	9.05%	VBD/VBN	13.02%
	VB/NN	5.96%	VB/NN	6.67%	VB/NN	6.48%	VB/NN	6.41%

Table 3.8: Top three contributors to error for each tagger.

both represent *nouns*, but the former indicates *proper nouns* as opposed to *common nouns*. The same is true for VBD and VBN which both signify different types of *verb*. This demonstrates that the type of error encountered is on a very fine level of detail. If subsequent processes are only concerned with coarse parts-of-speech detail (for instance, differentiating between *nouns* and *verbs*, regardless of finer classification) then these errors are of no consequence.

In conclusion the SVM tagger most consistently produces the highest accuracy. The majority of the errors produced are the result of the inability of each tagger to differentiate between similar tag categories, and are of no consequence if only broad tag categories are used.

Ensembles of tagging techniques

The purpose of this experiment is to determine if more accurate tagging can be achieved by combining the different parts-of-speech taggers. This experiment is designed to answer the following questions:

- *Which voting scheme produces the highest accuracy and reduction in error?*
- *Which ensemble of taggers produces the highest accuracy and reduction in error?*
- *What is the nature of the erroneous tags produced by the ensemble of taggers that produce the highest accuracy?*

We investigate different types of voting when combining parts-of-speech taggers:

1. **Simple Vote:** the tag that is selected by majority of the taggers is chosen. In the case of a tie, a random tag between the tied parties is chosen.

2. **Weighted Vote:** each tagger contributes a specified weighting for a particular tag. The tag with the highest cumulative score is the tag chosen. The following weighting schemes are tested:

   (a) *TotAccuracy:* each tagger has a weighting equivalent to its overall accuracy, according to the results listed in Table 3.7.

   (b) *TagPrecision:* according to the weighting scheme defined by van Halteren et al. (2001), a tag-specific weighting scheme is employed that uses the *precision* of the specific tagger with regards to any particular tag. Precision for any tag χ is the percentage of tokens tagged χ by the tagger that are also tagged thus in the gold standard:

   $$precision = \frac{number\,\chi\,tags\,correct}{number\,\chi\,tags\,assigned\,by\,tagger}$$

   (c) *Precision-Recall:* according to the weighting scheme defined by van Halteren et al. (2001), a tag-specific weighting scheme is used that not only takes into account how successful a particular tagger is at tagging a certain tag of type χ, but also the error that the other taggers in the ensemble experience when assigned a tag of type χ. Error is derived from a tagger's *recall* rate, where recall for any tag χ is the percentage of tokens tagged χ in the gold standard that are also tagged χ by the tagger:

   $$recall = \frac{number\,\chi\,tags\,correct}{number\,\chi\,tags\,in\,gold\,standard}$$

   Error is calculated as $(1 - recall)$, and measures how often a tagger fails to recognize a specific tag. The weighting assigned by tagger τ for tag χ in this scheme is therefore calculated as follows (where S is the set of taggers in the ensemble):

   $$weight_{\chi}^{\tau} = precision_{\chi}^{\tau} + \sum_{\forall\lambda\in S/\tau} error_{\chi}^{\lambda}$$

   where the set S/τ is the set of taggers, excluding tagger τ.

3. **Ranked Vote:** each of the taggers is given a rank (between 1 and 5) based on the accuracy results listed in Table 3.7, with the best scoring tagger assigned a rank of 5. Tag scores are calculated by adding the ranks of the taggers that voted for each specific tag. The tag with the highest score is chosen.

An alternative to voting is the use of a second level learner for deciding on which tag to choose for a specific word. However, van Halteren et al. (2001) show that these techniques do not perform well when there is a lack of training data. We do not train the taggers, which means that a second level learner cannot be trained for this experiment.

We tag each corpus individually using each parts-of-speech tagger. The tag assigned to a specific token by each tagger is used in the voting schemes described above to determine the most suitable tag for the token. We expect that some ensembles of taggers perform better than others, and we test every possible combination of taggers. We measure the accuracy of each tagged corpus and calculate the reduction in error experienced when using a combination of taggers (based on the metric defined by van Halteren et al. (2001)):

$$error\,reduction = \frac{correct_{ensemble} - correct_{best}}{total - correct_{best}} * 100$$

This metric indicates the reduction in error (as a percentage) achieved using an ensemble from the error produced using the best performing tagger in the ensemble (using the accuracies listed in Table 3.7). For instance, if an ensemble contains two taggers where the best accuracy of the two is 99.0%, then assuming the ensemble results in an accuracy of 99.5%, the percentage error reduction is 50%.

The ensembles resulting in the highest accuracy and largest reduction in error over each corpus are listed in Table 3.9. In the case of the Brown corpus, the same ensemble achieves the highest accuracy and the largest reduction in error, and we also show the `srpt` ensemble as the second best ensemble over this corpus.

Not all ensembles produce an improvement. Many results listed in Table 3.9 are reduced from the highest individual accuracy of the component taggers. This is always the case for the simple voting scheme. The Precision-Recall voting scheme suggested by van Halteren et al. (2001) produces the most reduction in error, and an increase in accuracy over all the component taggers.

The ensemble containing the SVMTool, the Rule-based tagger, the Probabilistic tagger, and the Tree tagger (`srpt`) provide the largest reduction in error over more corpora than any other ensemble.

We examine the precision and recall results obtained for each tag using the Precision-Recall voting scheme with the `srpt` ensemble. The mean precision and recall value for each tag in the coarse tag-set is graphed in Figure 3.2. The maximum and minimum values (over the set of corpora) are also indicated using error-bars. High levels of precision and recall are observed over majority of the tags. However, the ensemble produces poor results over foreign words (FW), particles (RP), and interjections (UH). However, these classes are unlikely to indicate scene-related information and are therefore unlikely to contribute to experimental error in subsequent tasks.

This section demonstrates that the error produced by individual parts-of-speech taggers is reduced using the `srpt` ensemble with the Precision-Recall voting scheme defined by van Halteren et al. (2001). The majority of the errors produced by this ensemble include tags that are unlikely to contribute to future scene related, annotation tasks.

3.3.1.3 Summary of findings

Publicly available parts-of-speech taggers produce high levels of accuracy without requiring further training beyond what is provided with each tool. We conclude the following with regards to the questions posed at the beginning of this section:

- The SVMTool produces the most accurate parts-of-speech tags over different types and styles of text.
- An ensemble of taggers produces higher accuracies than individual taggers, and the ensemble consisting of the pre-trained SVMTool, Brill Tagger, QTag, and TreeTagger produces the largest reduction in error using the Precision-Recall voting scheme.

				Simple		Weighted		TagPrecision		Ranked		Precision-Recall	
Corpus	Maximum:	Ensemble	Best	Accuracy	Error Reduction	Accuracy	Error Reduction	Accuracy	Error Reduction	Accuracy	Error Reduction	Accuracy	Error Reduction
WSJ	Accuracy	`sp`	98.17	85.00	-717.86	98.17	0	98.07	-5.35	98.17	0	98.17	***0.44***
	Error Reduction	`px`	83.36	76.78	-39.52	83.36	0	84.63	***7.64***	83.36	0	85.53	***13.06***
LOB	Accuracy	`srpt`	92.17	91.53	-8.18	92.45	***3.67***	92.73	***7.21***	92.45	***3.61***	92.75	***7.51***
	Error Reduction	`ep`	80.21	76.29	-19.83	80.21	0	85.55	***26.96***	80.21	0	86.63	***32.43***
SUS	Accuracy	`srpt`	91.15	90.98	-1.95	91.09	-0.74	91.23	***0.82***	91.03	-1.34	91.68	***5.96***
	Error Reduction	`px`	83.02	81.39	-9.58	83.02	0	85.72	***15.89***	83.02	0	85.42	***14.13***
Brown	Accuracy and Error Reduction	`srt`	95.1	95.76	***13.48***	95.75	***13.22***	95.85	***15.23***	95.75	***13.22***	95.82	***14.62***
	(second-best)	`srpt`	95.1	94.62	-9.87	95.75	***13.3***	95.72	***12.69***	95.75	***13.31***	95.75	***13.23***

Ensembles are indicated using groupings of the following letters: {s}VMTool, {p}robabilistic tagger, {r}ule based tagger, {t}ree tagger, Max-{e}nt tagger, POS Tagger {x}. Error reductions are indicated in bold-italics.

Table 3.9: Summary of the highest accuracy levels and error reductions over four different corpora.

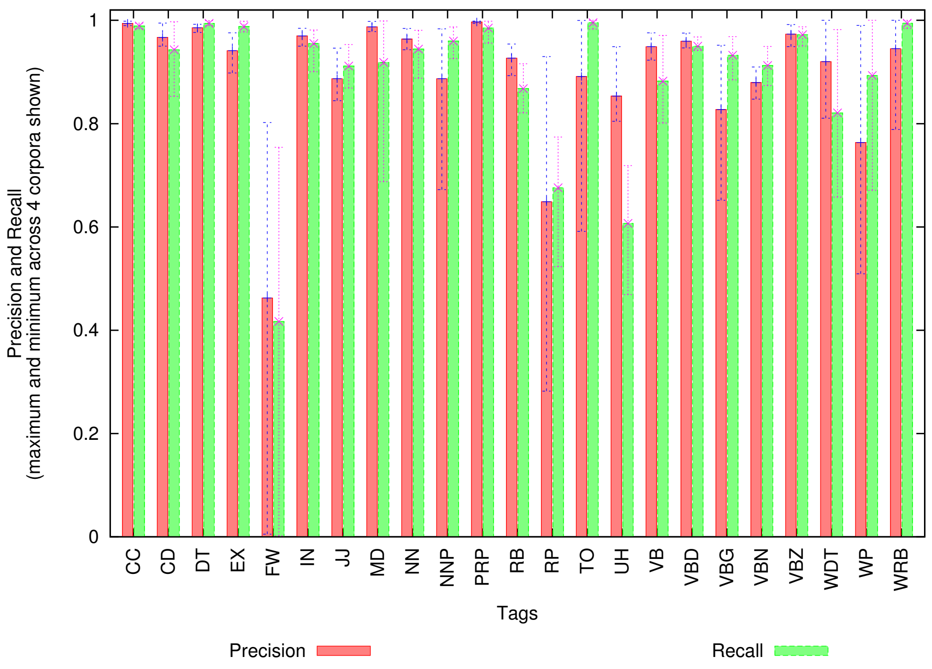
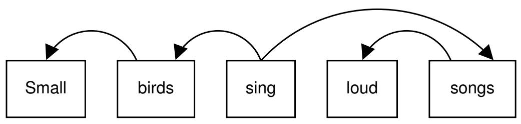
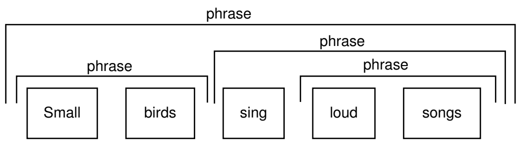


Figure 3.2: Precision and recall value for each tag using the `srpt` ensemble over the LOB corpus.



(a) Dependency Grammar



(b) Phrase Structure Grammar

Figure 3.3: Dependency Grammar versus Phrase Structure Grammar (as illustrated by Hudson (2005)).

3.3.2 Syntax

Syntactic information indicates the function of tokens in a sentence. This information is also useful for identifying scene-related descriptions in fiction text. For example, a token identified as a *main-verb* potentially identifies the primary action described in the sentence, while the token identified as the *subject* of the main-verb refers to the entity that performs this action.

Information regarding the function of tokens in a sentence is provided by a syntactic parser. Two theories exist for automatic parsing, namely phrase structure grammar (PSG) and dependency grammar (DG) (Hudson, 2005). The two are differentiated based on whether the basic unit of sentence structure is the phrase (PSG), or whether the basic unit of structure is the dependency between two words (DG). This difference is illustrated in Figure 3.3, where dependency grammar is concerned with finding the dependencies between words in the sentence and the nature of the dependency (such as *subject* or *object*). In contrast, phrase-structure grammar is concerned with forming groups of words each with a specific function, such as noun-phrase or verb-phrase.

We recognize the validity of both syntactic parsing paradigms. However, we choose dependency grammar as the parsing mechanism for creating surface annotations, implemented as the Functional

Dependency Grammar (FDG) parser from Connexor[8] (Järvinen and Tapanainen, 1998). The reasons for the choice of the FDG parser are summarized as follows:

- The FDG parser indicates function on a token level (for example *subject* and *object*), which corresponds to the structural properties identified in this section.
- The parsing tool is designed for running text (Tapanainen and Järvinen, 1997), and is not intended to maintain the grammaticality of the input sentence (Tapanainen, 1999). This means that ungrammatical sentences are also parsed, which is convenient for fiction books in which text is not guaranteed to adhere to a formal grammar.
- Dependency structures are successfully used in other text-to-scene conversion systems (Coyne and Sproat, 2001) in addition to other natural language processing tasks including anaphora-resolution (Kennedy and Boguraev, 1996; Mitkov et al., 2002).

We do not perform a quantitative evaluation of Connexor parser because of the absence of standardized corpora containing information indicating functional dependency.

3.3.3 Phrasing

In addition to parts-of-speech and syntax, tokens are grouped into *phrases* of a specific type, including noun-phrases or verb-phrases. As discussed in Section 3.3.2, phrase structure grammars offer the ability to create these groupings, but we use tools that rely on a simpler method called *phrase chunking* (Kudo and Matsumoto, 2001).

Phrase chunking is accomplished using similar methods to parts-of-speech tagging, employing machine learning methods such as statistical methods (Xun et al., 2000), support vector machines (Kudo and Matsumoto, 2001), or rule-based learning (Ramshaw and Marcus, 1995) to induce models that identify and annotate phrases in natural language text. We use the LTChunk[9] tool, created by the Language Technology Group in Edinburgh, because of its ability to make use of already computed parts-of-speech annotations. This tool takes text tagged using the `srpt` ensemble as input, and produces text annotated with noun-phrases and verb-phrases.

We do not perform a quantitative evaluation of the phrase chunker due to the lack of standardized test data for this task.

3.4 Case study: the use of surface annotations in identifying Avatars

This section illustrates the use of surface annotations in the creation of semantic annotations. One category of semantic annotation identifies *avatars* that take part in the described story. An example of an avatar annotation is presented in Figure 3.4.

The identification of avatars in fiction text is a specialization of the more general natural language processing problem of identifying named-entities (McDonald, 1996; Bennett et al., 1997;

[8]Connexor Machinese Syntax: `http://www.conexor.fi/` [accessed on 25 September 2007]
[9]Language Technology Group, Edinburgh: `http://www.ltg.ed.ac.uk/software/posdemo.html` [accessed on 10 August 2005]

"Hurrah!" thought <**avatar**>**Julian**</**avatar**>. "He's off again!" Quietly he stood up, holding the box.

Figure 3.4: Example text annotated in the Avatar category, from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

	Total	Automatic	Correct	Precision	Recall	Gender
Book 1	17	25	11	44.0%	64.71%	100.0%
Book 2	25	55	15	27.27%	60.0%	60.0%
Book 3	18	53	15	28.30%	83.33%	80.0%

Table 3.10: Success metrics for the automatic identification of avatars in fiction text.

Borthwick et al., 1998; Cohen and Sarawagi, 2004). This task is concerned with locating fragments of text that refer to avatars, and determining the gender of each identified avatar.

We make use of surface annotations to identify avatars and determine gender. We traverse every token in the text (identified using methods in Section 3.2.1), and consider each token annotated as a *proper noun* (using the ensemble of parts-of-speech taggers described in Section 3.3.1) as a candidate avatar. We maintain a list of unique candidates, and maintain a count of the number of times each candidate is mentioned. Every candidate with a count higher than a certain threshold is selected as an avatar. All tokens corresponding to a selected avatar are annotated.

We make further use of token and parts-of-speech information to determine the gender of an avatar. Personal pronouns indicate gender ("he" indicates a masculine reference, "she" indicates a feminine reference), and we count the number of masculine and feminine personal pronouns within the immediate vicinity of an avatar annotation (30 tokens to either side of the reference). The gender with the greatest percentage of personal pronouns in the vicinity of each reference is chosen as the gender for the specific avatar.

We evaluate the success of the avatar and gender identification processes by evaluating the list of unique avatars identified by the automatic process with respect to a manually created list. The success with which a set of avatars is automatically derived is listed in Table 3.10 for three different fiction books. The percentage of correctly identified avatars exceeds 60% in all cases (recall). The lower precision values indicate that additional avatars are identified erroneously, and these candidates must be manually deleted from the list by a human. The accuracy with which gender information is identified is also listed in Table 3.10 for each book, and indicates a high level of success for the pronoun-based approach.

The avatar identification process is successful in the context of fiction-to-animation conversion, and demonstrates the use of surface annotations in the identification of scene-related descriptions.

3.5 Conclusion

Structural and syntactic indicators are identified in digitized natural language text using automated techniques. We conclude the following with respect to the problems identified in Section 3.1:

1. Structural properties of digitized text are identified with a high level of accuracy using custom-built, knowledge-poor methods.

2. Syntactic properties are identified automatically with high levels of accuracy:
   (a) Parts-of-speech are identified accurately using publicly available taggers, even over text that is different in style and genre to the original training data. Accuracy is improved using an ensemble containing the SVMTool, Brill tagger, QTag and TreeTagger. The Precision-Recall voting scheme (van Halteren et al., 2001) produces the largest reduction in error.
   (b) Syntactic function of tokens in the input text are identified automatically using the Connexor FDG parser.
   (c) Phrases are identified in the input text using the LTChunk tool.
3. High levels of accuracy (greater than 95% in all quantified results) are expected in the creation of surface annotations over fiction text.

We believe that the errors produced by these automated processes have minimal impact on future text-analysis tasks. This belief is motivated by the fact that visual descriptions in fiction text are intermittent (that is, scattered throughout the text), and the few inconsistencies in the surface annotations potentially have no relation to these descriptions.

This chapter contributes to the text-to-graphics domain and the automated parts-of-speech tagging domain in the following respects:

- Text-to-graphics research often regards the creation of surface annotations as a "black-box" process, and does not examine methods for improving the results of these methods. The work presented in this chapter studies these problems in detail, allowing future enhancements to these components to be quantified if required.
- The work we present is the first study of automated techniques that evaluate the "off-the-shelf" usefulness of parts-of-speech taggers over large corpora. This research also contributes a reduced tag-set and corresponding maps to permit tagging and validation over a number of corpora using different tag schemes.

Future work includes the identification of further categories of surface annotation that aid in subsequent creation of semantic annotations. Categories include the recognition and classification of named-entities in text, the automatic resolution of ambiguity in the form of pronominal anaphora and co-reference (a task briefly examined in Chapter 6).

Chapter 4

Creation of annotated fiction text

This chapter presents *hierarchical rule-based learning* for automating the creation of annotated fiction text. We define the concept of an annotation as a mechanism for marking up portions of text in a particular semantic category (Section 4.1). Related strategies for automatically creating annotations are investigated in Section 4.2, where we motivate the use of a pattern-based machine learning technique. This machine learning technique constructs generalized patterns from manually created example annotations in a particular category, and these patterns are used for producing annotations (described in Sections 4.3 and 4.4). A range of semantic annotation categories for the fiction-to-animation task are developed in Section 4.5, and we investigate the properties of hierarchical rule-based learning in terms of these categories in Section 4.6 using a suite of experiments. Conclusions and contributions of the research presented in this chapter are presented in Section 4.7.

4.1 Introduction

4.1.1 Problem statement

Popular fiction books consist of natural language descriptions of rich visual environments that describe characters, objects, and behaviour. We investigate the identification and categorization of visual descriptions using annotations. This problem is characterized as follows:

1. Fiction text is interpreted differently based on an individual human's knowledge and experience. We investigate the modeling of individual human knowledge and experience in order to create annotations.

2. Fiction text contains different categories of visual information, a number of which must be identified for creating an intermediate representation (defined in Chapter 1 as the structured, visual representation of the fiction book). We investigate the induction of models that automate the creation of different categories of annotation.

3. Categories of visual information are parametrized in fiction text using other portions of text or by associating semantic information during human interpretation. We investigate methods for automatically identifying these additional parameters for an annotation.

> So her mother handed her some chocolate, and she and the boys munched happily, watching the hills, woods and fields as the car sped by.
> The picnic was lovely. They had it on the top of a hill, in a sloping field that looked down into a sunny valley. Anne didn't very much like a big brown cow who came up close and stared at her, but it went away when Daddy told it to.

Figure 4.1: Fiction text from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

The automatic creation of annotations is a problem defined in the context of fiction-to-animation. A solutions to the above problems should be verified according to its ability to handle different types of fiction text (for example different authors, genres and target audiences). A solution should also be capable of reducing the amount of effort required in creating the intermediate representation.

4.1.2 Problem formulation

Fiction books contain many types of visual descriptions, examples of which occur in the extract in Figure 4.1. The setting is established using words such as "hill" and "valley", while the contents of the scene are established using references to avatars (for example "Anne", "Daddy") and the objects ("chocolate", "car", and "cow"). Spatial relations are indicated between entities ("the cow who came up close"), and transitions are described that specify an entity's arrival or departure ("but it went away"). Visual attributes are specified ("big brown cow") for entities, emotional expressions ("happily"), and individual actions ("handed her some chocolate").

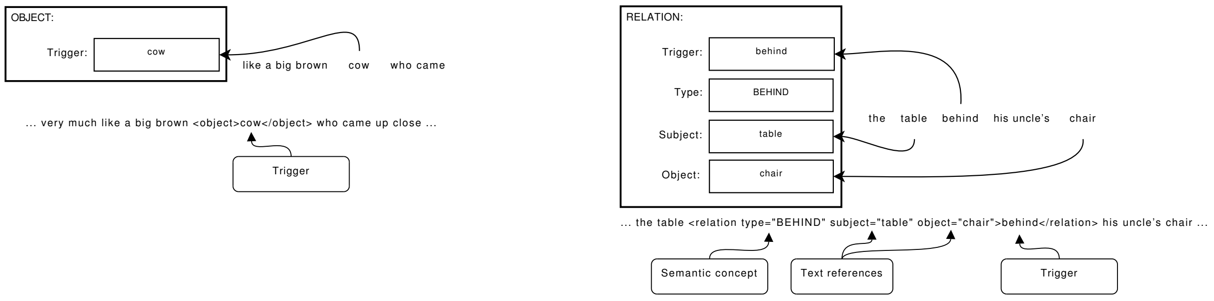
We *annotate* descriptions in fiction text to identify a particular fragment of text as a visual description, and also to specify the category to which the description belongs. Identifying the category assists in future interpretation of a description. For example, if the token "cow" is identified as belonging to the *object* category, then an automated process can be executed for locating a cow-shaped geometric model and placing it in a 3D environment (described further in Chapter 6).

An annotation in its simplest form is a marker that identifies a fragment of text (for example, a set of tokens) as belonging to a certain category. The following example contains two annotations in the *object* category:

```
The <object>box</object> was on the <object>table</object>.
```

The above example also contains a description of a spatial relation, indicated by the token "on". Spatial relations are parametrized in terms of the involved entities, in this case the *subject* of the relation (the entity affected by the relation: "box"), and the *object* of the relation (the reference point of the relation: "table"), both of which contribute to the meaning. In this respect, multiple text fragments can form a single annotation, where each fragment plays a specific role. The above example is extended as follows:

```
The <object>box</object> was <relation subject="box" object="table">on</relation>
                          the <object>table</object>.
```


OBJECT:
Trigger:
cow
like a big brown cow who came
... very much like a big brown <object>cow</object> who came up close ...
Trigger
RELATION:
Trigger:
behind
Type:
BEHIND
Subject:
table
Object:
chair
the table behind his uncle's chair
... the table <relation type="BEHIND" subject="table" object="chair">behind</relation> his uncle's chair ...
Semantic concept
Text references
Trigger

(a) Object annotation (b) Relation annotation

Figure 4.2: Illustration of Object and Relation annotations.

Annotations also specify semantic details regarding the fragment of text they identify. For instance, the token "on" identified in the above example is linked with the semantic concept $onTopOf$, providing for subsequent reasoning regarding the layout of these objects in a virtual environment. This is specified as another *field* in the annotation:

```
The <object>box</object> was <relation subject=“box” object=“table”
        type=“onTopOf”>on</relation> the <object>table</object>.
```

The above examples represent annotations using XML[1] syntax. Regardless of representation, we define an annotation as follows:

Definition 4.1. An annotation is a portion of text marked up in a particular category. Annotations define the following type of information:

- *Trigger*: the fragment of text identified as belonging to the category. Every annotation contains one trigger.
- *Text-references*: an annotation is qualified by zero or more references to other fragments of text that have a relationship with the trigger.
- *Semantic-concepts*: an annotation is qualified by zero or more semantic-concepts that are associated with the trigger.

Each category of annotation is parametrized using associated text-references and semantic-concepts. We use the term *qualifier* to collectively describe these parameters. Categories such as *object* have no associated text-references or semantic-concepts, only identifying a trigger that refers to an object. A *relation* annotation identifies a trigger describing a spatial relation, and is qualified by two text-references (*subject* and *object*), and a semantic *type* indicated by the annotation. These two categories of annotation are illustrated in Figure 4.2.

Annotated text is conducive to manual creation. We use a graphical interface that allows a human to select portions of text that belong in a particular category. Annotated text is also effective for human readability, allowing annotations to be viewed or edited in the context of the surrounding text.

[1]Extensible Markup Language

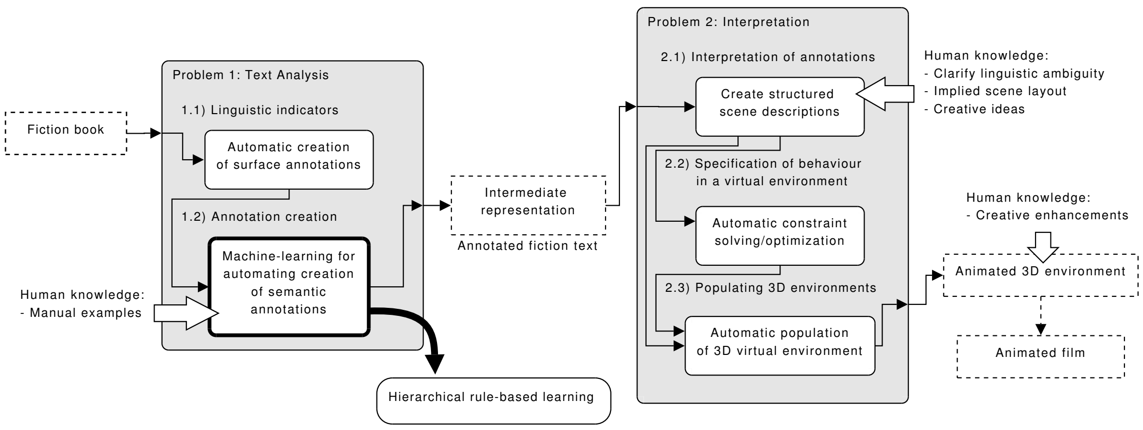

Fiction book
Problem 1: Text Analysis
1.1) Linguistic indicators
Automatic creation of surface annotations
1.2) Annotation creation
Machine-learning for automating creation of semantic annotations
Human knowledge:
- Manual examples
Intermediate representation
Annotated fiction text
Hierarchical rule-based learning
Problem 2: Interpretation
2.1) Interpretation of annotations
Create structured scene descriptions
Human knowledge:
- Clarify linguistic ambiguity
- Implied scene layout
- Creative ideas
2.2) Specification of behaviour in a virtual environment
Automatic constraint solving/optimization
2.3) Populating 3D environments
Automatic population of 3D virtual environment
Human knowledge:
- Creative enhancements
Animated 3D environment
Animated film

Figure 4.3: Context of the hierarchical rule-based learning mechanism with respect to the fiction-to-animation problem.

The first task in creating an annotation in fiction text requires the identification of what fragments of text represent triggers within a particular category. This involves discriminating text fragments that belong in the category (positive examples) from those that do not (negative examples). Once a trigger is located, the fragments of text are then located for each defined text-reference (if defined for the particular category of annotation), and semantic-concepts are associated with the annotation.

A human creates annotations using experience with natural language and personal discretion. We conducted a preliminary experiment in which a group of humans was instructed to create two categories of annotation over the same extract of fiction text. No two extracts were annotated in the same way. We speculate that the differences are caused by varying interpretations of the annotation categories and the fiction text. This implies that annotations cannot be objectively verified, as verification is subjective to the human performing the assessment.

Fiction text should be annotated in a manner that a single human considers correct. This chapter investigates a machine learning mechanism that uses examples provided by that human to learn a model for creating similar annotations. Annotations occur in many different categories, each of which are qualified by different combinations of text-references and semantic concepts. The machine learning system should be capable of deriving models in any category, and for any type of qualifier.

4.1.3 Context

The research presented in this chapter examines the problem of automating the text-analysis task of the fiction-to-animation process. The context of this problem within the conversion process is illustrated in Figure 4.3. We develop a method that automates the creation of annotations over fiction text, resulting in an intermediate representation. The input to this process is fiction text annotated with surface annotations, the processes for which are described in Chapter 3. Semantic annotations are created based on example annotations provided by a human in conjunction with these surface annotations.

4.2 Related work

The creation of semantic annotations over fiction text is similar to the task of information extraction. In both cases, fragments of text are identified as belonging within a certain category.

Information extraction is concerned with filling a structured template rather than marking up relevant portions of the text. A template consists of a number of slots that are labeled to indicate the kind of information that can fill them (Chinchor and Marsh, 1998). For example, a typical information extraction task is concerned with automatically obtaining details about a "seminar" from an unstructured text announcement. A template consists of slots that include "title", "venue", and "speaker"; while an information extraction process fills these slots for each text-based document that announces a seminar (Califf and Mooney, 2003).

Subsequent interpretation of the text fragments, including anaphora resolution and co-reference resolution, are performed as a post-processing phase of the information extraction task (named *discourse processing* (Soderland, 1997)). This interpretation is analogous to providing data for semantic-concepts defined in an annotation.

Information extraction methods are developed for a specific subject domain, and it is possible for domain-experts to hand-craft sets of extraction patterns for locating important fragments of text from documents within that domain. We experimented with this approach in identifying the *speech-verb*, *actor* and *speaker* of a quote in fiction text (Glass and Bangay, 2007c). The hand crafted rules produce accurate annotations, but the creation of the rules require many hours of specialized labour, and can not be applied to different categories of annotation.

We investigate machine learning techniques that perform the information extraction task by automatically inducing patterns from example data. These methods remove the need for specialized labour in creating extraction patterns and are adaptable over different categories of data.

4.2.1 Machine learning for information extraction

A machine learning algorithm generates a *model* or *theory* regarding how to identify text fragments that fill a particular slot in a template. A model is created using training examples (supervised learning), which are documents containing marked-up text-fragments that are guaranteed to be correct for the particular slot. Two categories of techniques exist for creating and representing a model, namely classifier-based techniques and pattern- or rule-based techniques (Turmo et al., 2006).

Classifier-based techniques use statistical models to determine whether a certain fragment of text should be used to fill a particular type of slot. These techniques determine the probability that a fragment of text is a slot-filler based on the types of *features* associated with the fragment. Features consist of meta-data regarding a particular item of text, for example the part-of-speech, or the syntactic relation between two words. Numerous classifier-based learning techniques exist for information extraction based on alternative statistical strategies and feature sets (Freitag, 2000; Freitag and McCallum, 2000; Chieu and Ng, 2002; Chieu et al., 2003; Bunescu and Mooney, 2004). The CarSim system (Johansson et al., 2005) makes use of statistical machine learning for the information extraction task within the text-to-animation process.

Pattern-based techniques for information extraction are concerned with the induction of a set of patterns that are able to distinguish fragments of text belonging to a particular category. Patterns contain *constraints* that must be satisfied by a portion of text before a text fragment can be extracted (Soderland, 1997). An example of a constraint is a sequence of tokens that precede or follow the fragment of text to be extracted. Patterns also contain a *slot-filler* that, assuming the constraints are satisfied, defines what fragment of text should be extracted to fill a specific slot. An example pattern "<victim> was murdered" (Califf and Mooney, 2003) contains "was murdered" as a constraint. If a similar string is found in the input document, the constraint is satisfied and the token appearing just before the constraint is extracted as the filler of the "victim" slot.

We use a pattern-based method for performing machine learning, but recognize that a classifier-based method also has the potential to perform the task of creating annotations. A pattern-based technique is chosen because it provides direct control over the models induced for categories of annotation. The link between the chosen set of features and the success of a learned model is unclear when using statistical techniques (Johansson et al., 2005). Patterns are also created using a very small set of examples, while statistical models require larger corpora of training data, a resource that is unavailable in the context of semantically annotated fiction text.

4.2.2 Techniques for pattern induction in information extraction

Pattern induction involves the construction of patterns that identify each example in a set of training data (Riloff, 1993, 1996).

Patterns created directly from examples are potentially over-specific, therefore limiting their applicability to only a few examples. Pattern induction algorithms employ *generalization* so that the pattern applies to more than a single example (Chai et al., 1999; Harabagiu and Maiorano, 2000). A single pattern can be generalized in a number of different ways, and a scoring technique is required to select what generalized pattern to use (Basili et al., 2000).

Algorithms that perform generalization are categorized into two groups, those performing *compression* and those improving *coverage* (Califf and Mooney, 2003). Alternatively, algorithms are categorized in terms of the direction of pattern creation, namely *top-down* and *bottom-up.* Another distinction between information extraction algorithms is the type of text over which they are designed to function, categories for which include *free, semi-structured* and *structured text.*

4.2.2.1 Generalization method: compression versus covering

Algorithms that perform compression begin with a set of highly specific patterns, usually one for each example. A more general pattern is constructed that replaces a number of specific patterns, but is still able to cover all the input examples (where *cover* means that a pattern successfully identifies the correct element in an example). This process continues until no further compression is possible (Califf and Mooney, 2003). General patterns are found by enumerating different generalized versions of the pattern, evaluating each on the example set, and selecting the generalization that covers the most examples correctly. Alternatively, two patterns are *merged*, creating a more generalized pattern that covers both the original patterns (Freitag, 2000).

Covering algorithms begin with a set of examples from which a single pattern is derived at a time. All of the examples covered by the newly created pattern are then removed from the starting set. New patterns are created until no more examples remain in the starting set (Soderland, 1997; Freitag, 1998). The new pattern created is either highly specific or generalized. If more than one correct generalization is created for a training example, then the pattern providing the best coverage over the start set is chosen (Ciravegna, 2001).

4.2.2.2 Direction of rule creation: top-down versus bottom-up

The set of examples provided for training a model is divided into *positive* and *negative* examples (Quinlan, 1986). Positive examples contain fragments of text correctly marked for a specific slot, while negative examples contain no such markings. Both are provided to induce patterns that correctly cover all positive examples while correctly rejecting all negative examples. This categorization of example data differentiates between two approaches for pattern induction, namely top-down and bottom-up approaches.

Top-down techniques begin with a general pattern that covers all examples (including both positive and negative training examples). This pattern is iteratively specialized with the aim of covering more positive examples, while rejecting negative examples (Quinlan, 1990; Soderland, 1999; Freitag, 2000; Turmo and Rodriguez, 2002; Déjean, 2002).

Bottom-up techniques begin with a pattern that explicitly describes one example in the set. Constraints within the pattern are iteratively generalized, with the aim of increasing the number of positive examples described by the pattern, while minimizing the coverage over negative examples (Soderland, 1997; Aseltine, 1999; Ciravegna, 2001; Califf and Mooney, 2003; Català et al., 2003). Generalization occurs in different forms, such as comparing two patterns and replacing two dissimilar portions of the constraints with an abstract concept that describes both, or by replacing the constraint with a wild-card (Califf and Mooney, 2003).

4.2.2.3 Type of input text

The type of input text for information extraction tasks is categorized into three classes, namely structured text, semi-structured text, and free text (Muslea, 1999; Turmo et al., 2006).

Structured text presents the simplest task for information extraction techniques, because formatting markers in the text clearly indicate the function of a fragment of text. Example data includes weather forecasts obtained from web-pages, and extracts from telephone directories (Soderland, 1999).

Semi-structured text exhibits structural markers, although these do not always conform to one single pattern. Examples include seminar announcements where, for example, the seminar title and speaker is generally indicated in a consistent manner (but are not marked up in any way) (Soderland, 1999; Ciravegna, 2001; Califf and Mooney, 2003). Other examples include job-vacancy announcements (Ciravegna, 2001; Califf and Mooney, 2003) and rental advertisements (Soderland, 1999).

Free text presents the most challenging task for information extraction where no structural detail exists in the input text, a characteristic that is exhibited by fiction text. Example test corpora include terrorist reports sourced from newspapers, in which slots to be filled from each report include the entity attacked and the attacking entity (Kim and Moldovan, 1995). Other examples include reports on management changes of companies (Huffman, 1996; Soderland, 1997; Sudo et al., 2003), and reports of disease outbreaks (Patwardhan and Riloff, 2007; Phillips and Riloff, 2007).

System	Test data	Precision	Recall	F-measure
TIMES (Chai et al., 1999)	Job listings (Company name, position, salary, location, experience, contact, skills, benefits)	40.4% - 96.8%	53.1% - 83.8%	47.5% - 84.5%
WHISK (Soderland, 1999)	Management succession (person in, person out, organization, post title)	48.5% - 68.9%	46.4% - 61.0%	-
EXDISCO (Yangarber, 2003)	Management succession (person in, person out, organization, post title)	Approx. 62% - 89%	Approx. 10% - 90%	-
CRYSTAL (Soderland, 1997)	Management succession (person in, person out, organization, post title)	Approx. 63% - 76%	Approx. 50% - 75%	-
LIEP (Huffman, 1996)	Management changes	89.4%	81.6%	85.2%
EVIUS (Turmo and Rodriguez, 2002)	Colour	98.77%	88.89%	93.57%
ESSENCE (Català et al., 2003)	Aircraft crash reports (crash site, crash date, aircraft, airline, departure, destination)	51.2% - 100.0%	39.5% - 75.4%	48.4% - 78.8%
WAVE (Aseltine, 1999)	Latin-American terrorist reports/hospital discharge summaries	-	-	Approx. 49% - 58%

Table 4.1: Reported performance of related pattern-based information extraction techniques over free text.

Few implemented pattern-based information extraction systems are evaluated using a standardized test corpus. This makes comparison between systems impossible. Different evaluation methods are also used in related research, with respect to the number of examples used for testing and the number of test cases evaluated. Despite these differences, reported performance results over a range of information extraction systems over free text are listed in Table 4.1. This table indicates that precision levels between 40% and 100% are achieved, with recall levels ranging between 10% and 90%. We quote these figures with caution, because the various tasks are not directly comparable, and neither are the various slot categories within each task. These figures highlight that information extraction over free text is a complex task for which no single technique can be established as superior. Turmo et al. (2006) present performance results for a range of systems over semi-structured texts which, as expected, tend to be higher than over free-text.

4.2.3 Hierarchical rule-based learning as a pattern induction mechanism

We present hierarchical rule-based learning for inducing and generalizing patterns from example annotations. This approach uses a bottom-up compression algorithm, where highly specific patterns are created for each positive and negative example in the training set. Patterns are merged one pair at a time, until no further merges are possible.

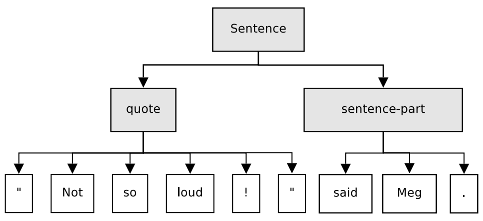
	Text structure			Generalization			Direction	
Pattern-based techniques:	Free	Semi-Structured	Structured	Compression	Coverage	Other	Top-down	Bottom-up
Palka (Kim and Moldovan, 1995)	✓					✓	-	-
AutoSlog (Riloff, 1993, 1996)	✓					✓	-	-
Liep (Huffman, 1996)	✓			✓	✓			✓
Crystal (Soderland, 1997)	✓				✓			✓
Wave (Aseltine, 1999)	✓				✓			✓
Times (Chai et al., 1999)	✓					✓	-	-
(Basili et al., 2000)	✓					✓	-	-
Snowball (Agichtein and Gravano, 2000)	✓				✓			✓
(Harabagiu and Maiorano, 2000)	✓					✓	-	-
Essence (Català et al., 2003)	✓				✓			✓
ExDisco (Yangarber, 2003)	✓				✓			✓
Hierarchical rule-based learning	✓			✓				✓
Whisk (Soderland, 1999)	✓	✓	✓		✓		✓	
Rapier (Califf and Mooney, 2003)		✓		✓				✓
Srv (Freitag, 1998, 2000)		✓			✓		✓	
$(LP)^2$ (Ciravegna, 2001)		✓			✓			✓
Evius (Turmo and Rodriguez, 2002)	✓	✓			✓		✓	

Table 4.2: Pattern-based learning systems summarized according to text structure, generalization strategy and direction of generalization.

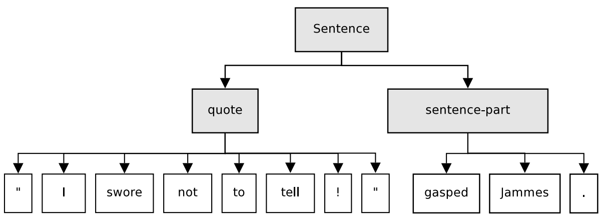
Hierarchical rule-based learning is compared with related research in Table 4.2, in which pattern-based machine learning systems are categorized according to the type of text used as input, the generalization method used, and the direction of the pattern creation. Our system is one of the few bottom-up compression algorithms designed for use with free text.

The benefit of the hierarchical rule-based system is the ability to learn many categories of annotation without modifying the internal learning mechanism. Rather, the structure and contents of a pattern are manually customized for different annotation categories, although future work includes the possibility for automating the customization of pattern structures (similar to work by Shinyama and Sekine (2006)). These patterns provide the ability to incorporate both structural elements of language (sentence, phrase, quote, and token) as well as syntactic information (parts-of-speech and syntactic function). Patterns also provide the ability to identify semantic data, allowing for the semantic-concepts of annotations to be identified. These concepts are learned in a knowledge-poor fashion, that is in the absence of external knowledge-bases (as opposed to some existing approaches (Stevenson and Greenwood, 2005; Li and Bontcheva, 2007)).

Existing corpora for evaluating information extraction techniques do not contain marked-up visual descriptions. These corpora are also created for testing the template-filling ability of an automated process. Annotations are different because they identify *every* fragment of text in a category as opposed to identifying *one* term that fills a slot. This makes existing test data inappropriate for evaluating hierarchical rule-based learning with respect to the fiction-to-animation task.



(a) "Not so loud!" said Meg.



(b) "I swore not to tell!" gasped Jammes.

Figure 4.4: Structural elements of a sentence arranged as a tree.

Instead, we evaluate hierarchical rule-based learning by performing training and evaluation over a custom-built corpus of fiction text marked-up with semantic annotations in visual categories.

4.3 Induction of rules for creating annotations

This section develops the concepts for hierarchical rule-based learning. We motivate the concept of tree-based patterns, and indicate how they are induced, generalized and applied to create annotations in fiction text. Detailed algorithms for each step in the process are deferred to Section 4.4, which has the same structure as this section for convenient reference.

Semantic annotations identify visual descriptions in free text, a source that is considered to be the least structured in the field of pattern-based information extraction. We argue that structural elements exist in free text that can be used to construct patterns for identifying annotations. Structural elements include units of text such as a sentence, a quote, and a token. Not only do these elements indicate structure, but they are also related in a hierarchical fashion, where certain elements encapsulate other elements. Consider the following example from *The Phantom of the Opera* by Gaston LeRoux (1911)[2]:

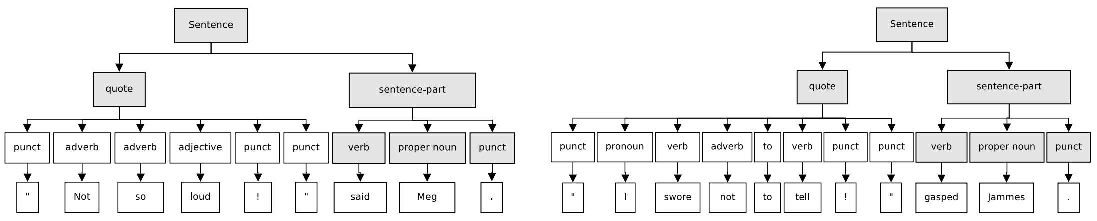
```
“Not so loud!” said Meg.
```

This example exhibits the following structural elements (each element is indicated using square brackets):

```
Sentence:               [“ Not so loud ! ” said Meg .]
Quote/Sentence-part:    [“ Not so loud ! ”] [said Meg .]
Token:                  [“] [Not] [so] [loud] [!] [”] [said] [Meg] [.]
```

The above example indicates that the sentence contains two structural elements, namely a quote and a partial sentence. The quote in turn contains a number of tokens. This idea of containment suggests that the original sentence can be abstracted on different levels using a tree structure to indicate the relationship between elements, illustrated in Figure 4.4(a). Abstracting text on a number of levels in this manner presents an opportunity for identifying patterns in text. This is illustrated by the similar structure exhibited in Figure 4.4(b) where the trees are abstracted (highlighted), but the tokens are entirely different.

[2]All subsequent examples in this section are taken from this source, and are possibly modified for illustrative purposes.



(a) "Not so loud!" said Meg. (b) "I swore not to tell!" gasped Jammes.

Figure 4.5: Additional syntactic abstractions within the tree structure.

Patterns need not contain only structural abstractions, but can also include syntactic abstractions. For example, the part-of-speech of a token presents an abstraction of the token, while a phrase type abstracts a group of tokens. The inclusion of syntactic structures is illustrated in Figure 4.5. The similar portions of the two trees are highlighted, indicating a pattern with finer detail to that in Figure 4.4.

The above examples show that hierarchical patterns exist in free text. We present a method for automatically inducing these patterns with the purpose of learning how to create a given category of annotation.

This section describes the full process of creating patterns from annotations, combining these to form a general model regarding a specific category of annotation, and applying a set of patterns to text for the creation of new annotations. The algorithms for performing these functions are formally presented in Section 4.4.

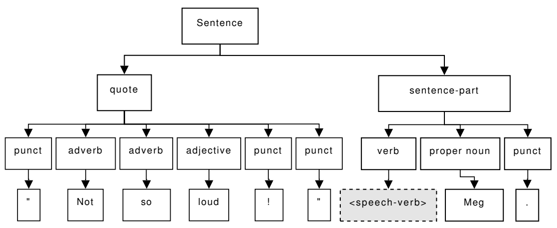
4.3.1 Hierarchical patterns and rules

Free text has the potential to exhibit a variety of patterns. Given a set of manually created annotations over fiction text, we are interested in inducing patterns that are able to identify portions of text as belonging to a particular category of annotation. Consider the following two annotated examples:

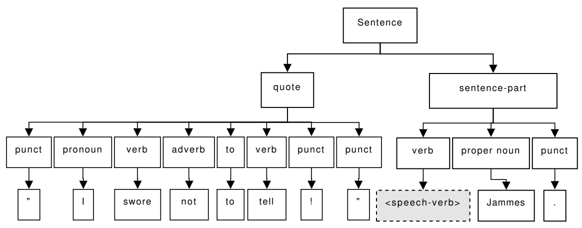
```
<quote speech-verb="said" actor="Meg" speaker="MEG">"Not so loud!"</quote> said
Meg.

<quote speech-verb="gasped" actor="Jammes" speaker="JAMMES">"I swore not to
tell!"</quote> gasped Jammes.
```

For the sake of demonstration, we consider the task of identifying the *speech-verb* of the above quote annotations. Hierarchical trees derived from these sentences are presented in Figure 4.5. Trees become *rules* when they are able to identify portions of text that belong in a specific category. Rules are created from the annotated example above, because the annotation indicates which token functions as the *speech-verb* of the quote, illustrated in Figure 4.6. In information extraction terms, all nodes in the tree that are not highlighted represent the "constraint" portion of the rule, while the highlighted node indicates the location in which to create the annotation. We name this the *answer* of the rule, and annotations containing answers are *positive examples.* Rules derived from these examples are named *positive rules.* Answer nodes are the equivalent of the "slot-filler" in information extraction, and act as a wild-card because any data can correspond to this node.



(a) "Not so loud!" said Meg.



(b) "I swore not to tell!" gasped Jammes.

Figure 4.6: Example of *rules* indicating a the location of a *speech-verb* in a sentence.

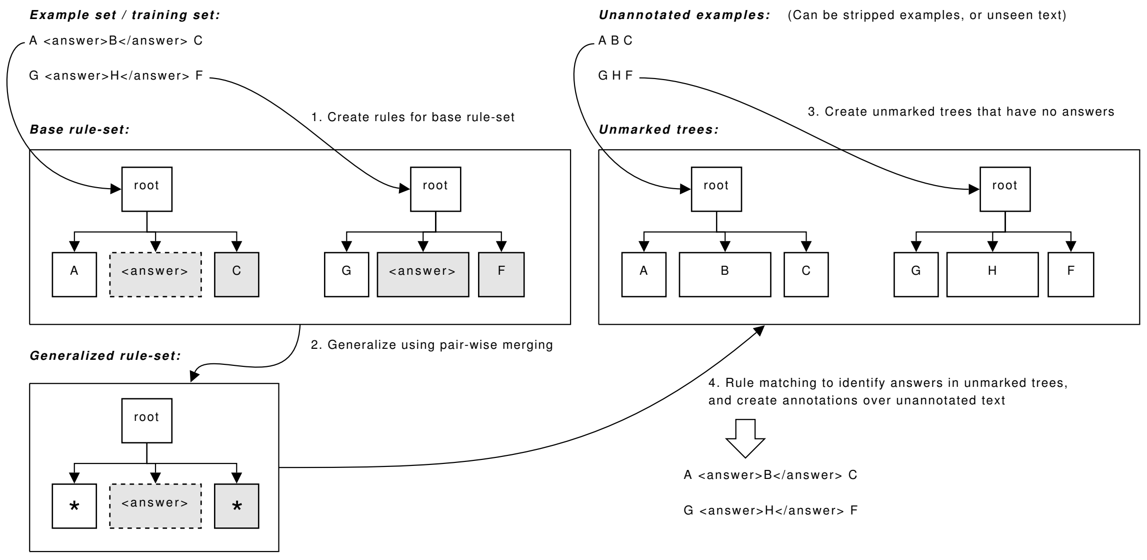

Example set / training set:
A <answer>B</answer> C
G <answer>H</answer> F
1. Create rules for base rule-set
Base rule-set:
root
A
<answer>
C
root
G
<answer>
F
2. Generalize using pair-wise merging
Generalized rule-set:
root
*
<answer>
*
Unannotated examples: (Can be stripped examples, or unseen text)
A B C
G H F
3. Create unmarked trees that have no answers
Unmarked trees:
root
A
B
C
root
G
H
F
4. Rule matching to identify answers in unmarked trees, and create annotations over unannotated text
A <answer>B</answer> C
G <answer>H</answer> F

Figure 4.7: Different components of hierarchical rule-based learning.

Rules can also indicate when to avoid creating an annotation. In this case, an example is provided by a human that contains no annotation. A tree is constructed containing no answer node. We name these *negative examples*, and their corresponding trees *negative rules.* In general, we define a rule as a tree pattern that is created from an example explicitly provided by a human.

We construct trees so that the leaf nodes represent individual tokens in the free text, and the root node represents a single global abstraction of the input (usually choosing the arbitrary term “root” as the type for the root node). The levels of abstraction between the leaf nodes and the root are customized according to a specific category of annotation.

The different components of hierarchical rule-based learning are illustrated in Figure 4.7. The induction of rules from annotated text begins by creating a *base rule-set* that contains one rule per example in the *example-set.* The base rule-set is generalized to create a *generalized rule-set*, which is capable of reproducing the original examples and produce further annotations. To test the generalized rule-set we strip the original examples of their annotations to create *unannotated examples*, from which *unmarked trees* are created (unmarked because they do not contain answers). Rules in the generalized rule-set are *matched* with each unannotated example, and annotations are created where matches occur.

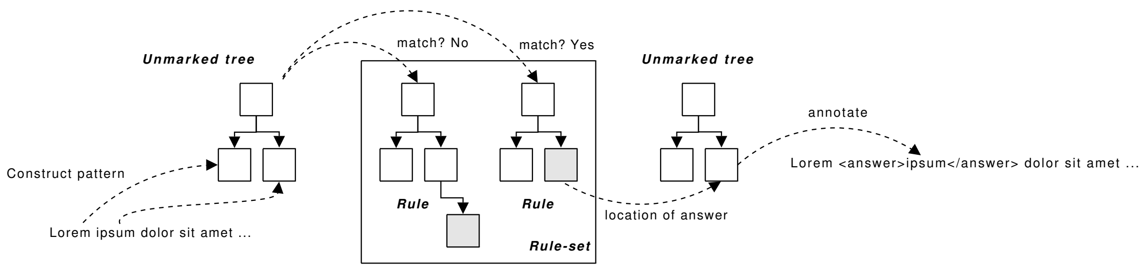


Figure 4.8: Relationship between text and rules through the application of a rule-set.

4.3.2 Rule-set creation

The goal of hierarchical rule-based learning is to construct a *model* regarding the creation of a specific category of annotation over free text. This model is represented by a *rule-set*, which is a collection of rules. The pair of rules presented in Figure 4.6 is considered a rule-set because it is a set of rules derived from manual examples with the explicit purpose of identifying the *speech-verb* of a quote.

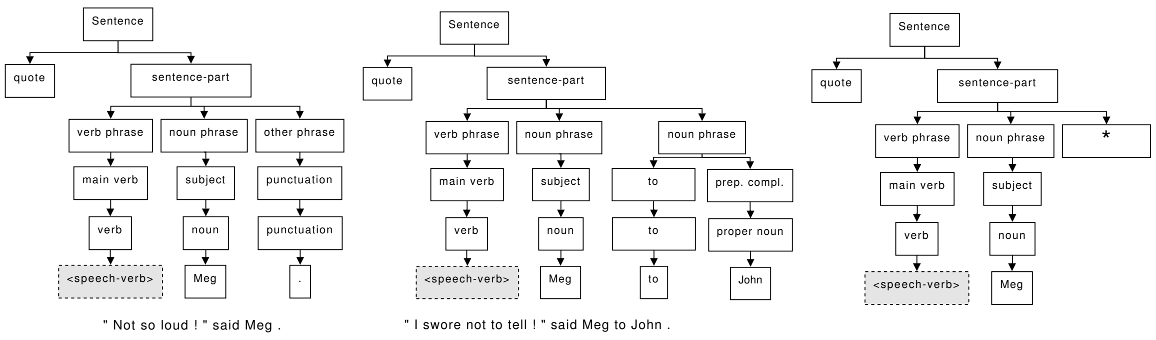
The relationship between annotations and a rule-set is illustrated in Figure 4.8. A new annotation is created by constructing a tree for an unannotated example (called an *unmarked tree*), and matching this tree with every rule in the rule-set. If all the constraint nodes in a particular rule are identical in the unmarked tree, then the node in the unmarked tree in the same location as the *answer* node in the rule is marked as the answer. The token in the unannotated example corresponding to this node in the unmarked tree is then annotated. If no matching rules exist in the rule-set, then the example is not annotated.

Rule-sets are constructed by creating a tree pattern for every example provided by a human. We expect that a human creates annotations by manually marking-up contiguous extracts of fiction text. Each extract is automatically sub-divided into positive and negative examples, and a tree is created that represents each example. Free text is sub-divided using clear structural features such as tokens, sentences, and quotes (each of which are automatically identified using methods described in Chapter 3). The structural feature to use for sub-division depends on the category of annotation.

Assume that rules are to be created for *object* annotations and that for this category free text is sub-divided into tokens. In this case, every token represents an example, and a rule is constructed for every example token in the text. Tokens not annotated as an *object* create negative rules, while tokens that are annotated create positive rules.

A rule is created for each example in the input extract, the result of which is a *base* or *start rule-set.*

The constraint portions of the rules in Figure 4.6 are highly specific, the consequence of which is that they match very few sentences other than those responsible for their construction. The trees illustrated in Figure 4.6 exhibit portions that are identical, forming a pattern common to the two rules. This observation implies that a generalized rule can be derived by comparing trees, and keeping only those portions that are similar. This idea is discussed further in the following section.



(a) Rule 1 (b) Rule 2 (c) Generalised rule

Figure 4.9: Examples of rule generalization.

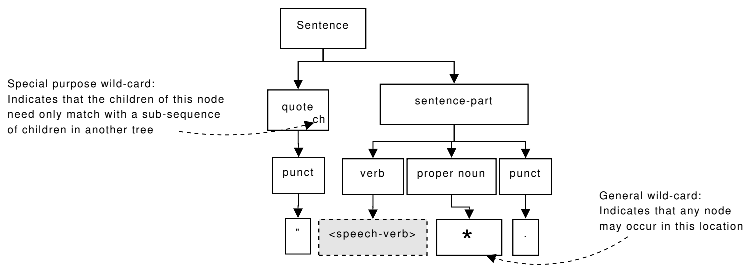

Sentence
Special purpose wild-card:
Indicates that the children of this node
need only match with a sub-sequence
of children in another tree
quote
ch
sentence-part
punct
verb
proper noun
punct
General wild-card:
Indicates that any node
may occur in this location
"
<speech-verb>
*
.

Figure 4.10: Example generalized rule containing general and special-purpose wild-cards.

4.3.3 Rule generalization

Consider the example rules presented in Figure 4.9(a) and (b). A large portion of both these trees is identical, with the exception of the existence of an additional phrase in rule (b). A single rule can be created that represents both rule (a) and rule (b) by replacing the different portions of the tree with a wild-card, as shown in Figure 4.9(c). In this example, the highest differing node in the tree is replaced with a wild-card, removing an entire sub-tree. The new tree is generalized, because it has the potential to match many more trees than the two responsible for its creation.

The rules presented in Figure 4.6 also contain similar patterns, and are different in two portions of the tree. The nodes indicating “Meg” and “Jammes” are different, and these are generalized by replacing the nodes with a wild-card. The sub-trees headed by the “quote” nodes are also different. However, we do not wish to replace the “quote” nodes with a wild-card, because this would remove an element that is common to both trees. The child nodes of “quote” cannot be replaced by wild-cards because the rule in Figure 4.6(a) has six children while the rule in Figure 4.6(b) has eight children, and no decision can be made regarding the number of wild-cards to insert. To solve this problem, special-purpose wild-cards are specified in nodes when only a sub-sequence of the children are common. The rule generalized from the two rules in Figure 4.6 is illustrated in Figure 4.10, and contains the two types of wild-card.

The special-purpose wild-card has the ability to produce a number of generalization options. This is because numerous sub-sequences of nodes have the potential to be common between two rules. This is illustrated in Figure 4.11, in which four different options exist for generalizing the two trees. In all cases the “root” node is common, but the children contain numerous sub-sequences of

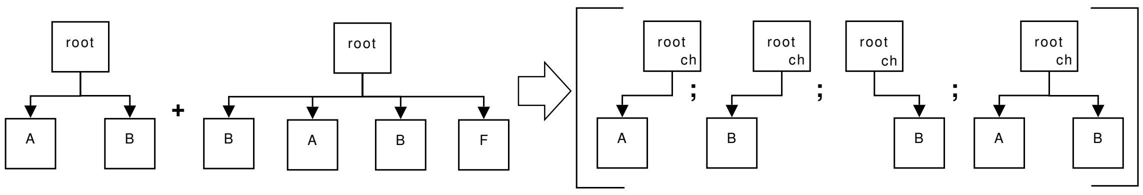


Figure 4.11: Multiple options for generalization when using a special-purpose wild-card.

matching nodes. The choice of which generalized rule to accept depends on whether the generalized rule still contains an answer (that is, it has not been removed by generalization).

Rule generalization results in the removal of portions of a tree from a rule, but this must be done with care to prevent the removal of answer nodes. Other nodes in a tree might also be critical to the structure of a rule and should never be removed in the generalization process. We call nodes with this characteristic *preserved* nodes. All answer nodes are preserved, but preserved nodes need not necessarily be answers. Preserved nodes must never be removed from a rule and as a result, a wild-card can only be used if it does not result in the removal of a preserved node from the tree.

The idea of comparing two rules and removing dissimilar portions of the trees is named *merging*, because we replace two individual rules with one general rule that represents both. In some cases, a pair of rules contains no similar patterns, in which case a merge is impossible. Merges are also impossible if two rules contain a similar pattern, but this pattern does not contain all the preserved nodes in both rules.

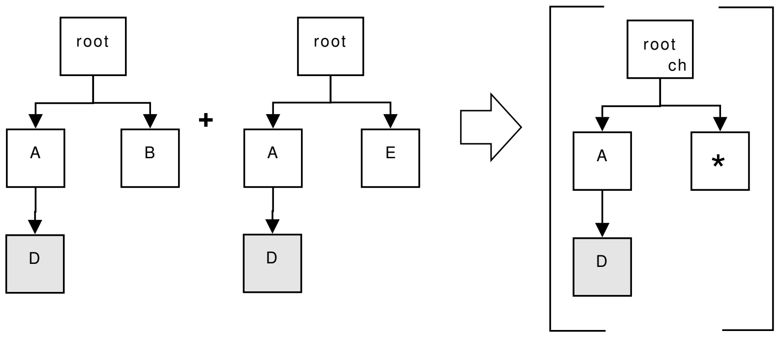
Example merge scenarios are presented in Figure 4.12 to illustrate the concept of successful merging and merge failure. A merge resulting in the insertion of a general wild-card is illustrated in Figure 4.12(a). A merge in which multiple sub-sequences of children are common resulting in two merged rules is illustrated in Figure 4.12(b). Additional patterns exist that are common to these two rules, but none of these contain the preserved node "D", and so are discarded as viable generalization candidates. A merge failure is illustrated in Figure 4.12(c), where no common pattern exists between two rules that does not remove a preserved node.

4.3.4 Application of a generalized rule

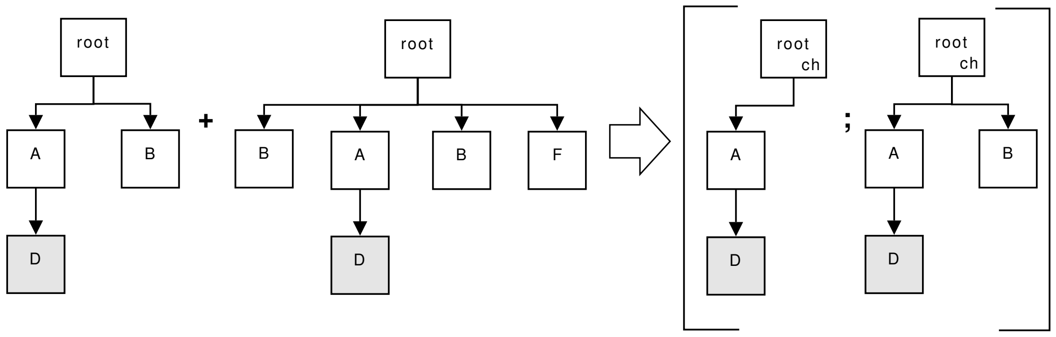
A generalized rule has the ability to match correctly with trees other than those from which it was created. Assume the *speech-verb* is to be located for the following sentence containing a quote:

```
“Where are you?” asked Paul.
```

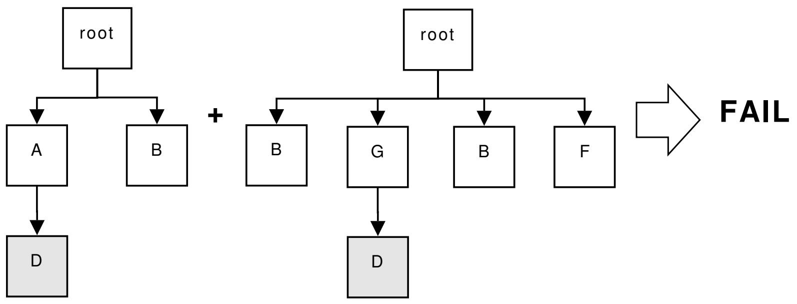
A tree is constructed from this unannotated sentence, illustrated in Figure 4.13. This tree does not match with either of the rules in Figure 4.6, because it contains a number of different nodes. However, the generalized rule derived from these two rules (shown in Figure 4.10) matches with this tree because of the presence of wild-cards. The token "asked" is annotated as a *speech-verb* because the match is successful, and the node containing "asked" is in the same location as the "speech-verb" node in the rule.



(a) Successful merge, demonstrating insertion of a general wild-card.



(b) Successful merge with common child sub-sequences and insertion of special purpose wild-card.



(c) Failed merge, due to the lack of common patterns.

Figure 4.12: Example merges between pairs of rules.

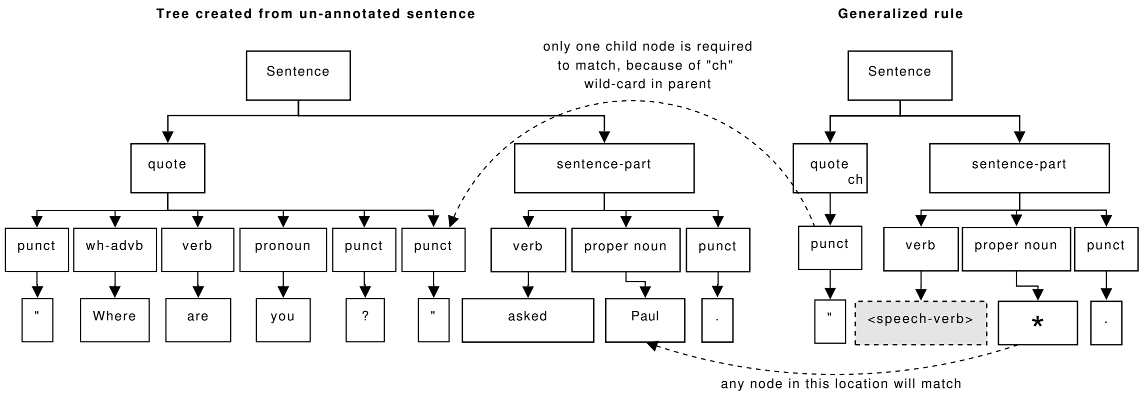


Figure 4.13: Matching between a generalized rule and a tree generated from an unannotated sentence.

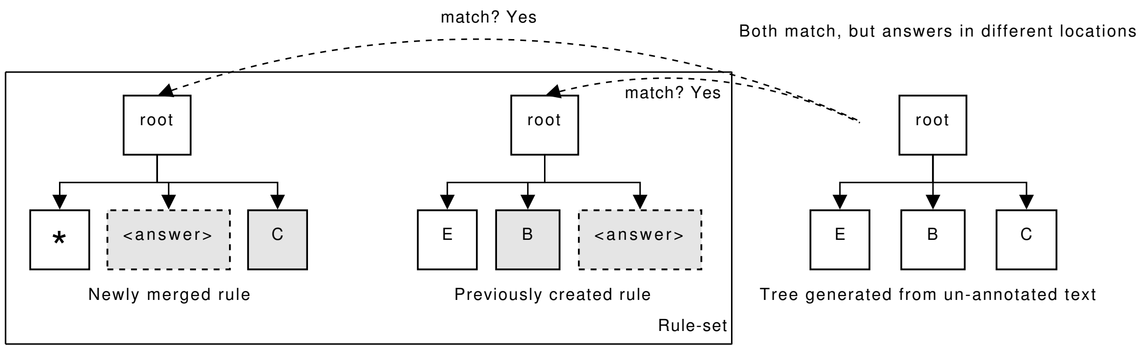


Figure 4.14: Example of conflicting rules in a rule-set.

4.3.5 Generalization of a rule-set

An annotator presents a set of examples from which a base rule-set is created. Initially, this rule-set is over-specific and applies only to the examples provided. This set of rules must be generalized so that it has the ability to apply to *unseen text*, that is, new text for which annotations must be created.

If two rules can be generalized by identifying common patterns in rules and removing dissimilar portion, then the base rule-set can be generalized in a similar manner. However, the more rules compared at once, the less likely that a common pattern exists in the trees that also contains all the preserved nodes in the same locations. Alternatively, we propose an iterative process in which pairs of rules are merged at a time until no further merges are possible. Using this method, generalized rules are likely to be merged multiple times before the process ends.

Given that pairs of rules are merged at a time, the question remains as to which pairs of rules to merge at each iteration. One approach is to merge every pair of rules in the set, and choose the merged rule that covers the most examples (Glass and Bangay, 2006).

The rule-set generalization process is characterized by the problem illustrated in Figure 4.14, namely that a newly merged rule has the potential to *conflict* with an existing rule in the set. A pair of rules conflict if both can be matched successfully to a tree created from an example sentence, but which indicate answers in different locations. In Figure 4.14 both rules in the rule-set match the unannotated tree, yet indicate different answers. If the incorrect rule is chosen, then an incorrect annotation is created. A conflict can also occur between a positive and a negative rule, where both match an example, but where one rule indicates an answer and the other does not.

Conflicting rules are prevented by matching a newly merged rule with every individual rule in the rule-set. Rules are trees, and can be matched against one another. If a successful match occurs, but the answers are in different locations in the trees then a conflict is found, and the new merged rule is deemed invalid.

Matching between two rules is different to regular matching because both rules contain answer nodes and wild-cards. If regular matching is used, the two rules in Figure 4.14 do not match, but they both match the unannotated tree. However, if the answer nodes are interpreted as wild-cards (as opposed to a literal string "<answer>") then the two trees match, but point to different answers and a conflict is detected. This is a conservative method because possible conflicting rules

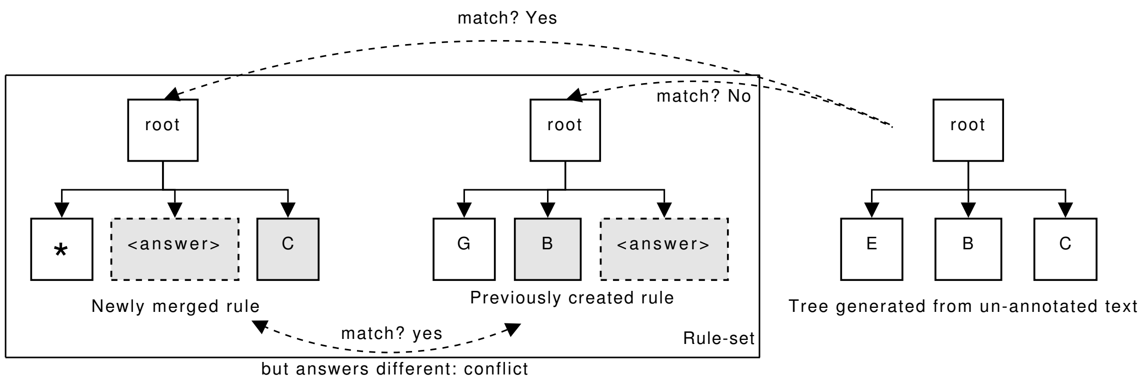


Figure 4.15: Conflict detected using conservative matching between rules.

are avoided, even if no conflict exists within the example data. This point is illustrated by Figure 4.15, in which the second rule does not match the unannotated tree, in which case there is no conflict. However, a positive match still exists between the rules when interpreting the answer nodes as wild-cards, and the new merged rule is rejected.

The rule-set generalization process consists of selecting an arbitrary pair of rules and merging them. If the merge does not fail, then a number of merge candidates are presented. Merge candidates that remove preserved nodes are discarded. The remaining rules are compared with every rule in the rule-set to detect if a conflict occurs, and are discarded if this is the case. A merged rule is selected from the remaining set of merge candidates and replaces the pair of original rules in the rule-set (selection strategies are described in Section 4.4.5). This process continues until no pair of rules can be found that results in a valid merge.

4.3.6 Rule-set application to unseen text

Unseen text is sub-divided into structural units in the same manner used for creating a rule-set, where a tree is constructed for each unit (see Section 4.3.2). Every rule in the generalized rule-set is matched against the new tree, and if a match is found, the answer in the tree is located and the appropriate annotation created. If a negative rule matches the tree, or if no matching rule is found, then no annotation is created for the unseen example.

We assume that conflicting rules do not exist in the rule-set as a result of the discussion in Section 4.3.5. In this respect, the first matching rule is assumed to point to the same answer as any subsequent matching rule that potentially exists in the rule-set, and therefore the search ends when any matching rule is located.

A base rule-set is the most constrained set of rules representing the examples provided by a human. Assume that the same set of examples is used to create a set of unmarked trees, then every rule in the base rule-set should be identical to exactly one unmarked tree. In this manner, annotations in the example data are reproduced by applying the base rule-set to the example extract.

If an incorrect rule from the base rule-set matches with an unmarked tree then the result is an erroneous annotation. To avoid this scenario, rules in a base rule-set must be sufficiently constrained to match only a single unmarked tree (the degree to which a rule is constrained depends on the category of the rule, as discussed in Section 4.5). However, if sufficient constraints cannot be defined for a particular category of annotation, we make the following assumption:

Assumption 4.2. *(Consistency Assumption) The base rule-set induced by a set of training examples produces a lower bound regarding the number of accurate matches with the unmarked trees (that represent the training examples).*

The Consistency Assumption (Assumption 4.2) provides for the detection of loss in annotation ability as a result of generalization. A generalized rule-set should not reduce the number of accurate matches below the lower bound achieved by the most specific base rule-set. An ideal generalization produces more accurate matches than this lower bound.

4.4 Algorithms for pattern induction and application

This section formally defines hierarchical rules and presents algorithms for the construction of a rule-set, the pairwise generalization of two rules, the generalization of a rule-set, and the application of a rule-set to unseen text.

4.4.1 Hierarchical patterns and rules

This section defines a structure for representing multiple levels of abstracted text, and for indicating where answers are located.

4.4.1.1 Structure of patterns: nodes and trees

Patterns derived from free text can be abstracted on multiple levels, where a concept on one level encapsulates zero or more concepts on a more specific level. We represent a single concept as a *node*, where encapsulation is represented by links between nodes. A node is defined as follows:

Definition 4.3. A *node* N represents a single concept that is an abstraction of zero or more specialized concepts. A node has the following properties:

- **Type:** An identifier for the concept that the node represents, or a wild-card ω indicating that the node represents any concept.
- **Parent:** A link to a single node representing a more abstract concept.
- **Children:** An ordered set of links to zero or more nodes that represent more specialized concepts.
- **Answer:** Indicates whether the concept represented by the node is manually marked as belonging to a particular annotation category.
- **Preserved:** Indicates whether the concept can be removed from the linked node structure. A value of *true* indicates that the node cannot be removed.
- **ChildSubSeq:** Indicates whether the concept is represented exclusively by its set of children, or whether the set of children represents only a sub-sequence of possible children. A value of *true* indicates the latter.

Each node contains a single link to an abstract concept, but contains multiple links to more specific concepts. A collection of nodes linked in this manner is called a *tree*, and contains one node that has no parent, called the *root*.

We use the following terminology to refer to certain aspects of a tree: the *ancestors* of a node N in a tree is the set of linked nodes between N and (including) the root of the tree; the *descendants* of a node N in a tree is a set of all nodes in the tree for which N is an ancestor; the *siblings* of a node N are all other nodes in the tree that have the same parent as N; the *depth* of a node N refers to the number of nodes between N and (including) the root (root is at depth 0); and a node that contains no children is referred to as a *leaf* node.

The induction of patterns common in a collection of trees is performed using a process of pair-wise merging, described in Section 4.4.3. This involves a node-by-node comparison of two trees, where nodes in the same *location* in a tree are compared. We define the concept of *location* in the following section.

4.4.1.2 Location and corresponding nodes

The comparison of two trees involves examining the properties of *corresponding* nodes. We define corresponding nodes in terms of the *location* of two nodes in their respective trees. Location is expressed in terms of a node's relative position to its siblings and ancestors.

Definition 4.4. Let the pair $\genfrac{}{}{0pt}{}{number}{siblings}$ define a node's relative position with respect to its siblings, where *number* is the position of the node in the ordered list of siblings, and *siblings* is the total number of siblings.

Two nodes in two different sequences are corresponding if their relative positions are identical. If the sequences contain different numbers of nodes, then the relative position of two nodes cannot be the same. However, this is not the case if the parent node N_P to one of these sequences has the property $childSubSeq(N_P) = true$. Assume N_P is a parent to n nodes but has the possibility of being a parent to additional nodes ($childSubSeq(N_P) = true$), and N_Q is a parent to m nodes ($m > n$). The relative position of each child node of N_P is defined in terms of the number of child nodes of N_Q. Relative position is expressed by the pair $\genfrac{}{}{0pt}{}{k + number}{*}$, given a choice of the value of k in the range $0 \leq k \leq m - n$.

Definition 4.5. *Location* of a node N is an ordered tuple that contains the relative position of every node from the root of the tree to the node N (in order of root to N). The number of pairs defining a location for node N is equal to $1 + depth(N)$. Nodes with identical ordered sequences of relative position tuples are said to be *corresponding*.

Location is demonstrated in Figure 4.16. The node E in the first tree has the location $\left\langle \genfrac{}{}{0pt}{}{1}{1} ; \genfrac{}{}{0pt}{}{2}{3} ; \genfrac{}{}{0pt}{}{1}{1} \right\rangle$, which is identical to the node J in the second tree, indicating that these are corresponding nodes. The node E in the first tree does not correspond to either leaf node in the third tree, because the location tuples are different.

Location accommodates instances where only a sub-sequence of children are explicitly defined for that node (where $childSubSeq(N) = true$). For example, in Figure 4.17 the node E corresponds

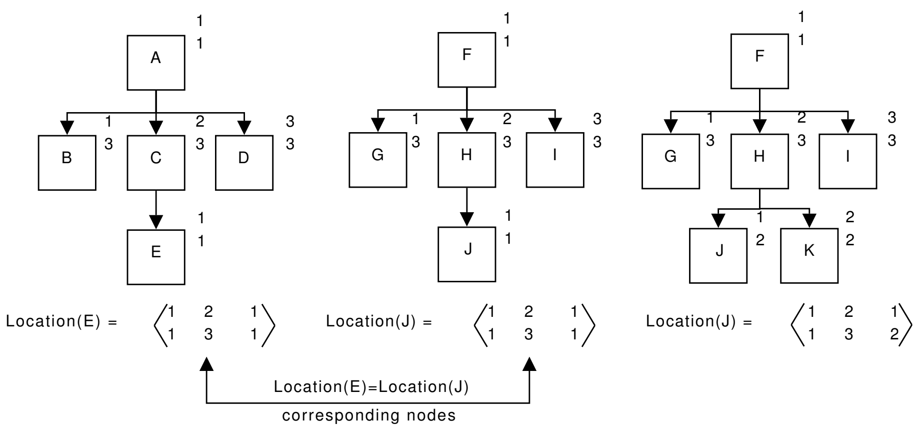


Figure 4.16: Example use of *location* to determine corresponding nodes.

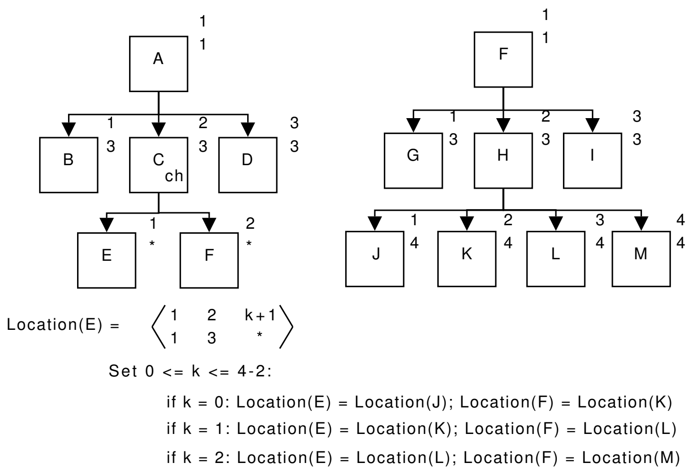

Location(E) =
Set 0 <= k <= 4-2:
if k = 0: Location(E) = Location(J); Location(F) = Location(K)
if k = 1: Location(E) = Location(K); Location(F) = Location(L)
if k = 2: Location(E) = Location(L); Location(F) = Location(M)

Figure 4.17: Interpretation of *location* when only a sub-sequence of child nodes are defined.

to node J if $k = 0$, but correspond to node K if $k = 1$. However, node E can never correspond to node M, because the upper limit of k is 2. In cases such as this, the term *corresponding* applies for any value of k in its defined range.

This definition allows nodes from two different trees to be referred to as corresponding. We also use the concept of *location* to determine whether two trees contain corresponding answer and preserved nodes.

4.4.1.3 Rules and unmarked trees

We define two types of trees: those that are constructed from examples provided by a human, and those that are constructed from text for which an annotation must be created (*unseen* text). These are defined as *rules* and *unmarked trees* respectively:

Definition 4.6. A *rule* is a tree representing a positive or negative annotation example provided by a human. A rule has the following properties:

- If a node N has the property $type(N) = \omega$ (node represents a wild-card type), then the node has zero children.
- Answer nodes are always preserved, that is any node N with the property $answer(N) = true$ also has the property $preserved(N) = true$.

Rules have the potential to contain answer nodes and preserved nodes, depending on the annotations in the input example. Rules also contain wild-cards and *childSubSeq* markers (if the rules have been generalized).

Unmarked trees represent text for which an annotation must be created, and therefore contain no answer or preserved nodes. Unmarked trees are never generalized and never contain wild-cards or *childSubSeq* markers:

Definition 4.7. An *unmarked tree* represents a unit of input that requires annotation. For every node N in an unmarked tree, the following properties hold:

- $type(N) \neq \omega$: No wild-cards are permitted in the tree.
- $preserved(N) \neq true$: No preserved nodes are permitted in the tree.
- $answer(N) \neq true$: The tree contains no answer nodes.
- $childSubSeq(N) \neq true$: Children cannot be a sub-sequence of the total set.

The use of the same tree structure for representing rules and unmarked trees has a number of advantages: the same method for constructing a tree can be employed regardless of whether a manually annotated example or an unseen unit of text is provided; corresponding nodes in rules and unmarked trees can be compared to determine whether the trees match; if the trees match then the node in the unmarked tree that corresponds to the answer node in the rule can be identified, and the corresponding text annotated as a result.

4.4.2 Rule-set creation

A rule-set is an ordered sequence of rules derived from an extract of annotated text. We denote an ordered sequence as follows:

$$\underset{\rightarrow}{S} = \langle s_1, ..., s_n \rangle$$

The concept of a rule-set is an example of an ordered sequence:

Definition 4.8. A rule-set $\underset{\rightarrow}{\mathbf{R}}$ is an ordered sequence of rules R_i, such that:

$$\underset{\rightarrow}{\mathbf{R}} = \langle R_1, ..., R_n \rangle$$

Algorithm 4.1 creates a single rule for every (positive or negative) example encountered in the input extract, the result of which is referred to as the *base rule-set.*

4.4.3 Rule generalization

This section describes an algorithm for creating a generalized rule from two input rules. We define a valid generalized rule in terms of the following criterion:

Algorithm 4.1 Rule-set creation.

```
create(in: extract $E$ containing a number of examples;
      out: rule-set $\underset{\rightarrow}{\mathbf{R}} = \langle R_1, ..., R_n \rangle$)
begin
  while $E$ has more examples do
    $e \leftarrow$next example in $E$
    construct rule $R$ from example $e$
    append $R$ to $\underset{\rightarrow}{\mathbf{R}}$
end
```

Criterion 4.9. *A valid generalized rule has the following properties:*

1. *All preserved nodes in the two input rules exist at the same location in the generalized rule.*
2. *The generalized rule matches both input rules.*
3. *The generalized rule indicates answers in the same location as the answers in both input rules.*

A valid generalized rule is only created if both input rules contain preserved and answer nodes in identical locations. If the two input rules fail to meet these pre-conditions then generalization fails for this pair of rules.

We present an algorithm that generalizes two rules by traversing the two trees node by node in a depth-first manner, comparing each pair of corresponding nodes. If the concept represented by both nodes is the same, then a duplicate node is created and inserted into a new generalized rule. If the nodes do not represent the same concept, then a wild-card is inserted instead and no further descendants of the nodes are compared. The generalization process fails if the insertion of a wild-card prevents preserved nodes in the input rules from being duplicated in the generalized rule.

The traversal through the two trees is accomplished using a recursive function called *merge* that compares pairs of nodes, and creates a new generalized tree:

Definition 4.10. Define the function $merge(N_1, N_2) \mapsto \{N^0_{merge}, ..., N^n_{merge}\}|FAIL$ to be a function that takes as input two trees with roots N_1 and N_2 and produces either a set containing merged trees N^i_{merge} or $FAIL$ if N_1 and N_2 cannot be merged.

The merge function operates in two stages. The first stage is concerned with comparing the two input nodes and creating a duplicate or wild-carded node for the generalized rule. The second stage is concerned with invoking a recursive call to the *merge* function for merging the children of the two input nodes.

4.4.3.1 Node comparison

Merging is initially considered between a pair of nodes, where two nodes are used as input to a process that generates a single, generalized node representing the two input nodes:

Definition 4.11. Define the function $combineNodes(N_1, N_2) \mapsto N_{combine}|FAIL$ to be a function that takes two nodes N_1 and N_2 as input, and produces either a new node $N_{combine}$, or $FAIL$ if the nodes cannot be merged. $FAIL$ results if N_1 and N_2 have the following properties:

- $isPreserved(N_1) \neq isPreserved(N_2)$
- $isAnswer(N_1) \neq isAnswer(N_2)$

Otherwise, $combineNodes(N_1, N_2)$ results in a new node $N_{combine}$ that has the following properties:

- Type:
  - if $type(N_1) = \omega$ or $type(N_2) = \omega$, then $type(N_{combine}) \leftarrow \omega$
  - if $type(N_1) \neq type(N_2)$, then $type(N_{combine}) \leftarrow \omega$
  - if $type(N_1) = type(N_2)$, then $type(N_{combine}) \leftarrow type(N_1)\ [= type(N_2)]$
- $children(N_{combine}) \leftarrow \langle\rangle$
- $isPreserved(N_{combine}) \leftarrow isPreserved(N_1)\ [= isPreserved(N_2)]$
- $isAnswer(N_{combine}) \leftarrow isAnswer(N_1)\ [= isAnswer(N_2)]$
- $childSubSeq(N_{combine}) \leftarrow false$

Assume that nodes N_1 and N_2 are to be merged. The *combineNodes* function (Definition 4.11) has the ability to create a wild-card, which means that the sub-trees of N_1 and N_2 are irrelevant to the generalized rule. This is incorrect if these sub-trees contain preserved nodes, which means that wild-cards should not be created from nodes that are ancestors to preserved nodes. This condition is enforced using the following function:

$$checkPreserved(N_{combine}, N_1, N_2) = \begin{cases} N_{combine} & \text{if } type(N_{combine}) \neq \omega \\ N_{combine} & \text{if } type(N_{combine}) = \omega \text{ and number of} \\ & \text{preserved descendants of } N_1 \text{ and } N_2 = 0 \\ FAIL & \text{if } N_{combine} = FAIL \text{ or } type(N_{combine}) = \omega \text{ and} \\ & \text{number of preserved descendants } (N_1 \text{ or } N_2) > 0 \end{cases}$$

Both the *combineNodes* and *checkPreserved* functions are encapsulated within a single function:

$$mergeNodes(N_1, N_2) = checkPreserved(combineNodes(N_1, N_2), N_1, N_2)$$

The *mergeNodes* function returns a single node representing both input nodes (or *FAIL* if the nodes cannot merge). This node does not have any allocated child nodes. These are assigned in the next stage of the *merge* function.

4.4.3.2 Merge of sequences of children

Each node in a pair of input rules has zero or more child nodes. Each child node represents the root of its own sub-tree, and the *merge* function is applied recursively to every pair of corresponding child nodes. Cases exist in which a number of sub-sequences of children from one node can be merged with a number of sub-sequences of children from another node, as illustrated in Figure 4.18.

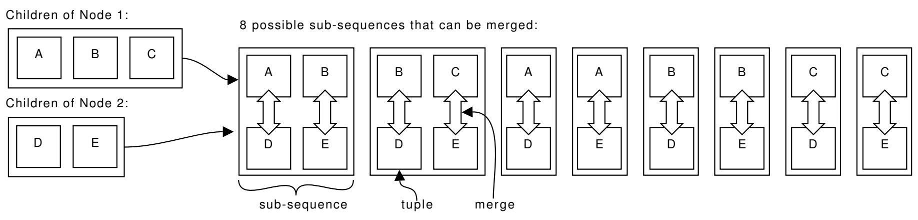


Figure 4.18: Illustration of the enumeration of node sub-sequences.

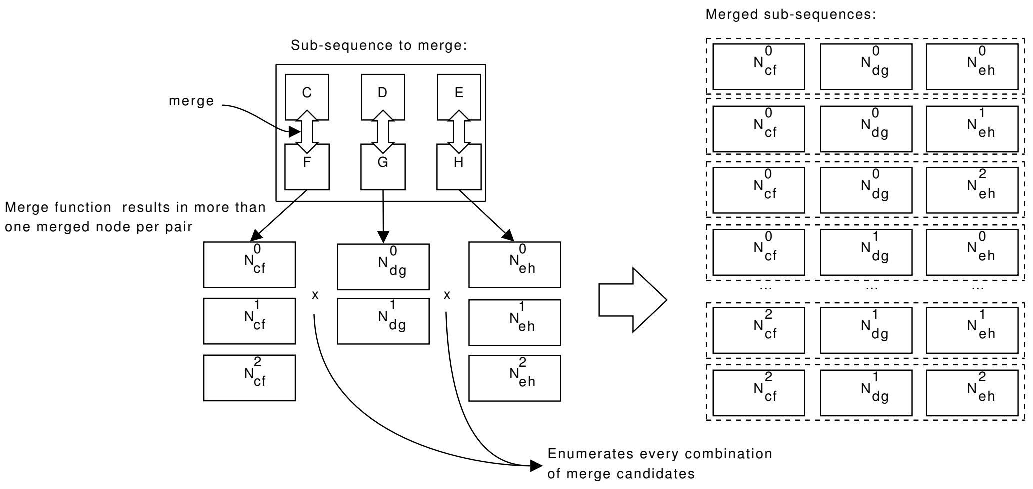


Figure 4.19: Illustration of merge between sequences of nodes.

We define a function $enumerate(\underrightarrow{C}, \underrightarrow{D}) \mapsto \{\underrightarrow{E_1}, ..., \underrightarrow{E_n}\}$ that takes as input two sequences of nodes $\underrightarrow{C}$ and $\underrightarrow{D}$, and produces all possible sub-sequences of corresponding nodes. Each sub-sequence $\underrightarrow{E_i}$ is a list of tuples, where each tuple defines a pair of nodes to be merged. If $m = size(\underrightarrow{C})$ and $n = size(\underrightarrow{D})$, then the total number of possible sub-sequences generated by *enumerate* is:

$$\sum_{k=0}^{max(n,m)} (n-k)(m-k) \tag{4.1}$$

The *merge* function is invoked recursively for each pair of nodes in each sub-sequence. If a pair does not produce a valid generalized node, then the entire sub-sequence is discarded.

The *merge* function returns a set of merge candidates for a single pair of input nodes (Definition 4.10), as is illustrated in Figure 4.19. Every combination of merge candidates from each pair in a sequence is enumerated using the $\times$ operator as illustrated in the figure. This process is encapsulated in the $mergeSub(\underrightarrow{E}) \mapsto \{\underrightarrow{S_1}, ..., \underrightarrow{S_n}\}$ function that takes as input a sequence of tuples $\underrightarrow{E} = \langle\langle C_1, D_1\rangle, ..., \langle C_k, D_k\rangle\rangle$, where each tuple represents a pair of nodes to be merged. The output of this function is a set containing every possible combination of merged sequences:

$$\begin{aligned} mergeSub(\underrightarrow{E}) &= merge(C_1, D_1) \times ... \times merge(C_k, D_k) \\ &= \{N^0_{c_1 d_1}, ..., N^x_{c_1 d_1}\} \times ... \times \{N^0_{c_k d_k}, ..., N^z_{c_k d_k}\} \\ &= \{\langle N^0_{c_1 d_1}, ..., N^0_{c_k d_k}\rangle, ..., \langle N^x_{c_1 d_1}, ..., N^z_{c_k d_k}\rangle\} \end{aligned}$$

The *mergeSub* function is invoked for every enumerated sub-sequence of child nodes. We use the *mergeChildren* function to merge every possible sub-sequence of corresponding child nodes. This function takes as input two sequences of child nodes, enumerates every possible sub-sequence, merges each sub-sequence, and finally enumerates all possible merged sub-sequences. If $\{\underrightarrow{E_1}, ..., \underrightarrow{E_k}\}$ is the set of enumerated sub-sequence pairs resulting from $enumerate(\underrightarrow{C}, \underrightarrow{D})$, then:

$$\begin{aligned} mergeChildren(\underrightarrow{C}, \underrightarrow{D}) &= mergeSub(\underrightarrow{E_1}) \cup ... \cup mergeSub(\underrightarrow{E_k}) \\ &= \{\underrightarrow{M_1}, ..., \underrightarrow{M_n}\} \end{aligned}$$

The *mergeChildren* function returns candidate merged sub-sequences of the child nodes of the two input nodes. A valid sub-sequence contains every preserved child node of the two input nodes. This is checked as follows:

$$checkChildPreserved(\underrightarrow{M}, \underrightarrow{C}, \underrightarrow{D}) = \begin{cases} true & \text{if number of preserved nodes in } \underrightarrow{C} \text{ and } \underrightarrow{D} = \\ & \quad \text{number of preserved nodes in } \underrightarrow{M} \\ false & \text{if number of preserved nodes in } \underrightarrow{C} \text{ and } \underrightarrow{D} \neq \\ & \quad \text{number of preserved nodes in } \underrightarrow{M} \end{cases}$$

Let N_{merge} be a merged node produced from nodes N_1 and N_2 using $mergeNodes(N_1, N_2)$. If more than one sub-sequence of child nodes is produced using *mergeChildren*, then N_{merge} is cloned for each valid sub-sequence, and the nodes in one sub-sequence become children of one clone of N_{merge}. The set of cloned N_{merge} nodes is returned as the result of the *merge* function.

Algorithm 4.2 defines the *merge* function, where the *mergeNodes* function is used to create new generalized nodes, and the *mergeChildren* function is used to recursively merge sub-sequences of child nodes. This algorithm has two base-cases: if a generalized node is a wild-card, in which case no further recursion occurs; or if no valid merged sub-sequences of child nodes exist (either if nodes are leaf nodes, or if no valid merge is possible). In the latter case, the new node N_{merge} is not assigned any child nodes.

The *childSubSeq* marker is set to true in N_{merge} whenever a merged sub-sequence of child nodes is smaller than the original set of child nodes. This indicates that the set of child nodes assigned to N_{merge} is a sub-sequence of possible child nodes. A special case exists in which no child nodes are assigned to a clone of N_{merge}. In this case, *childSubSeq* is only set to true if the original nodes have children.

We prove that the first property of generalized rules (generalized rule contains all preserved and answer nodes, as per Criterion 4.9 on page 66) is satisfied using the *merge* function defined by Algorithm 4.2:

Lemma 4.12. *Let N_1 and N_2 be the root nodes of two trees that have preserved and answer nodes in corresponding locations. Any generalized node N_{merge} produced by Algorithm 4.2 contains all preserved and answer nodes in N_1 and N_2, in the same location as in N_1 and N_2.*

Proof. All answer nodes are preserved nodes, therefore we only need to prove that all preserved nodes in the two input trees occur at the same location in a generalized tree. Define N_i to be the

Algorithm 4.2 Merge algorithm.

```
merge(in: node $N_1$ with children $\underset{\rightarrow}{C}$,
          node $N_2$ with children $\underset{\rightarrow}{D}$;
      out: set of merged nodes or $FAIL$)
begin
  $N_{merge} \leftarrow mergeNodes(N_1, N_2)$     %% $N_{merge}$ has no children yet
  $M \leftarrow \{\}$
  if $N_{merge} \neq FAIL$ then
    if $type(N_{merge}) = \omega$ then          %% Do not recurse if wild-card inserted
      $M \leftarrow M \cup \{N_{merge}\}$
    else
      $E \leftarrow mergeChildren(\underset{\rightarrow}{C}, \underset{\rightarrow}{D})$    %% Recurse
      for each $\underset{\rightarrow}{e} \in E$ do
        if $checkChildPreserved(\underset{\rightarrow}{e}, \underset{\rightarrow}{C}, \underset{\rightarrow}{D})$ then
          $N_{clone} \leftarrow clone(N_{merge})$    %% Clone $N_{merge}$ for each sub-sequence
          $children(N_{clone}) \leftarrow \underset{\rightarrow}{e}$     %% Assign children to $N_{clone}$
          %% Indicate if sub-sequence is not complete set of child nodes
          $childSubSeq(N_{merge}) \leftarrow (size(\underset{\rightarrow}{e}) \neq size(\underset{\rightarrow}{C})$ or $size(\underset{\rightarrow}{e}) \neq size(\underset{\rightarrow}{D}))$ or
                                    $(childSubSeq(N_1)$ or $childSubSeq(N_2))$
          $M \leftarrow M \cup \{N_{clone}\}$
      %% Special case: leaf nodes or no valid merges
      if $checkChildPreserved(\langle\rangle, \underset{\rightarrow}{C}, \underset{\rightarrow}{D})$ then
        %% Indicate if sub-sequence is not complete set of child nodes
        $childSubSeq(N_{merge}) \leftarrow (size(\underset{\rightarrow}{C}) \neq 0$ or $size(\underset{\rightarrow}{D}) \neq 0)$
        $M = M \cup \{N_{merge}\}$
  if $M \neq \{\}$ then
    return $M$
  else
    return $FAIL$
end
```

root node of a tree of height i, and N_j to be a root node of a tree of height j. We employ an inductive proof in four cases:

Case 1: $i = j = 0$ and N_i and N_j are both preserved. The *mergeNodes* function is guaranteed to return a merged node N_{merge} that is also preserved as specified by the definition of the *combineNodes* function (because both N_i and N_j are preserved). $FAIL$ can never result from *checkNodes* because neither N_i nor N_j have children. If $type(N_{merge}) = \omega$ in Algorithm 4.2 then N_{merge} is the only possible result. Otherwise, the recursive *mergeChildren* function is invoked over the children of N_i and N_j, returning an empty sequence because neither of these nodes have children. *checkChildPreserved* is guaranteed to return true over the empty sequence (because there are no children), and the result is the return of the preserved node N_{merge}. The preserved node in the generalized tree is in the same location as in the original trees.

Case 2: $i = j = 1$ and N_i and N_j have preserved children in identical locations.
If $type(N_{merge}) = \omega$ then *checkPreserved* is guaranteed to return a $FAIL$, preventing the possibility of a generalized tree with fewer preserved nodes than in N_i and N_j. Three cases are possible when merging the sequences of child nodes:
- *mergeChildren* returns an empty sequence: the special case is encountered, but *checkChildPreserved* prevents the addition of N_{merge} to M because the number of preserved child nodes of the input nodes is greater than zero. The result is an empty set M, which causes the algorithm to result in a $FAIL$.
- *mergeChildren* returns a set of sub-sequences, none of which pass the *checkChildPreserved* condition: this has the same result as the previous point.
- *mergeChildren* returns a set of sub-sequences, some of which pass the *checkChildPreserved* function: *checkChildPreserved* guarantees that the number of preserved nodes in the merged sub-sequence is the same as the number of preserved child nodes of N_i and N_j. If a sub-sequence is of the same size as the number of children of N_i and N_j (assuming this number is equal), then the location of the preserved nodes is identical because the sub-sequence is an exact copy of the child sequences of N_i and N_j. If the sub-sequence is of a reduced size then it is still guaranteed to contain all preserved nodes because of the *checkChildPreserved* function. In this case, $childSubSeq(N_{merge}) = true$, and by the definition of location, a value exists for k so that the preserved nodes in the generalized sub-sequence corresponds with the preserved child nodes of N_i and N_j.

Case 3: $i = 0$ and $j > 0$ where N_i is preserved and N_j is root to a sub-tree containing preserved nodes: in this case, N_j cannot be a parent to a sub-tree containing preserved nodes (by definition, N_i and N_j must have preserved nodes in identical locations). Therefore, N_j is the only preserved node in its tree. In this case, the *mergeNodes* function is guaranteed to return a preserved merged node N_{merge}, and because N_j cannot have preserved children (and N_i has no children) the insertion of a wild-card never removes preserved nodes.

Case 4: $i = x$ and $j = y$ and N_i and N_j both have preserved descendants. If N_i and N_j is preserved, then the *mergeNodes* function is guaranteed to return a preserved merged node N_{merge}, and a $FAIL$ is guaranteed by *checkNodes* if a wild-card is inserted. Recursive invocation of *merge* eventually results in either Case 1, 2, or 3, each of which guarantee preserved nodes in the same location in the sub-tree, or $FAIL$. Every generalized sub-tree is guaranteed to have preserved nodes in the same location as in the input trees.

□

We evaluate complexity of the *merge* algorithm in terms of the number of candidate merged rules that are produced from a pair of input rules. The worst case occurs if every node in both rules contains n child nodes (branching factor of n). The *enumerate* function produces no greater than $n \times n^2$ child sub-sequences (as a result of expression 4.1 on page 68). Assuming there are siblings to these parent nodes, then the upper bound on the total number of candidate merged sub-trees is $n \times n^2 \times n^2$ (in the worst case, each parent node corresponds to every other parent node, resulting in an additional factor of n^2). If this occurs for every node in the tree, and if there are p nodes in a tree, then the total number of merge candidates cannot exceed $O(pn^5)$ rules.

The upper bound of merge candidates is never reached due to the insertion of wild-cards that prevent traversal and duplication of the entire input trees. In addition, merge candidates are disqualified because of failure points in the *merge* algorithm. Empirical evaluation indicates that an average branching factor of 2 occurs, where input rules contain on average 30 nodes. This presents an upper bound of 2×30^5 candidate trees, but in practice we observe an average of only 2 candidates created for each pair of rules.

4.4.4 Application of a generalized rule: matching

The matching process compares a given rule with an unmarked tree. If the pattern specified by the rule exists in the unmarked tree, then the match is successful. Similar to generalization, matching is performed as a depth-first traversal of the rule tree and the unmarked tree, starting at the root.

We refer to a node from the rule as N_R, while a node from the unmarked tree is referred to as N_U. As per Definition 4.7 on page 65, an unmarked tree contains neither wild-cards, *childSubSeq* markers, nor answer nodes. The *match* algorithm is only concerned with checking that corresponding nodes are identical.

Algorithm 4.3 presents the recursive rule-matching process beginning at the root nodes of two trees N_R and N_U. If the node under consideration is an answer node, then the corresponding node N_U is marked as an answer. Subsequent checks are performed to ensure that the constraint portions of the rules are identical. If the node is a wild-card, then the corresponding node N_U can be of any type, and its sub-tree requires no further matching of N_U's children. The result is a positive match for the current sub-tree.

The type fields of the corresponding nodes are compared, and if these are different then the match fails. If they are the same, then the child nodes are matched recursively.

The matching of child nodes depends on the status of the *childSubSeq* marker. If set to *true*, this marker indicates that the sequence of child nodes of N_R is not complete, and may be surrounded on either side by nodes. In this case, a match occurs only if a sub-sequence of child nodes of N_U can be found, for which each child node results in a positive match with each corresponding child node of N_R. If no matching sub-sequence can be found, then the match fails. If *childSubSeq* is set to $false$, the the entire set of children of N_R must match the entire set of children of N_U. If

Algorithm 4.3 Rule matching.

```
match (in: node $N_U$ with children $\underrightarrow{C_U}$,
           node $N_R$ with children $\underrightarrow{C_R}$;
       out: a Boolean value $match$ indicating a successful match)
begin
 if $isAnswer(N_R)$ then
   $isAnswer(N_U) \leftarrow true$
 if $type(N_R) = \omega$ then
   return $true$
 else
   if $type(N_U) = type(N_R)$ then
     if $\underrightarrow{C_U} = \langle\rangle$ and $\underrightarrow{C_R} = \langle\rangle$ then
       return $true$
     else
       if $childSubSeq(N_R) = true$ then
         if $size(\underrightarrow{C_R}) = 0$ then
           return $true$
         else
           if (a sub-sequence in $\underrightarrow{C_U}$ exists that contains the same
               number of nodes as in $\underrightarrow{C_R}$, and where every node in the
               sub-sequence returns positive $match$ with every
               corresponding node in $\underrightarrow{C_R}$) then
             return $true$
           else
             return $false$
       else
         if (every node in $\underrightarrow{C_U}$ returns a positive $match$ with every
             corresponding node in $\underrightarrow{C_R}$) then
           return $true$
         else
           return $false$
   else
     return $false$
end
```

this is not the case (for example, if N_U has a different number of child nodes to N_R), then the match fails. Regardless of the status of *childSubSeq*, a recursive call to *match* is made for each corresponding pair of nodes.

The second property of a generalized rule is that it must match the two rules responsible for its creation (by Criterion 4.9 on page 66). This property is ensured by the following lemma:

Lemma 4.13. *Let R_1 be an example rule with root N_1, and R_2 be an example rule with root N_2. Let R_1 contain preserved nodes and answer nodes in the same locations as in R_2, and let N_{merge} be the root of a merged rule derived by applying $merge(N_1, N_2)$ from Algorithm 4.2. Then:*

$$\forall N_{merge} \in merge(N_1, N_2) : match(N_{merge}, N_1) = true \text{ and } match(N_{merge}, N_2) = true$$

Proof. To prove that any merge resulting from $merge(N_1, N_2)$ will match with N_1 and N_2, all failure points in the *match* algorithm (Algorithm 4.3) must be shown not to be reachable when matching $N_{merge} = N_R$ (the rule) with $N_1 = N_U$ or $N_2 = N_U$ (the unmarked tree). Algorithm 4.3 can result in $false$ in 3 cases:

Case 1: if $type(N_U) \neq type(N_R)$: By this stage in Algorithm 4.3, it is asserted that $type(N_R)$ is not a wild-card. Algorithm 4.2 employs the function *mergeNodes* that returns a node with type equal to either a wild-card ω if $type(N_1) \neq type(N_2)$ or the type of N_1(and N_2) if $type(N_1) = type(N_2)$. As such, it is impossible for N_R to have been created containing a node in which $type(N_U) \neq type(N_R)$, which means this case will never occur if $N_U = N_1$ or $N_U = N_2$. This case is therefore guaranteed to never be the reason for a match failure.

Case 2: if $type(N_U) = type(N_R)$ and $childSubSeq(N_R) = true$ and no matching sub-sequence exists: A match failure occurs only if no sub-sequence of child nodes from N_U can be found that match the sequence of child nodes of N_R. The existence of N_R with $size(C_R) > 0$ indicates that at least one merged sub-sequence of children was produced from the children of N_1 and N_2. The *match* function enumerates and matches every possible sub-sequence of child nodes of N_1 and N_2, guaranteeing that the sub-sequence found by *merge* is also found by *match*, making a match failure impossible.

Case 3: if $type(N_U) = type(N_R)$ and $childSubSeq(N_R) = false$: the property $childSubSeq(N_R) = false$ can only result from merge in two cases: if the $type(N_R) = \omega$ because *mergeNodes* sets this property to false by default and no subsequent changes are possible if a wild-card is inserted; or if the number of merged child nodes is equal to the number of child nodes of N_1 and N_2. The former case does not apply for this point. In the latter case, N_R is guaranteed to contain the full sequence of child nodes identical to the children of N_1 and N_2, and therefore a *match* failure is impossible.

□

Of the three properties that generalized rules should exhibit, only two are guaranteed by the *merge* and *match* functions. Lemma 4.12 guarantees that any generalized tree contains all preserved nodes in the same locations as those in the two input rules. Answer nodes are always preserved nodes, as per the definition of a rule (Definition 4.6), and therefore answers in a generalized rule are always in the same location as in the input rules. Lemma 4.13 guarantees that a

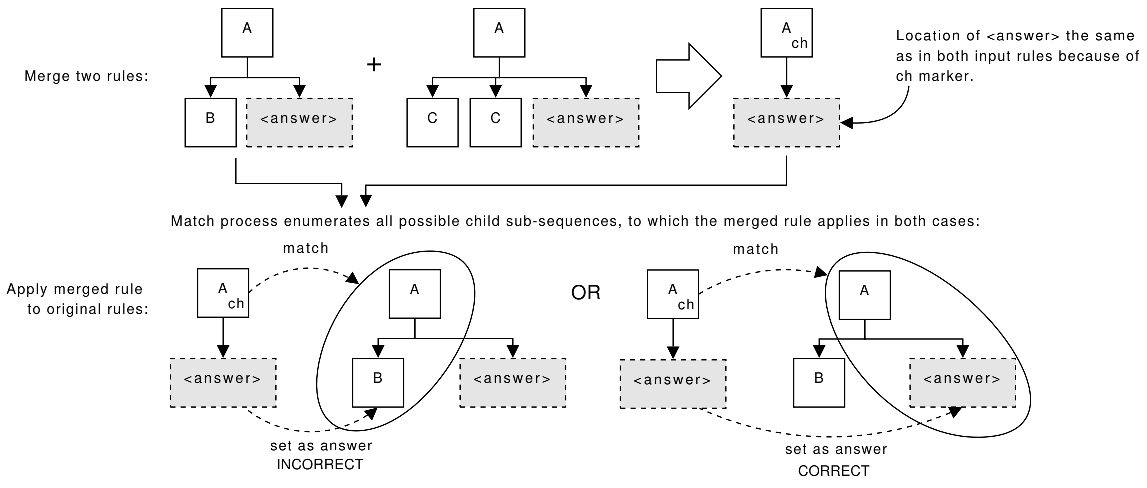

Merge two rules:
A
B
<answer>
+
A
C
C
<answer>
A ch
<answer>
Location of <answer> the same as in both input rules because of ch marker.
Match process enumerates all possible child sub-sequences, to which the merged rule applies in both cases:
Apply merged rule to original rules:
match
A ch
<answer>
A
B
<answer>
set as answer INCORRECT
OR
match
A ch
<answer>
A
B
<answer>
set as answer CORRECT

Figure 4.20: Example of the failure to guarantee that all identical answers are produced by a merged rule.

merged rule generalized from two input rules will always match the two input rules. This ensures that the generalized rule is capable of replacing both input rules.

The third property of a generalized rule (Criterion 4.9 on page 66) cannot be guaranteed, namely that the answers produced by the application of the generalized rule are the same answers produced by both input rules. An example is illustrated in Figure 4.20, which shows a merged rule that generalizes two input rules. By the definition of location, the answer node is in the same location as in the original rules because of the *childSubSeq* marker. However, when this rule is matched to an original rule, the *childSubSeq* marker results in the enumeration of every sub-sequence of child nodes. The *match* algorithm has the potential to match two different enumeration possibilities and indicate a different answer in either case, one of which is incorrect. The match algorithm has no method for determining which enumeration possibility is correct, allowing the potential for incorrect annotations.

The error illustrated in Figure 4.20 is the result of the creation of a merged rule that is not constrained enough to distinguish between the two input rules. One option for correcting this is to prevent the use of the *childSubSeq* marker when a descendant is an answer. However, this over-constrains rules making them less applicable over unseen text. We avoid this problem by identifying and preventing only the special cases where conflicts occur, and by performing a post-merge *match* between every generalized candidate and the original two rules. If a match occurs that results in an incorrect answer, then the generalized candidate is discarded. This post-merge check is a special case concerned with the avoidance of *conflicting* rules, a scenario dealt with when generalizing rule-sets.

4.4.5 Generalization of a rule-set

We generalize a rule-set in a pair-wise fashion. We begin with en empty set of merged rules. A single rule is removed from the base rule-set and added to this set. Thereafter one rule is removed at a time and merged with every rule in the merged rule-set. This results in a set of generalized merge candidates, from which only a single candidate is chosen. This candidate replaces the rule in

Algorithm 4.4 Pairwise rule-set generalization.

```
generalize(in: $\underrightarrow{\mathbf{S}}$ the base rule-set
           out: $\underrightarrow{\mathbf{M}}$ the merged rule-set)
begin
  $\underrightarrow{\mathbf{M}} \leftarrow \langle\rangle$
  $\underrightarrow{\mathbf{O}} \leftarrow \langle\rangle$   %%Original rule-set
  while $size(\underrightarrow{\mathbf{S}}) > 0$ do
    $R \leftarrow removeFirst(\underrightarrow{\mathbf{S}})$
    $\underrightarrow{\mathbf{O}} \leftarrow$ append $R$ to $\underrightarrow{\mathbf{O}}$   %% Record original rule to prevent conflicts
    $\left(\underrightarrow{\mathbf{N}}, R_{merge}\right) \leftarrow selectMerge(R, \underrightarrow{\mathbf{M}}, \underrightarrow{\mathbf{S}}, \underrightarrow{\mathbf{O}})$
    if $\underrightarrow{\mathbf{N}} = FAIL$ then
      if (no rule exists in $\underrightarrow{\mathbf{M}}$ identical to $R$ and
            not $isConflict(R, \underrightarrow{\mathbf{M}}, \underrightarrow{\mathbf{S}}, \underrightarrow{\mathbf{O}})$) then
        $\underrightarrow{\mathbf{M}} \leftarrow$ append $R$ to $\underrightarrow{\mathbf{M}}$   %% append non-merged rule to merged set
      else
        discard rule $R$  %% $R$ causes unresolvable conflict
    else
      $\underrightarrow{\mathbf{M}} \leftarrow \underrightarrow{\mathbf{N}}$  %% Replace merged set with updated version
      if (no rule exists in $\underrightarrow{\mathbf{S}}$ identical to $R_{merge}$) then
        append $R_{merge}$ to $\underrightarrow{\mathbf{S}}$ %% Allow future generalization of $R_{merge}$
end
```

the merged rule-set that was used for the creation of the candidate. If no valid generalized merge candidates are produced, then the rule from the base rule-set is appended to the merged rule-set. This process continues until no further rules exist in the base rule-set. Every time a generalized merge candidate is added to the merged rule-set, a copy is also appended to the base rule-set to allow for future generalization of the same rule.

Algorithm 4.4 performs the pair-wise rule-set generalization process. A function called *selectMerge* is used to merge the rule R from the base rule-set with every rule in the merged rule-set, returning a tuple containing the new merged rule-set and the merged rule that was added to it. Before any merged or unmerged rule is added to the merged rule-set, it is verified to ensure that it does not cause a conflict with any existing rule in the merged rule-set (thus ensuring the third property of generalized rules).

Algorithm 4.4 has a time complexity of $O(n^2)$, where n is the number of rules, because of the merge between every rule in the base rule-set with every rule in the merged rule-set (which in the worst case contains no merged rules). However, in our experience this scenario never occurs because merged rules are always produced.

4.4.5.1 Detection of conflicting rules

Conflicting rules are capable of incorrectly matching examples and produce an incorrect answers. They occur under the following circumstances:

1. Rules in the base rule-set are conflicting;

2. Generalized rules conflict with rules in the merged rule-set;

3. Generalized rules conflict with the rules responsible for their creation.

If rules in the base rule-set are conflicting, then the most that can be done is detect when this occurs, and raise a warning. The structure of the rules should be enhanced to contain more detailed constraints to avoid conflicting rules, or alternatively the example data should be checked for inconsistencies.

The second circumstance is avoided by matching every newly generalized rule with every rule in the merged rule-set, to ensure that no match occurs that results in an incorrect answer.

The third circumstance is illustrated in Section 4.4.4, where a generalized rule conflicts with both input rules. This is avoided by matching the generalized rule with both input rules, and discarding the generalized rule if conflicting answers are produced. However, this is a special case of a larger problem. A newly generalized rule might not conflict with a rule in the merged rule-set, or with the two input rules, but it might conflict with one of the rules in the original base rule-set. To avoid this, every generalized rule is matched with every rule in the base rule-set (which includes the two input rules), and is discarded if any conflicting answers are produced.

We define the function $match_P(R_1, R_2) \mapsto (Boolean, Boolean)$ that accepts two rules and returns a tuple of two Boolean values $(match, same)$. The $match$ variable is assigned the value of $true$ if the two rules successfully match. The $same$ variable is assigned the value of $true$ if both the following conditions hold:

- R_1 identifies the correct answer nodes in the R_2 tree; and
- R_2 identifies the correct answer nodes in the R_1tree.

The $match_P$ function differs from the $match$ function in that both input trees can be rules, and so both contain wild-cards, *childSubSeq* markers, preserved nodes, and answer nodes. Answer nodes are considered as wild-cards (as described in Section 4.3.5).

We define *conflicting rules* to be rules for which the $match_P$ algorithm produces the tuple $(match = true, same = false)$. We use the function $conflictExists(R, \underrightarrow{\mathbf{R}}) \mapsto Boolean$ to detect whether a rule R is conflicting with respect to a rule-set $\underrightarrow{\mathbf{R}}$ (if any rule M exists in $\underrightarrow{\mathbf{R}}$ such that $match_P(R, M) = (true, false)$).

Every time a new rule is added to the merged rule-set, all circumstances that result in a conflict are checked. This means checking the merged rule-set $\underrightarrow{\mathbf{M}}$, the base rule-set $\underrightarrow{\mathbf{S}}$, and the original rule-set $\underrightarrow{\mathbf{O}}$ for conflicts. All these checks are encapsulated in the $isConflict(R, \underrightarrow{\mathbf{M}}, \underrightarrow{\mathbf{S}}, \underrightarrow{\mathbf{O}})$ function that returns $true$ if the following condition holds:

$$conflictExists(R, \underrightarrow{\mathbf{M}}) = true \text{ or } conflictExists(R, \underrightarrow{\mathbf{S}}) = true \text{ or } conflictExists(R, \underrightarrow{\mathbf{O}}) = true$$

The $isConflict$ function is used in Algorithm 4.4 in cases where no valid merge candidates are produced. The rule R is appended to the merged rule-set only if no conflict is detected.

4.4.5.2 Selection of a generalized rule

Algorithm 4.4 chooses a single rule from $\underrightarrow{\mathbf{S}}$ that is merged with every rule in $\underrightarrow{\mathbf{M}}$ using the $selectMerge$ function, which is defined in Algorithm 4.5. A merge results in a set of merge candidates, each of which is checked to determine whether it is a conflicting rule. Once all valid merge candidates are determined, a single candidate is chosen. We previously explored evaluating

Algorithm 4.5 Select merged rule-set.

```
selectMerge(in: rule $R_C$ to be merged with any rule in $\underrightarrow{\mathbf{M}}$,
                merged rule-set $\underrightarrow{\mathbf{M}}$,
                rule-set $\underrightarrow{\mathbf{S}}$ containing rules still to be merged,
                rule-set $\underrightarrow{\mathbf{O}}$ containing original rules;
            out: tuple $\left(\underrightarrow{\mathbf{N}}, R_{merge}\right)$ | $FAIL$)
begin
  $W \leftarrow \{\}$
  for each $R$ in $\underrightarrow{\mathbf{M}}$ do
    if $R$ is not identical to $R_C$ then
      $V \leftarrow merge(R, R_C)$
      if $size(V) > 0$ then
        for each $R_m$ in $V$ do
          $\underrightarrow{\mathbf{N}} \leftarrow \underrightarrow{\mathbf{M}}/R$     %% Remove $R$ from $\underrightarrow{\mathbf{M}}$
          if not $isConflict(R_m, \underrightarrow{\mathbf{M}}, \underrightarrow{\mathbf{S}}, \underrightarrow{\mathbf{O}})$ then
            append $R_m$ to $\underrightarrow{\mathbf{N}}$
            $W \leftarrow W \cup \{\left(\underrightarrow{\mathbf{N}}, R_m\right)\}$
  if $W = \langle\rangle$ then
     return $FAIL$
  else
     choose one tuple from $W$ and return it
end
```

each merged rule over the example set, choosing the rule that creates the most correct annotations (Glass and Bangay, 2006). However, this method requires repeated rule-set application and evaluation. Alternatively, we choose the candidate that matches most rules in the merged rule-set without conflict (to promote future generalization), removing the application and evaluation steps.

4.4.6 Rule-set application to unseen text

An unmarked tree T is created for each example. The rule-set $\underrightarrow{\mathbf{R}}$ is then traversed in order, and a *match* is performed between each rule R from this rule-set and T. If a positive match occurs, then the answer node created in T by the *match* function is used to create the annotation. This process is described in Algorithm 4.6.

Assume that the rule-set contains n rules and the example set consists of n examples. Algorithm 4.6 has a run-time complexity of $O(n^2)$ because the last rule in each rule-set has the potential to match every rule in the example set. We observe that this worst-case scenario is never realized. In general, the *apply* algorithm is far more efficient than the *generalize* algorithm because every possible merge candidate need not be enumerated, and the *match* algorithm ends as soon as a matching candidate is located. As with most machine learning algorithms, the activity of training requires substantial effort (in the order of hours), but the application process requires less effort or resources to execute (in the order of seconds).

Algorithm 4.6 Rule-set application.

```
apply(inout: extract $E$ containing a number of examples;
      in: rule-set $\underset{\rightarrow}{\mathbf{R}} = \langle R_1, ..., R_n \rangle$)
begin
  while $E$ has more examples do
    $e \leftarrow$next example in $E$
    construct unmarked tree $T$ from the example
    for $i \leftarrow 1$ to $n$ do
      if $match(T, R_i) = true$ then
        annotate $e$ according to answer nodes in $T$
        break from loop over $i$
end
```

4.5 Hierarchical rules for semantic annotations

This section describes the construction of rules for different categories of semantic annotation. We choose categories that we believe are useful for creating animated 3D virtual environments, and which also demonstrate the capabilities of hierarchical rule-based learning:

1. **Quote:** identifies the avatar responsible for voicing instances of direct speech. A quote annotation is defined in terms of the following qualifiers:

   (a) *Quote* (trigger): the actual quote consisting of a sequence of tokens.

   (b) *Speech-verb* (text-reference): a token in the text that describes the act of speaking. A quote need not have a speech-verb.

   (c) *Actor* (text-reference): a token in the text that refers to the avatar performing the act of speaking, and is either a direct reference (for example, “Julian”, “Anne”) or an anaphoric reference (“he”, “she”). A quote need not have an actor.

   (d) *Speaker* (semantic-concept): We assume that a list of avatars occurring in the book exists (using a method such as described in Section on page 41 in Chapter on page 26). This field contains the identifier of the avatar responsible for the speech. Every quote annotation must identify a speaker.

2. **Setting:** identifies tokens that indicate physical location, for example “hill”, “valley”, “bedroom”, or “town”. This category is defined in terms of the following qualifier:

   (a) *Setting* (trigger): a token that describes the physical location of the current scene.

3. **Object:** identifies tangible entities, for example furniture, flora, and fauna. This category is defined in terms of the following qualifier:

   (a) *Object* (trigger): a token that describes a tangible entity in the scene.

4. **Transition:** identifies behaviour of an entity in terms of entry or exit from the scene, and is defined in terms of the following qualifiers:

   (a) *Transition* (trigger): a token that indicates an entry or exit.

They had it on the top of a hill, in a sloping field that looked down into a sunny <**setting**>**valley**</**setting**>. <**avatar**>**Anne**</**avatar**> didn't very much like a big brown <**object**>**cow**</**object**> who <**transition type="INSIDE" subject="cow"**>**came**</**transition**> up <**relation type="NEAR" subject="cow" object="her"**>**close**<**relation**> and stared at her, but it <**transition type="OUTSIDE" subject="it"**>**went**</**transition**> away when <**avatar**>**Daddy**</**avatar**> told it to.

Figure 4.21: Example annotated fiction text, from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

   (b) *Subject* (text-reference): a token referring to the entity performing the transition. This token is either a direct reference or an anaphoric reference. Each transition is required to have an associated *subject* to be valid.

   (c) *Type* (semantic-concept): describes the type of transition occurring. We define two alternatives for transition annotations, namely *inside* and *outside*. The former indicates that the entity is moving into the scene, while the latter indicates that the entity is leaving the scene. Only one of these two is chosen for each transition annotation.

5. **Relation:** identifies behaviour in terms of spatial relationships between two entities in the scene, and is defined in terms of the following qualifiers:

   (a) *Relation* (trigger): a token that indicates a spatial relation (for example "on", "under", or "behind").

   (b) *Subject* (text-reference): a token referring to the entity to which the relation applies. This token is either a direct reference or an anaphoric reference. Each relation is required to have an associated *subject* to be valid.

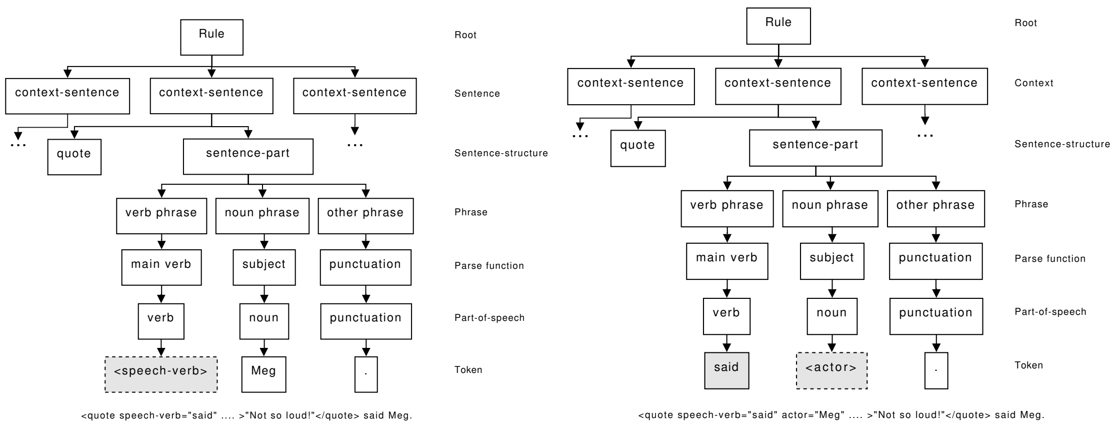
   (c) *Object* (text-reference): a token referring to the entity that serves as a reference point for the relation. This token is either a direct reference or an anaphoric reference. Each relation is required to have an associated *object* to be valid.

   (d) *Type* (semantic-concept): describes the type of spatial relation being described. We define the following semantic relations: *near*, *inFrontOf*, *behind*, *toLeftOf*, *toRightOf*, *onTopOf*, and *below*. Only one of these is chosen for each relation annotation.

An example of fiction text annotated using these categories is presented in Figure 4.21. The following section describes the levels of abstraction we include in rules for every category of annotation.

4.5.1 Rule structure for different annotation categories

We design rules so that they contain as many levels of abstraction as possible to support generalization. The general principle we adopt for rule structure is as follows: leaf nodes in the trees represent individual tokens from the input text. Multiple levels of abstraction are provided between the leaf nodes and the root node. For example, tokens are abstracted using parts-of-speech (resulting in a parent node for each token node). These nodes are grouped into phrases and sentences, all of which form descendants of the root node.



(a) Speech-Verb (b) Actor

(Preserved nodes are highlighted in gray; answer nodes are indicated with dashed borders)

Figure 4.22: Example rules for extracting *speech-verbs* and *actors*.

We always mark the answer at a leaf node in a rule because answers are generally tokens. Additional preserved nodes are included when there is a relation between annotation tasks. For example, rules that locate the *actor* of a quote mark nodes that indicate the *speech-verb* as preserved, because we believe that the *speech-verb* is fundamental in identifying the *actor*.

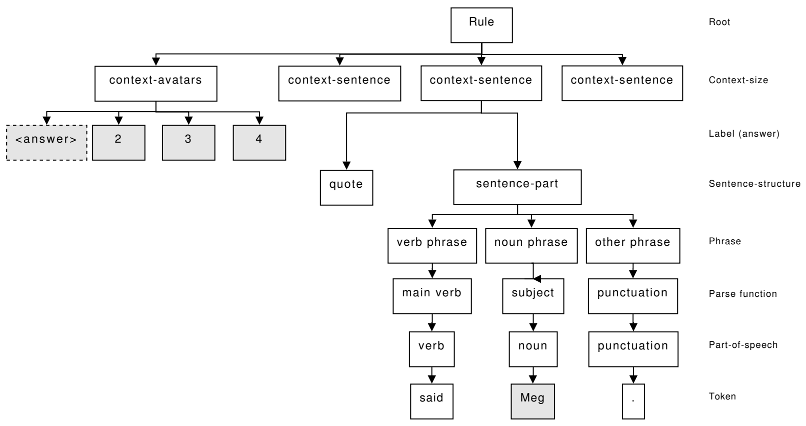
Rule-structures can also contain non-textual data to cater for the semantic concepts. We devise the method of including a sub-tree below the root node that contains a node for every available semantic concept. Each of these is preserved, but only one is marked as the answer. This ensures that no semantic concepts are removed during generalization.

The advantage of hierarchical rule-based learning is that rule-structures can be customized for each category of annotation, without modifying the core induction processes. The structures we use are described in the following sections.

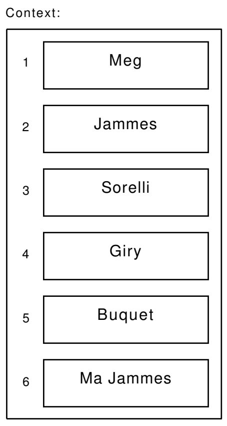
4.5.1.1 Quotes

Quotes are identified in fiction text with a high level of accuracy (described in Chapter 3). As such, we consider only the tasks of locating the *speaker* for each quote. Glass and Bangay (2007c) show in previous research that this task can be achieved by first locating the *speech-verb* of a quote. A link is then identified between this token and another in the text that identifies the *actor* of the verb. This token is then used to select a *speaker* using a set of hand-coded rules. The rule induction process removes the need for the intermediate steps of locating *speech-verbs* and *actors*. However, we include these annotation qualifiers as toy examples for demonstrating the abilities of hierarchical rule-based learning.

A rule for identifying the *speech-verb* of a quote is constructed for each quote trigger. Tokens from one sentence prior to the quote, the adjoining sentence of the quote (if there is one), and from one sentence after the quote form leaf nodes of the rule. The node containing the *speech-verb* is marked as the answer. Every token is abstracted using parts-of-speech and syntactic function. Tokens are then grouped into phrases, and then into sentences. A rule in the *speech-verb* category is illustrated in Figure 4.22(a).



(a) Speaker



(b) Context-model

(Preserved nodes are highlighted in gray; Answer nodes are indicated with dashed borders)

Figure 4.23: Example *speaker* rule and avatar context model.

A rule for the *actor* of a quote is identical in structure, as illustrated in Figure 4.22(b), with one addition: the token indicating the *speech-verb* is marked as preserved (to maintain the link between the actor and the quote).

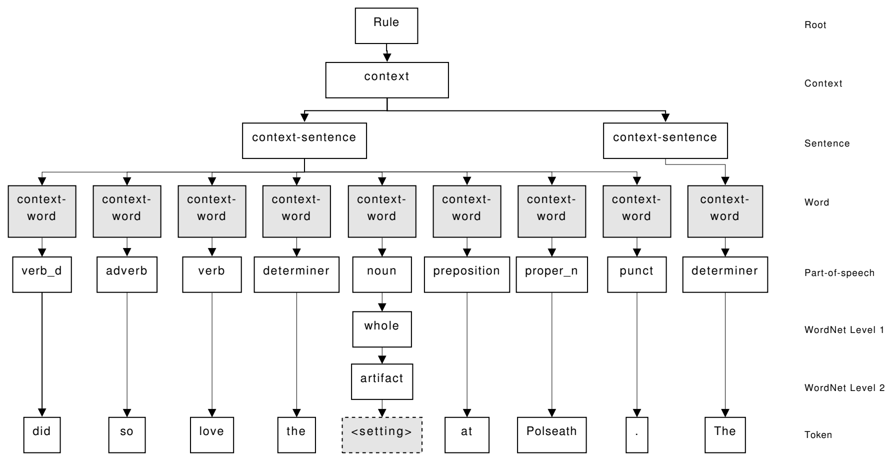
The formulation of the *speaker* rule demonstrates the ability to include non-text data in a rule. We make use of a custom built *context-model* to provide a list of avatars that occur in a current scene. An avatar identifier is placed at the front of this list every time an explicit reference is made to an avatar. The *speaker* qualifier identifies which avatar to choose from the list for a specific quote. This makes possible the identification of speakers involved in two-way dialogue where the speakers are not indicated explicitly in the text.

A rule for identifying the speaker is constructed in the same manner as rules for *speech-verbs* and *actors* (but actor nodes are marked as preserved). These rules include an additional sub-tree containing nodes that serve as an index to the context model. An example rule and context model is illustrated in Figure 4.23. Each index node is marked as preserved to prevent its removal, and the node indicating the correct index is marked as the answer.

The context model potentially introduces error to the rule-creation process. Assume that a rule is created for a quote that has avatar M as a speaker, but the context-model for this quote does not contain the identifier for M. A possible reason for this is an indirect reference to M that cannot be resolved correctly by the context model. However, a human resolved this reference during the creation of the manual annotation. In this case, no answer node can be highlighted during the creation of the rule because no M exists in the context model. Error introduced in this manner reduces the lower bound of correct annotations created by the base rule-set and subsequently generalized rule-sets (by the Consistency Assumption on page 62).

4.5.1.2 Settings and Objects

Setting and Object annotations are similar in that they only require the identification of a single token as a trigger. We divide an annotated extract into tokens, and create a rule for each token. Explicitly annotated tokens result in positive rules, and the rest result in negative rules.



(Preserved nodes are highlighted in gray; Answer nodes are indicated with dashed borders)

Figure 4.24: Example Setting rule derived from an example from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

An example rule for identifying Settings is illustrated in Figure 4.24. We believe that tokens before and after a positive example are important in discriminating whether a token belongs in a category. Empirical tests indicate that a window consisting of four tokens before and after the trigger is sufficient to ensure a lower bound of 100% correct matches as per the Consistency Assumption 4.2 on page 62.

We abstract the trigger token using hypernyms extracted from WordNet. All tokens are abstracted using parts-of-speech, and are grouped into sentences. Preliminary tests indicate that rule generalization removes surrounding tokens, reducing the discrimination ability of each rule in the set. We prevent this by inserting a preserved abstracted node ("context-word") as an ancestor to each token.

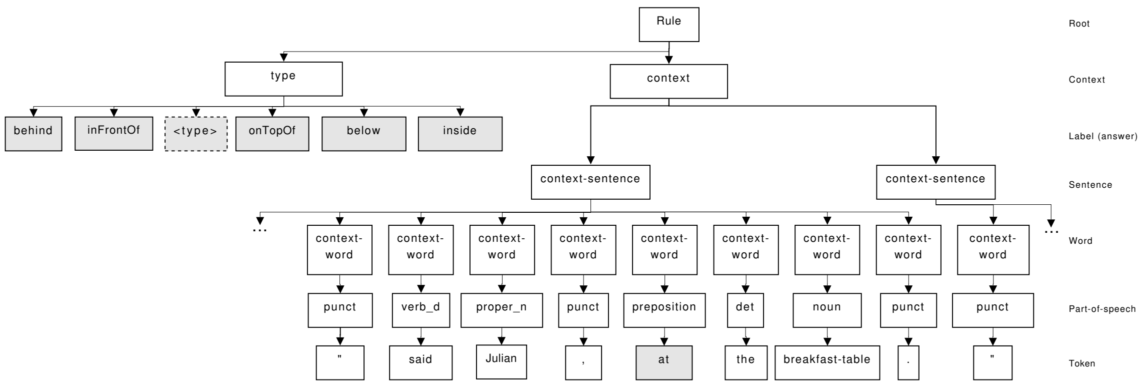
We use the same structure for rules created from Object annotations.

4.5.1.3 Transitions and Relations

Both Transition and Relation annotations identify triggers in their respective categories, a task that is identical to identification of Setting and Object triggers. We use the same rule-structure for locating triggers in these categories.

A similar rule structure for Setting triggers are used to identify the *subject* of a Transition or Relation annotation. The difference is that the trigger of the annotation is marked as preserved, indicating that there is a relationship between the *subject* token and the trigger of the annotation. Rules for the *object* of a Relation annotation have the same structure as those used to identify the *subject.*

Rules for identifying the *type* of a Relation are structured in the manner illustrated by Figure 4.25. To the right of the rule is a similar structure to Setting rules, but the trigger of the Relation is marked as preserved. The left child of the root node contains *type* nodes with the defined semantic concepts for the annotation category. All are marked as preserved, but only the node corresponding to the *type* of the example annotation is marked as the *answer.* Transition rules are similar, except they only have *inside* and *outside* as children to the type node.



(Preserved nodes are highlighted in gray; Answer nodes are indicated with dashed borders)

Figure 4.25: Example Relation rule for annotation *type* (derived from an example from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942)).

The order in which the *subject*, *object,* and *type* qualifiers are determined is not significant. A Transition or Relation annotation is only defined if these fields have data, and so only positive examples are provided for *subject, object,* or *type,* one for each valid trigger.

4.6 Analysis of hierarchical rule-based learning

We examine the properties of hierarchical rule-based learning to determine if it is effective in automatically creating semantic annotations over fiction text. These properties are investigated in terms of the following questions:

1. *Is there a small set of patterns in English that identifies a large portion of annotations in a particular category?*
   Hierarchical rule-based learning is based on the premise that patterns exist for identifying categories of semantic annotation. We investigate if this is the case, and whether hierarchical rule-based learning is capable of inducing these patterns.

2. *Does the type of book make a difference to the types of patterns learned?*
   Fiction books are written by different authors, in different genres and for different audiences. We investigate if the patterns induced from one book are applicable to different books, and whether examples from different books enhance a model induced by the learning system.

3. *How does the composition of the example set impact the ability of the induced rule-set in creating correct annotations?*
   Some annotation categories contain both negative and positive examples. We investigate the relationship between these different types with respect to the creation of accurate rule-sets.

4. *Can accurate rule-sets be induced for different categories of annotation using the same rule-structure?*
   One potential problem with hierarchical rule-based learning is the need for customized rule-structures for each annotation category. We investigate if the same rule-structure can be used to induce accurate models for different categories.

Book	Series	Author	Flesch	Fog	Ave. Words / Sentence
Book 1	A	A	95.6	4.7	9.3
Book 2	A	A	96.2	5.2	10.5
Book 3	B	B	87.8	6.6	11.7
Book 4	B	B	88.1	6.6	12.0
Book 5	C	C	92.2	6.3	13.0
Book 6	G	G	83.6	7.8	13.4
Book 7	C	C	90.5	6.9	14.6
Book 8	D	D	82.8	8.6	16.7
Book 9	D	D	80.0	9.1	17.6
Book 10	E	E	77.7	9.6	18.4
Book 11	E	E	76.7	10.0	19.2
Book 12	F	E	76.3	10.4	20.4
Book 13	F	E	64.8	13.9	28.3

Table 4.3: Breakdown of manually annotated fiction text corpus.

5. *Can generalized rule-sets be accurately induced for different annotation qualifiers?*
   Annotations are defined in terms of triggers, text references, and semantic concepts. We investigate if hierarchical rule-based learning is capable of automating the creation of each qualifier type for different categories of annotation, without modifying the core algorithm.

6. *Can hierarchical rule-based learning be used to automate the creation of accurate semantic annotations?*
   Assuming that patterns exist for identifying annotations, we investigate if induced patterns support automation in the following manner:

   (a) Can 50% (or less) of the total positive examples be used to identify more than 50% of the annotations in a category?

   (b) How can hierarchical rule-based learning be used to reduce the effort required to create the intermediate representation?

This section presents a suite of experiments and describes a corpus of data used to answer these questions. Metrics for measuring success are defined, and possible sources of experimental error are discussed.

4.6.1 Test corpus

We create a corpus of fiction that is manually annotated with the semantic categories identified in Section 4.5. Properties of the books used to construct this corpus are listed in Table 4.3, indicating which books are from the same series and by the same author. The books are ordered according to average sentence length and according to readability metrics, namely the Fog Index (Gunning, 1952) (where low scores describe "easy reading"), and the Flesch Index (Flesch, 1949) (where high scores describe "easy reading"). We observe that the readability indices correlate approximately with the average sentence length. Books by the same author, and in the same series generally cluster together in terms of these readability scales.

	Tokens	Quotes			Settings	Objects	Transitions	Relations
		speech-verbs	*actors*	*speakers*				
Book 1	49662	1109	1109	1229	399	425	108	68
Book 2	50011	1382	1382	1446	-	-	-	-
Book 3	113950	1169	1169	1666	-	-	-	-
Corpus	984036	9675	9674	13913	-	-	-	-

Table 4.4: Summary of annotation categories over corpus.

We consider books to be of different type if they have different authors, and are not clustered together in Table 4.3. In this respect, Book 1 is similar to Book 2, but is of a different type to Book 3.

All the books in Table 4.3 contain Quote annotations, but only Book 1 contains annotations in the other categories. The number of annotated examples in each category is listed in Table 4.4, which indicates Book 1, 2, and 3 because these are examined individually in the experiments.

4.6.2 Metrics

Experiments are conducted by providing an extract sourced from the corpus of annotated books to the rule-base learning system. A base rule-set is constructed from a sub-set of the examples in the extract. The base rule-set is applied to the entire extract to determine the accuracy of the least generalized rule-set over unseen data. When the number of rules in the base rule-set equals the total number of examples, the lower bound described in Consistency Assumption 4.2 on page 62 is established for the book. The base rule-set is then generalized using the algorithms presented in Section 4.4.5. The generalized rule-set is applied to the extract and the resulting annotations are compared with the original set.

The above process is repeated using base rule-sets of increasing size, until all examples are included in the base rule-set. The success of each merged rule-set provides an indication of the relationship between the number of example annotations required and the success of the generalized rule-set.

Some annotation qualifiers contain both positive and negative examples (Quote: *speech-verb, actor*; Setting: *trigger*; Object: *trigger*: Transition: *trigger*; Relation: *trigger*). In these cases, we measure the success with which positive annotations are created using:

- *False positives*: the number of positive annotations created where they should not occur. The ideal case is a value of zero for this metric.
- *False negatives*: the number of positive annotations not created where they should occur. The ideal case is a value of zero for this metric.
- *Correct annotations*: the sum of true positive and true negative annotations.
- *Recall*: the ratio of correct annotations to the total number of annotations that should exist:

$$recall = \frac{number\,of\,automatic,\,correct,\,positive\,annotations}{number\,of\,manual,\,positive\,annotations}$$

This metric determines how well a rule-set identifies positive annotations, but does not provide an indication of how well the rule-set avoids creating incorrect annotations. As such, it should always be viewed in conjunction with precision.

- *Precision*: the ratio of correct annotations to the total number of annotations created automatically:

  $$precision = \frac{number\,of\,automatic,\,correct,\,positive\,annotations}{number\,of\,automatic\,positive\,annotations}$$

  This metric provides an indication of the extent to which non-correct annotations are avoided. However, if the number of automatic positive annotations is small, then the precision ratio is high, and so this figure might be deceptive. As such, it should always be viewed in conjunction with recall.

Some annotation qualifiers are concerned only with positive examples because each annotation contains exactly one instance (Quote: *speaker*; Transition: *subject, type*; Relation: *subject, object, type*). In these cases, the number of false positives is irrelevant, and we are concerned with ensuring that the correct data is associated with the annotation. We measure success in terms of:

- *Accuracy*: the ratio of correct annotations to the total number of manual annotations:

  $$accuracy = \frac{number\,of\,automatic,\,correct,\,annotations}{number\,of\,manual\,annotations}$$

4.6.3 Sources of experimental error

We acknowledge the complexity of the English language and its understanding as a source of experimental error. The manual annotations produced for the corpus of test data are not guaranteed to consistently follow a single interpretation, and as a result cannot be evaluated as consistently correct. The possibility exists that identical examples are annotated differently, resulting in conflicting rules. An example of such a scenario is when the context-model is used for rule-creation, described in Section 4.5 on page 79. These cases are identified by evaluating the base rule-set over the example data, allowing a baseline to be established for annotation success (by Consistency Assumption 4.2 on page 62).

The variety of fiction authors, genres, and target audiences is another source of experimental error. Examples in a single book need not correlate with examples in other books, because of different writing styles. We propose a method for mitigating this source of error that involves training a rule-set with examples sourced from a number of different books.

The structure used to represent rules (defined in Section 4.5) is also a source of experimental error. The definition of these structures is based on experience with the learning system, but are not validated as being the optimum structures for achieving generalized rules, or representing a specific category of annotation. However, we believe that only marginal improvements are possible given tailored rule-structures for each category of annotation.

The automatically created surface annotations that are used for obtaining structural and syntactic properties of text are also a source of error in these experiments (as a result of the small portion of error in their creation, as described in Chapter 3). We believe that special purpose rules are induced to handle these errors, mitigating the effects on overall accuracy, but potentially resulting in larger generalized rule-sets.

4.6.4 Results

This section describes the individual experiments performed during the investigation of the questions posed at the beginning of this section.

4.6.4.1 Analysis of generalized rule-set induction

The experiments in this section investigate whether a small set of patterns exist that identify a large portion of annotations in a particular category.

Can hierarchical rule-based learning induce a small set of patterns from annotated examples?

We provide the learning system with a set of example annotations and induce generalized rule-sets from these examples. We evaluate the accuracy with which annotations are created using the induced patterns. These experiments are conducted for identifying the *speech-verb*, *actor,* and *speaker* of a quote.

We plot the size of the induced rule-sets (and the number of correct annotations created) against the increasing number of examples for *speech-verb, actor,* and *speaker* annotations in Figures 4.26(a), (c), and (e) respectively. In all three figures, the size of the generalized rule-set is markedly smaller than the base rule-set (wherever the number of provided examples is greater than zero). This result indicates that a small set of patterns exist for these categories, and that these patterns are successfully induced using hierarchical rule-based learning.

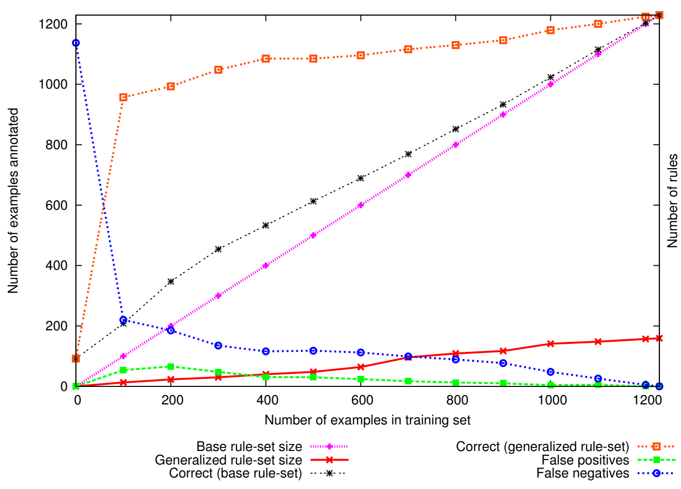
The generalized rule-sets correctly annotate more examples than provided for training in all three categories. For example, a rule-set generalized from only 100 examples correctly annotates nearly 1000 *speech-verb* examples. This indicates that the induced patterns correctly annotate additional examples in the text.

The lower bound for accuracy, determined as the number of example annotations correctly created by the base rule-set, is also indicated in Figures 4.26(a), (c), and (e). The base rule-sets generated for *speech-verbs* and *actors* reproduce the example annotations precisely, but the base rule-set for *speakers* produces a reduced number of correct annotations. This is attributed to weakness in the context model (described in Section 4.5). However, consistency checks on induced rules reduce this error, demonstrated in that more correct annotations are created by the generalized rule-set than the base-rule set for any number of examples.

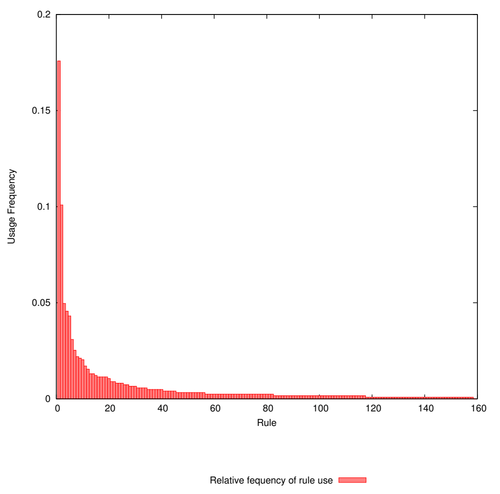
This experiment demonstrates that a small set of patterns exists for identifying annotations, that these patterns are induced using hierarchical rule-based learning, and that generalized rule-sets correctly reproduce the example annotations and create further correct annotations.

What is the nature of the induced patterns?

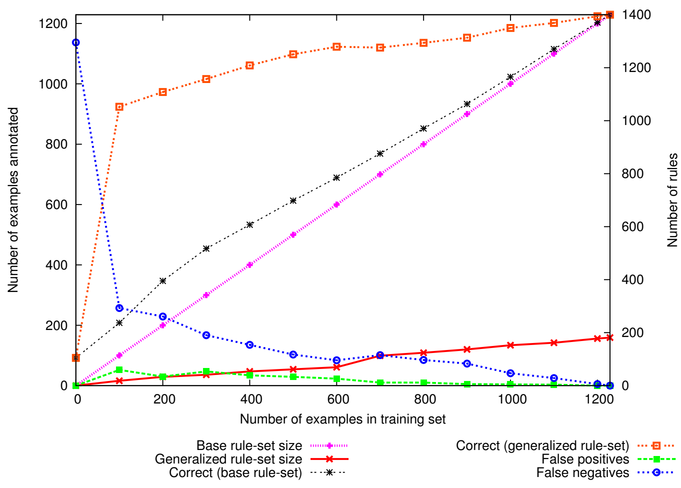
Given a generalized rule-set that is capable of reproducing all the annotated examples correctly, we examine the nature of the induced rules and their usage over the examples. The relative



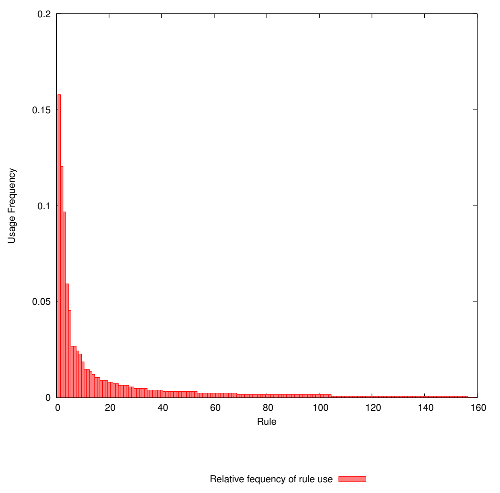
(a) Training and application for *speech-verb*



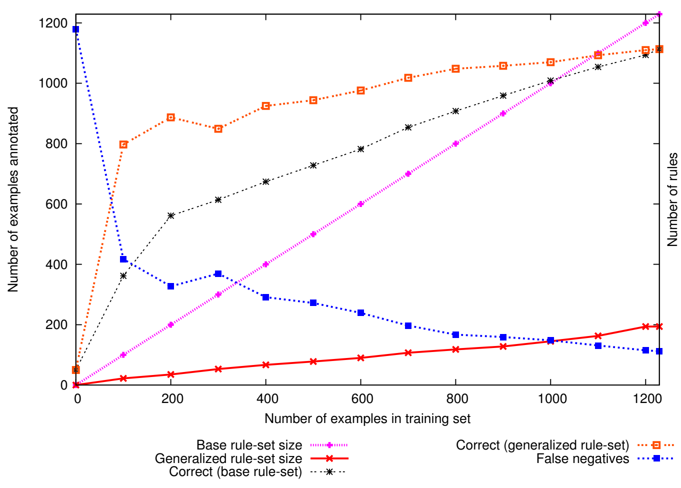
(b) Distribution of generalized rule usage (*speech-verb*)



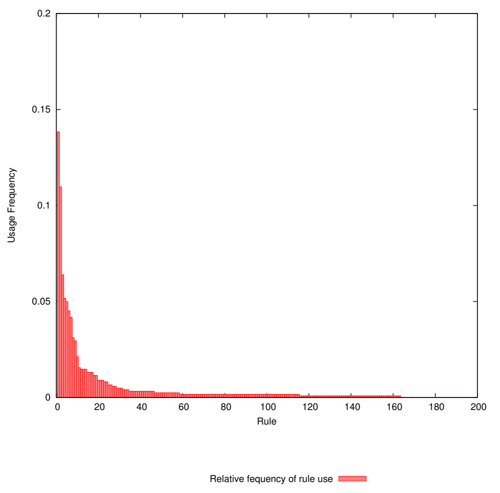
(c) Training and application for *actor*



(d) Distribution of generalized rule usage (*actor*)

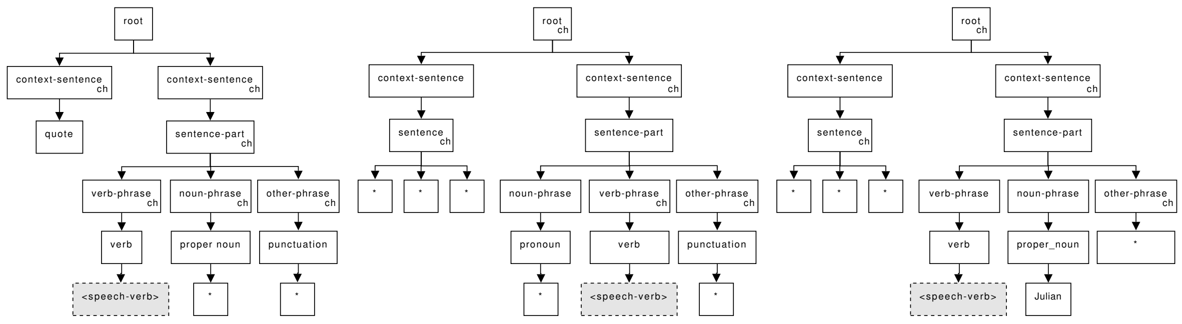


(e) Training and application for *speaker*



(f) Distribution of generalized rule usage (*speaker*)

Figure 4.26: Analysis of rule generalization for *speech-verb*, *actor* and *speaker*.



(a) Most frequently used rule. (b) Second most frequently used rule. (c) Third most frequently used rule.

Figure 4.27: Illustrations of the three most frequently used generalized rules for locating *speech-verb.*

distributions of generalized rule-use for *speech-verb*, *actor,* and *speaker* are presented in Figures 4.26(b), (d), and (f) respectively. In the case of the *speech-verb* a single rule correctly annotates 17.57% of the entire example set, while the top three rules are responsible for correctly annotating 32.62% of the entire example set. This shows that patterns summarize large portions of the example annotations.

The top three rules from the induced rule-set for *speech-verb,* are illustrated in Figure 4.27. The most frequently used rule indicates a quote followed by the *speech-verb*, which must be followed by a proper noun. The second most common rule is similar, but indicates that a pronoun should precede the *speech-verb.* The third most common rule is similar to the first, but is more specific, in that it indicates that "Julian" should be the token following the *speech-verb.* While applicable to the book over which the rule was induced, it is not applicable to other books that do not contain a character named "Julian". This suggests a question regarding the applicability of an induced rule-set over different books, as examined in Section 4.6.4.2.

Does the order in which examples are presented influence the accuracy of the generalized rule-set?

We investigate whether the order in which examples are provided affects the accuracy of the induced generalized rule-set by providing the learning system with a reversed set of example annotations. The number of correct *speaker* annotations using both a non-reversed and a reversed set of examples is plotted in Figure 4.28. There is a difference in the size and accuracy of the generalized rule-sets, but these differences are minimal.

4.6.4.2 The effect of the book type on pattern induction

This experiment investigates whether the type of book influences the induced rule-sets, and the applicability of rule-sets over different books.

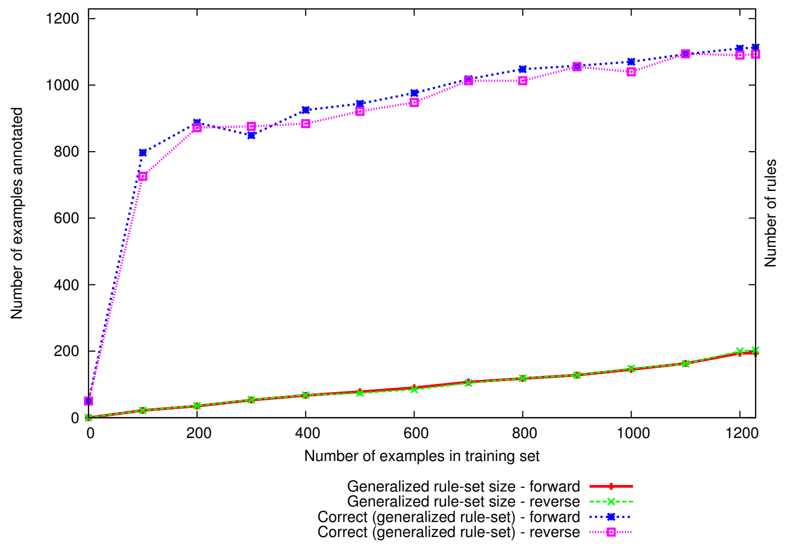


Figure 4.28: Generalized rule-sets resulting from differently ordered example sets.

Can hierarchical rule-based learning create accurate rule-sets from different types of books?

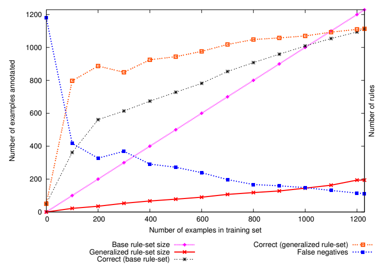
We answer this question by inducing rule-sets from books of similar and different types for identifying the *speaker*. Book 1 and Book 2 are of similar type, while Book 3 is of a different type to Books 1 and 2. We evaluate the induced rule-sets over the same books.

The success of applying a base rule-set and generalized rule-set to the same book is plotted against the size of the example set in Figures 4.29(a), (e), and (i). Correct annotations are created for all three books, indicating that for these cases hierarchical rule-based learning is effective over different types of books.

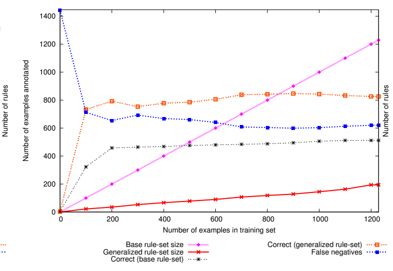
Can the rule-set learned over one book be used to create accurate annotations in different books?

To answer this question, we induce rule-sets using example *speaker* annotations from one book, and apply these rule-sets over other books. Book 1, Book 2, and Book 3 are used for this experiment, comparing the application of rule-sets to similar books (Book 1 and Book 2) and different books (Book 1 and Book 3).

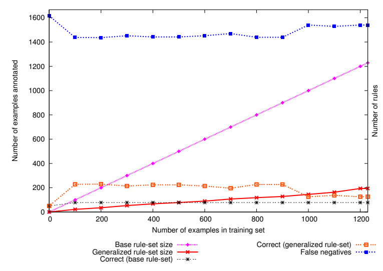
We are primarily interested in the number of correct annotations created by the generalized rule-set. The success (the curve labeled as "Correct (generalized rule-set)") of rule-sets from Book 1 applied to Book 2 is plotted in Figure 4.29(b), while Book 2 applied to Book 1 is plotted in Figure 4.29(d). In both cases, correct annotations are created in the other book, where the use of only 100 rules from one book results in a large number of correct annotations in the other book. Rule-sets created from Book 1 or 2 applied to Book 3 (Figure 4.29(c) and (f)), and Book 3 applied to Book 1 or 2 (Figure 4.29(g) and (h)), result in markedly fewer correct annotations. This demonstrates that a rule-set from one book creates accurate annotations in a different book of the same type.



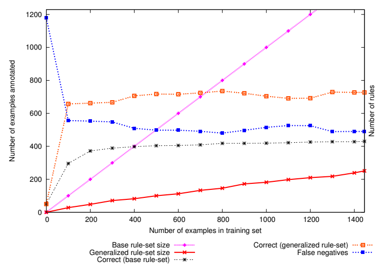
(a) Book 1 applied to Book 1



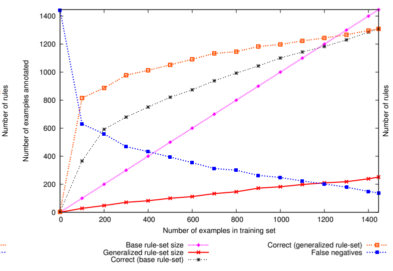
(b) Book 1 applied to Book 2



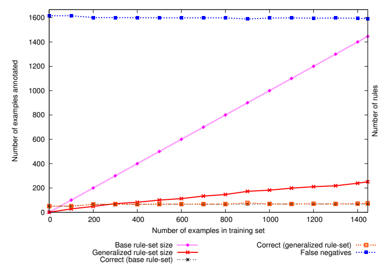
(c) Book 1 applied to Book 3



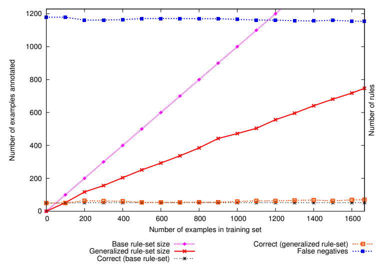
(d) Book 2 applied to Book 1



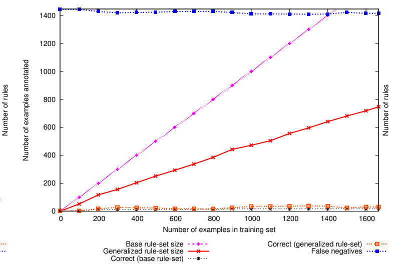
(e) Book 2 applied to Book 2



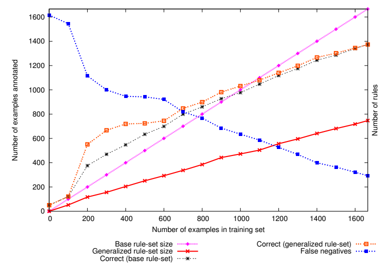
(f) Book 2 applied to Book 3



(g) Book 3 applied to Book 1



(h) Book 3 applied to Book 2



(i) Book 3 applied to Book 3

Figure 4.29: Application of rule-sets learned from a single book to the same or different books (*speaker*).

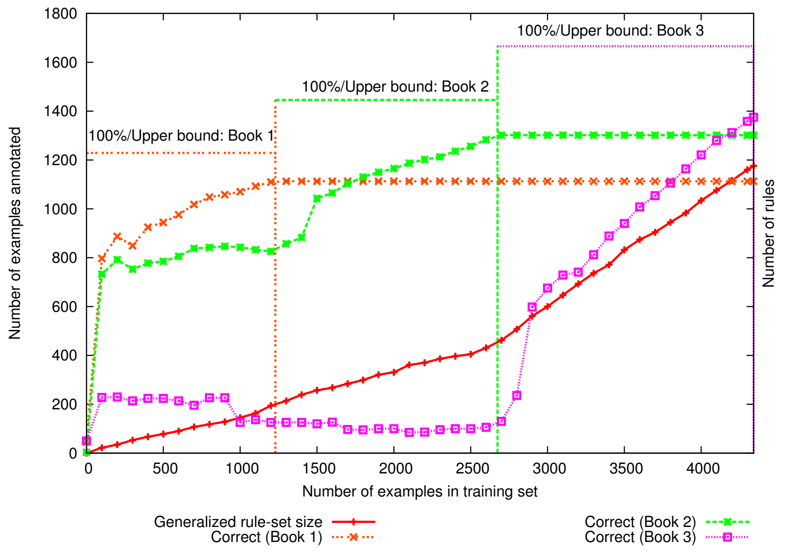


Figure 4.30: Application of a rule-set learned from different books applied to books of different type.

The same does not hold for books of a different type. We believe this is due to different writing styles used by different authors.

Does the accuracy of a rule-set improve over different books if examples from different books are used for training?

We provide the learning system with example *speaker* annotations from the three books, using all the examples from Book 1 first, then adding examples from Book 2 and Book 3 in succession. We chart the number of correct annotations created by each generalized rule-set over each book in Figure 4.30. As the full set of examples are used from Book 1 and Book 2, so the number of correct examples identified in these books reaches a maximum (which explains the leveling out of the curves for these books). There is a marked increase in the number of generalized rules as examples are added from Book 3. This further substantiates the differences in type between this book and Books 1 and 2. This demonstrates that different books result in rules that are not necessarily applicable to one another.

Once examples from all three books are used, annotations are created correctly in all three books. This demonstrates that rule-sets improve in accuracy over different books if examples are used from different books. This is significant because it shows that a single model can be created for annotating different books given a variety of examples.

The number of generalized rules resulting from examples from all three books (1176) is smaller than the total number of generalized rules created individually for each book (194 (Book 1) + 251 (Book 2) + 747 (Book 3) = 1192). This indicates that some patterns exist that are common between the three books, but which are not induced from each individual book.

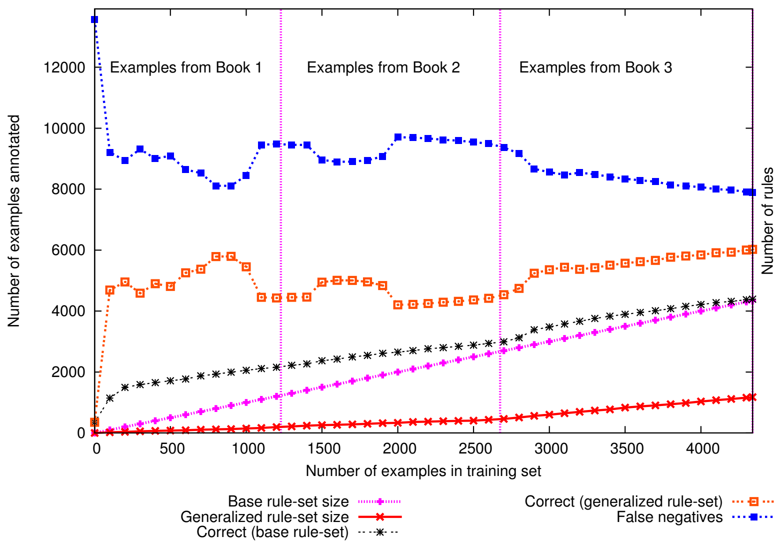


Figure 4.31: Application of a rule-set learned from different books applied to a large corpus of books of different type.

How applicable is an induced rule-set to a large corpus of fiction?

We investigate whether generalized rules apply to a large corpus of fiction containing 13 books of different type. We provide the learning system with example *speaker* annotations from Books 1, 2, and 3, using all the examples from Book 1 first, then adding examples from Book 2 and Book 3 in succession. We apply the generalized set to the corpus of 13 books.

The number of correct annotations is plotted against the number of training examples in Figure 4.31. After the addition of only 100 examples from Book 1, the number of correct annotations created exceeds 4000 (out of 13 913) over the larger corpus. This indicates that rules induced from just one book cater for annotations in many differing types of books.

We believe that some induced patterns represent fundamental rules in the English language, which explains their applicability over the large corpus of fiction.

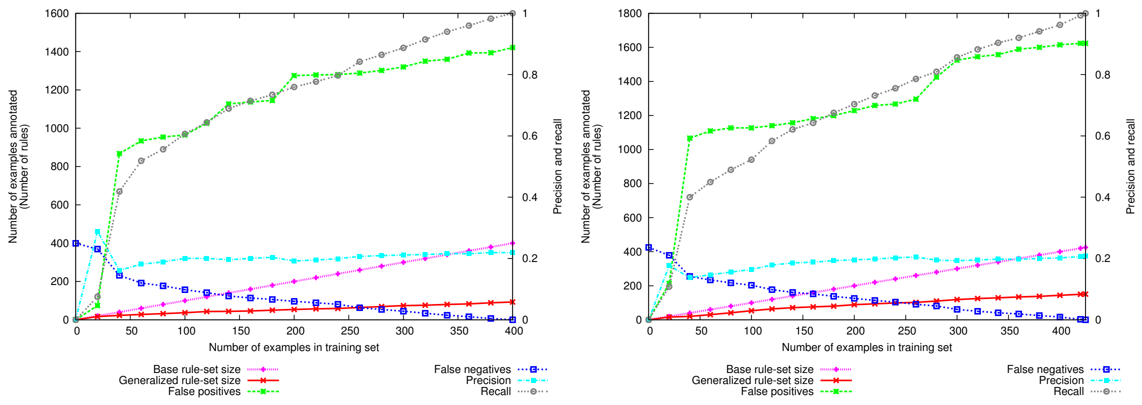
4.6.4.3 Composition of the example set

There are only 399 annotated Settings and 425 annotated Objects in Book 1. If a rule is created for each token in the book, the number of positive training examples accounts for less than 0.01% of the total example set (49 662 examples). The experiments in this section investigate the effect that positive and negative examples have on an induced rule-set.

Can an accurate generalized rule-set be created from only positive examples?

We provide the learning system with an example set containing only positive annotations, and evaluate the induced rule-set over the example set containing positive and negative examples. We use the Setting and Object annotation categories for these experiments.

The number of false positives, false negatives, precision, and recall are plotted against the number of training examples in Figures 4.32(a) and (b) for Setting and Object annotations. As



(a) Setting (b) Object

Figure 4.32: Training and application for Setting and Object annotations, using only positive examples.

the number of positive examples increases, the number of false negatives decreases. The number of false positives increases sharply using less than 50 positive annotations, indicating that negative examples are being erroneously annotated in these categories. This demonstrates that positive examples alone are not sufficient to create consistent annotations, highlighting the need for negative examples.

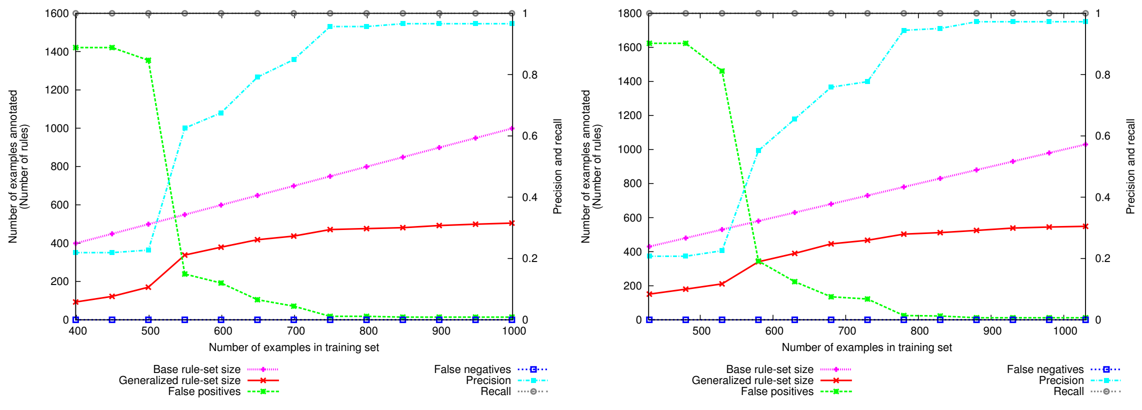
What quantity of negative examples is required to reduce the number of false positives?

We provide the learning system with an example set containing all positive examples and increasing numbers of negative examples. A continuation from the charts presented in Figures 4.32(a) and (b) are presented in Figures 4.33(a) and (b) with increasing numbers of negative examples. The addition of only a small number of negative examples (approximately 150) results in a sharp decrease in false positives. It is at this point that the negative rules become sufficiently generalized to apply to many negative examples. In total there are 49 263 negative Setting examples, and 49 237 negative Object examples. The number of false positives are reduced to less than 50 given approximately 400 negative examples for both categories. This demonstrates that far fewer negative examples are required to prevent false positives than what is available.

What effect does the inclusion of both positive and negative examples have on a generalized rule-set?

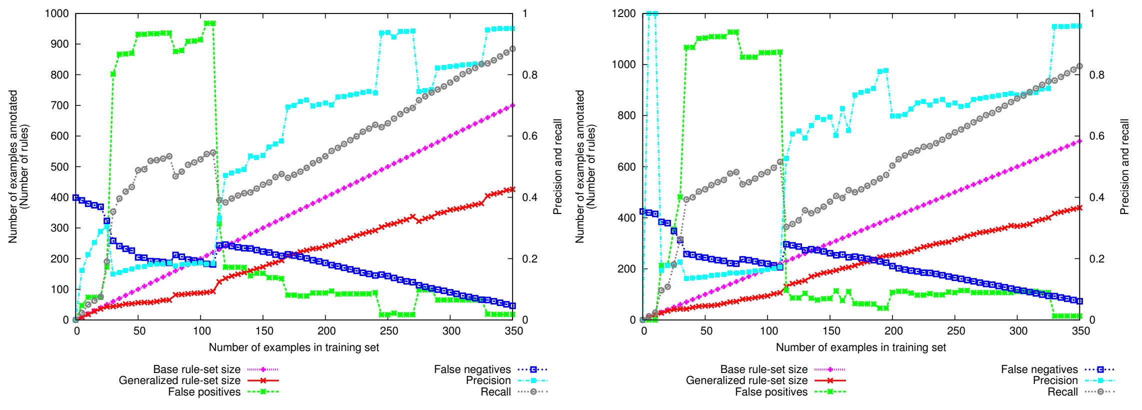
The question remains as to whether positive and negative examples impact one another when used simultaneously to create a generalized rule-set. We provide the learning system with an example set containing positive and negative annotations, and evaluate the generalized rule-set over the example set containing all positive and negative examples.

The number of false positives and negatives, along with precision and recall, are plotted in Figures 4.34(a) and (b). The number of false positives and false negatives does not decrease homogeneously with the addition of examples. For instance, between approximately 25 and 120 examples there is a marked reduction in the number of false negatives, due to the inclusion of a general rule that effectively identifies positive annotations. However, this rule does not discriminate



(a) Setting (b) Object

Figure 4.33: Training and application for Setting and Object annotations, using fixed positive examples and increasing negative examples.



(a) Setting (b) Object

Figure 4.34: Training and application for Setting and Object annotations, using increasing positive and negative examples.

negative examples, resulting in an increased number of false positives. This is corrected when a negative rule is induced that counters the positive rule, explaining the sharp decrease in false positives where approximately 120 examples are provided.

The discrete nature of the rule-based learning system is demonstrated in these experiments, specifically that single rules have the ability to dramatically alter the accuracy of the generalized rule-set. The results become more stable (smaller jumps in false positive or false negatives) as more example rules are provided for training.

This experiment demonstrates that training with both positive and negative examples produces erratic quantities of false positives and negatives, but less so as further examples are provided.

4.6.4.4 Rule structure for different annotation categories

The charts for Setting and Object in Figures 4.34(a) and (b) exhibit similar characteristics. In both cases a generalized rule-set smaller than the base rule-set is found that lowers the number

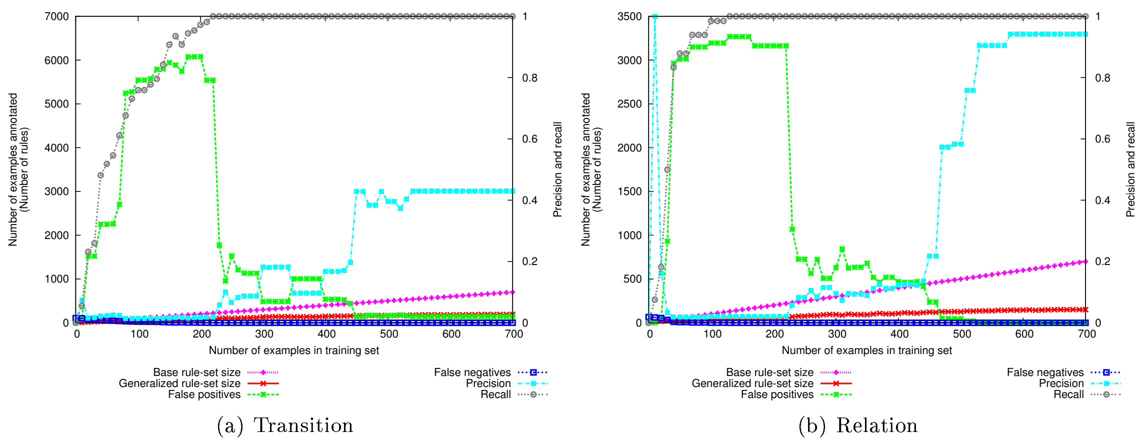


Figure 4.35: Training and application for Transition and Relation triggers using increasing positive and negative examples.

of false positives and false negatives. This demonstrates that patterns are induced in different categories of annotation using the same rule-structure (rules for Setting and Object have identical structures, as shown in Section 4.5). We investigate this observation further in this section.

Similar to Setting and Object annotations, Transition and Relation annotations also identify triggers in the text, and we investigate if the same pattern structures used for Setting and Objects can be used for identifying Transition and Relation triggers.

Can accurate patterns be induced for different categories of annotation using the same rule structure?

We provide the learning system with an example set containing positive and negative annotations for identifying Transition and Relation triggers, and evaluate the induced rule-set over the example set containing all positive and negative examples.

The number of false positives and negatives (along with precision and recall) are plotted in Figures 4.35(a) and (b) for Transition and Relation triggers. A generalized rule-set is always produced that is smaller than the base rule-set (for Subject, Object, Transition, and Relation). The number of false positives follows a similar trend, beginning with high numbers, but reducing sharply after enough negative rules are added.

The induced rule-sets and their ability to create annotations in four different categories demonstrate that the same rule structure can be used for different categories of annotation. This means that custom rule-structures need not be designed every time a new annotation category is defined.

4.6.4.5 Rule-set induction for different annotation qualifiers

Sections 4.6.4.3 and 4.6.4.4 provide evidence indicating that hierarchical rule-based learning is effective for identifying triggers in fiction text. We investigate whether accurate rule-sets are induced for text-references and semantic concepts.

We use Transition and Relation annotations for these experiments because both categories include a text-reference qualifier (*subject* and *object*) and both require a semantic-concept to be associated with the annotation (*type*). Every Transition and Relation annotation must have exactly

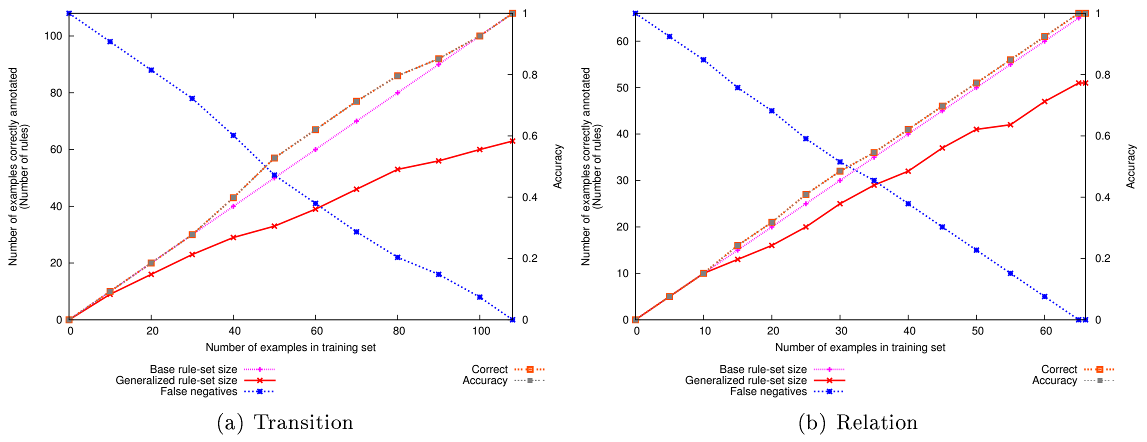


Figure 4.36: Training and application for the *subject* of Transition and Relation annotations.

one defined *subject, object,* and *type.* This means that there are no negative examples for these qualifiers.

Can patterns be induced for identifying text-references?

We provide the learning system with example Transition or Relation annotations, each of which has a defined *subject* or *object.* We evaluate the accuracy of the text-references produced by the induced rule-set.

Accuracy is plotted against the number of examples used for training in Figures 4.36(a) and (b) for the *subject* of Transitions and Relations respectively ("Accuracy" and "Correct" are equivalent in these graphs, and result in overlapping curves). In both cases a generalized rule-set is created containing fewer rules than the base rule-set. The generalized rule-set creates a greater number of correct annotations than examples provided, given enough examples. Transitions require greater than 30 examples, while Relations require greater than 10 examples (indicated by the divergence between the Correct/Accuracy curves and the Base rule-set curve).

Accuracy is plotted against the number of training examples in Figure 4.37 for the *object* of Relation annotations. The generalized rule-set is smaller than the base rule-set, and given enough training examples, creates more correct annotations than examples provided.

These experiments demonstrate that generalized patterns are induced for text-reference qualifiers that result in the creation of accurate annotations.

Can patterns be induced for associating semantic-concepts to annotations?

We provide the learning system with example Transition or Relation annotations, each of which has a defined *type.* We evaluate the accuracy of the semantic concept identified for each annotation produced by the induced rule-sets.

Accuracy is plotted against the number of examples used for training in Figures 4.38(a) and (b) for the *type* qualifier of Transition and Relation annotations respectively. Generalized rule-sets are induced for the *type* field in both the Transition and Relation category, and a greater number of correct annotations are created using the generalized rule-set than the total number of examples

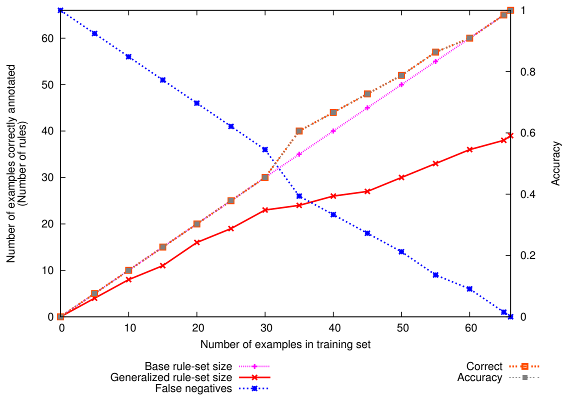
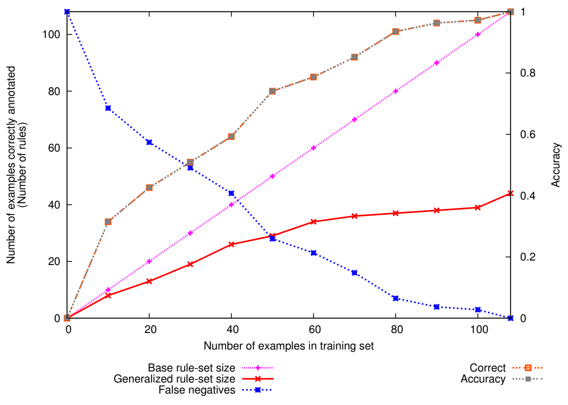
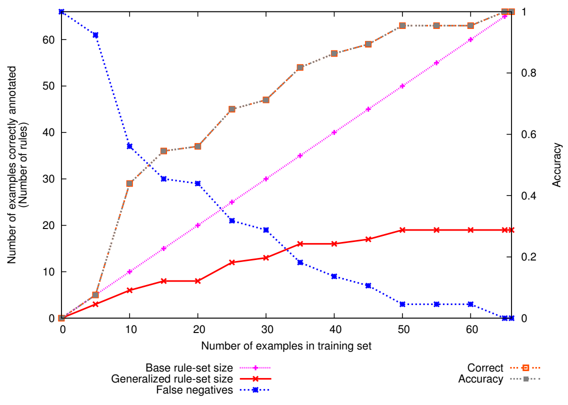


Figure 4.37: Training and application for the *object* of Relation annotations.



(a) Transition



(b) Relation

Figure 4.38: Training and application for the *type* of Transition and Relation annotations using increasing positive negative examples.

provided. For example, using only 10 examples, approximately 35 correct *type* annotations are created in the Transition category.

These experiments demonstrate that patterns are induced for creating accurate semantic-concepts. This is a significant contribution in the field of information extraction, because it allows semantic data to be associated with a text-based annotation without an external knowledge-base. These results indicate that the formulation of the rule structures is flexible enough to be augmented with non-textual data, and that the generalization process provides for the induction of patterns even for non-text data.

Can accurate patterns be induced for annotation qualifiers in different categories using the same rule structure?

The rule structures used for *subject* and *object* patterns in the Transition and Relation categories are identical. The successful induction of generalized rule-sets that create correct annotations in both categories (shown in Figures 4.36(a) and (b)) demonstrates that the rule-structure need not be modified for different categories of text-reference. An identical rule structure is used for inducing

Annotation	Qualifier	Positive Annotations	Positive Examples	Negative examples	Precision	Recall
Quote	*speech-verb*	1109	500	60	97.14%	89.62%
	actor	1109	500	60	97.27%	90.94%
	speaker	1229	500	-	-	76.92%
Setting	*trigger*	399	195	195	70.27%	52.13%
Object	*trigger*	425	210	210	67.06%	53.64%
Transition	*trigger*	108	54	250	96.55%	52.83%
	subject	108	54	-	-	56.60%
	type	108	54	-	-	76.41%
Relation	*trigger*	68	34	250	50.0%.	51.4%
	subject	68	34	-	-	51.4%
	object	68	34	-	-	57.3%
	type	68	34	-	-	77.9%

Table 4.5: Summary of results when training with 50% or less of the positive training examples.

patterns for *type* qualifiers of Transitions and Relations, demonstrating that the same applies for semantic concepts (shown in Figures 4.38(a) and (b)).

The implication of these results is that the addition of categories of annotation containing text-references or semantic concepts need not require the custom creation of rule-structures for these qualifiers.

4.6.4.6 Automatic creation of semantic annotations

All previous experiments verify that hierarchical rule-based learning induces patterns from example annotations, and substantial evidence is provided indicating that generalized rule-sets produce greater number of correct annotations than the number of examples provided. We investigate whether the automatic creation of annotations is truly supported in each semantic annotation category for the fiction-to-animation task.

Can hierarchical rule-based learning be used to automatically create semantic annotations?

The creation of positive examples is the most arduous task in creating manual examples for training (because negative examples need not be annotated). We consider automation to be successful if the provision of 50% (or less) of the total positive annotations results in the creation of more than 50% of the total number of annotations in the book.

We induce a rule-set using 50% or less of the positive annotations in each semantic annotation category. The precision and recall for every category is listed in Table 4.5 using the induced rule-set. A recall of greater than 50% is achieved in all categories, demonstrating that the induced rule-sets produce additional correct annotations.

The creation of triggers is the most difficult task, indicated by the low recall levels for this qualifier in all categories. However, highly effective rule-sets are induced in some categories, for example in identifying the *speaker* of a quote, and the *type* of a Transition or Relation. Recall rates are highest where the number of examples available is large (demonstrated by the Quote annotations), and we believe that higher recall levels are possible should the corpus be enlarged with examples in the Setting, Object, Transition, and Relation categories.

These results indicate that the creation of semantic annotations is automated using hierarchical rule-based learning. We speculate that a human annotator would not create 50% of the total positive annotations before training the learning system. We develop a boot-strapping method that uses the learning system to reduce the effort in creating annotations.

A boot-strapping method for reducing effort in the creation of semantic annotations

The question remains as to the practicality of hierarchical rule-based learning for automating the creation of semantic annotations. The manual creation of sufficient example data for inducing accurate rule-sets potentially requires significant effort (for example, in creating 50% of the total number of positive examples).

We propose a *boot-strapping* method that guides the rule-set creation process through an iterative validation of automatically produced results. We believe that validating a sub-set of annotations requires less manual effort than reading the original text and explicitly creating examples. The boot-strapping process begins with the manual creation of a small set of positive annotations in a particular category. These are presented as examples to the rule-based learning system along with a small set of negative examples (that are automatically obtained from the annotated extract). A rule-set is induced and applied to the entire book, the result of which is a larger number of annotated examples, some of which are correct, but most of which are likely to be false positives (as demonstrated in Section 4.6.4.3). An annotator reviews the automatically created annotations by selecting and marking some annotations as correct or incorrect. A new rule-set is created using the validated positive and negative examples. This rule-set creates more accurate annotations than the previous rule-set, and the process of correction is repeated until the annotator is satisfied with a set of annotations created by the system.

We demonstrate the validity of the boot-strapping method using the Object annotation category. The number of true positives and false positives is recorded each time the generalized rule-set is applied to the fiction text. The actions taken by a human annotator in creating Object annotations are summarized in Table 4.6. The human initially creates 15 positive annotations (corresponding to one chapter of the book). The generalized rule-set created from these examples results in a total of 254 annotations, 41 of which are correct. The annotator reviews these, and selects 10 erroneous annotations and 5 additional correct ones, and retrains the model. This results in 337 annotations, of which 10 are selected as being incorrect (more could be selected depending on the energy or time available to the annotator). The process repeats until the annotator is satisfied or runs out of time. The last set created contains 42 positive annotations and only 32 false positives. The annotator explicitly creates only 15 positive annotations, and *validates* only 40 automatically generated annotations.

We observe that automatically identified annotations help the annotator refine his or her own idea of which fragments of text belong in a certain category. During the experiment summarized in Table 4.6, additional annotations are suggested by the learning system that are correct, but were missed during the creation of the manually annotated corpus. In this respect, the results presented in Table 4.6 unfairly penalize false positives, where these actually represent inconsistencies in the original test data.

Human effort		Automation	
Positive annotations selected	False positives deselected	True positives presented (425 total)	False positives presented
Human creates 15 positive annotations; trains:			
15	0	41	213
Human selects 10 negative examples, and an additional 5 positive; trains:			
20	10	69	268
Human selects 10 negative examples; trains:			
20	20	27	15
Human selects 5 positive examples; trains:			
25	20	33	15
Human selects 5 positive examples; trains:			
30	20	72	159
Human selects 5 negative examples: trains:			
30	25	42	32

Table 4.6: Evidence in support of a boot-strapping process using the Object annotation category.

4.6.5 Summary of findings

The experiments presented in Section 4.6.4 provide insight into the characteristics of the rule-based learning system, with respect to the questions posed at the beginning of this section:

1. A small set of patterns exist in natural language that correctly identifies a large portion of annotations in a particular category. These patterns are induced using hierarchical rule-based learning.

2. The type of book makes a difference to the number and success of patterns induced by the learning system. Rules induced from one book are applicable to books of a similar type, but less so for books of different type. However, including examples from different types of books increases the accuracy of the induced rule-set over different books.

3. Example sets containing both positive and negative examples produce the best balance between correct annotation creation, and false positive elimination. However, only a small sub-set of negative examples is needed to remove the majority of false positives.

4. Accurate rule-sets are induced for different categories of annotation using the same rule-structure.

5. Accurate rule-sets are induced for different annotation qualifiers (including text-references and semantic-concepts) using the same rule-structure.

6. Hierarchical rule-based learning automates the creation of semantic annotations. Specifically:

   (a) 50% of the total examples are generalized into a rule-set that identifies more than 50% of the annotations in that category.

   (b) Hierarchical rule-based learning reduces the effort of creating annotations through the use of a boot-strapping technique.

4.7 Conclusion

This chapter presents hierarchical rule-based learning as a mechanism for automatically creating annotations over fiction text. This learning system induces and generalizes patterns from example annotations provided by a human, and applies these patterns to text to accurately identify additional annotations. Patterns are structured as trees that abstract input text on different levels. We present algorithms that automatically create these trees, generalize them, and apply them to examples in the creation of new annotations.

Hierarchical rule-based learning supports the automatic creation of annotations in fiction text. With particular reference to the problems listed in Section 4.1.1, we conclude the following:

1. Hierarchical rule-based learning induces patterns that reflect individual human discretion regarding the annotation task. Our automated mechanism can be refined to match a human's annotation style, and produce similar annotations to the examples provided.

   (a) Hierarchical rules express patterns for identifying annotations, using both structural and syntactic properties of text. These provide an effective mechanism for expressing patterns in the English language that identify annotations in fiction text.

   (b) A model for creating annotations is represented using a set of rules. A rule-set encapsulates a wide range of scenarios peculiar to a particular category of annotation. This is significant in that both common and rare scenarios can be accommodated in a single model.

   (c) The tree-structure of a rule provides for the abstraction of concepts to create generalized rules. Generalized rules provide for the application of a model to *unseen* text in the creation of accurate annotations.

   (d) Rules generalized using our algorithms are consistent (indicate the same answers as the original rules, match with the two rules that result in its creation, and do not conflict with other rules). This means annotation ability is not lost during the generalization process.

   (e) Tree matching determines when a rule applies to a portion of text (and create a corresponding annotation). This is made possible by the use of the same tree structure for representing rules and unseen text.

   (f) Generalized rule-sets never produce fewer annotations than non-generalized rule-sets. The significance is that additional correct annotations are created by a generalized rule-set, supporting automation.

2. Hierarchical rule-based learning induces models for multiple categories of annotation. This means that the creation of a rich intermediate representation (containing many annotation categories) is automated using this technique.

   (a) Rule-structures can be tailored for different categories of annotation. The core rule-set creation, generalization, and application processes are independent of the rule-structure. Further annotation categories can be defined without modifying the fundamental algorithms.

   (b) Rule-structures need not be customized every time a new annotation category is defined. Structures exist for successfully deriving rules in multiple categories.

3. Hierarchical rule-based learning qualifies annotations with text-references and semantic concepts. This means that annotations are automatically parametrized appropriately for future interpretation processes.
   (a) Tree-structures provide for the creation of rules that identify qualifiers for an annotation. The implication is that patterns induced by the learning system reflect structural, relational and semantic patterns in the English language.
   (b) Rules are effective in associating semantic concepts to annotations without the use of an external knowledge-base. This removes the need for subsequent discourse processing in certain categories of annotation.

Accurate models are induced over different types of fiction text using hierarchical rule-based learning. Some induced patterns are applicable across different books, but better quality models use examples sourced from a variety of book types. The boot-strapping process that uses hierarchical rule-based learning automates the creation of annotations, and also reduces the repetitive task of creating examples for training a model. We conclude that the creation of the intermediate representation in the form of annotated fiction text is automated using these methods.

A model induced using hierarchical rule-based learning can potentially be considered a detailed knowledge-base, which conflicts with our desired knowledge-poor paradigm for achieving the fiction-to-animation task. However, knowledge-centric systems use manually pre-constructed bases of *specialized* knowledge to guide the text analysis process (Coyne and Sproat, 2001; Lu and Zhang, 2002; Ma, 2006). Our technique is knowledge-poor in that it makes no prior assumption (in terms of encoded knowledge) about how to perform the task, but rather induces assumptions from a specific human. We believe that this distinction is in keeping with the knowledge-poor paradigm described in Chapter 1.

Hierarchical rule-based learning resembles existing techniques in its ability to learn models for different categories (although we cannot compare any two of these systems directly, as described in Section 4.2). Similar to existing information extraction techniques, our method exhibits a wide range of recall levels, depending on the category of annotation.

The research presented in this chapter contributes innovative work with respect to the text-to-graphics and information extraction domains:

- The use of a pattern-based information extraction technique for creating semantic annotations is novel in the text-to-graphics field. The only other text-to-graphics system that employs information extraction techniques is CarSim, which uses statistical machine learning algorithms (Johansson et al., 2005).
- We define a set of semantic annotation categories for identifying visual descriptions in fiction text. The formalization of these categories contributes to the text-to-graphics domain in that they are the first that are formally specified for handling fiction text.
- We contribute to the domain of information extraction by demonstrating that tree-structures are effective for representing, abstracting, and generalizing patterns in free text.

- We provide two innovative methods for generalizing patterns in natural language, using general and special purpose *childSubSeq* wild-cards.
- The ability to induce models regarding different qualifiers of annotations such as triggers, text-references, and semantic concepts contributes to the field of information extraction. The ability to associate semantic data (without the use of a knowledge-base) and thus perform a type of discourse analysis is a significant contribution.

Future work in this area includes the extension of the corpus of annotated fiction text, in terms of size and in terms of the categories of annotation created over the corpus.

Chapter 5

Constraint-based quantification of behaviour

This chapter investigates the problem of quantifying behaviour in a virtual environment from a constraint-optimization perspective (described in Section 5.1). The use of a constraint optimization method is motivated in Section 5.2 by comparing alternative approaches to this problem. We present an innovative optimization technique that locates solutions (or solution approximations) to constraint systems defined over contiguous intervals of time. This technique is based on interval arithmetic, a brief overview of which is provided in Section 5.3. We develop an interval-based optimization approach for locating solutions (or solution approximations) to constraint systems in Section 5.4. Properties of the interval-based optimization approach are investigated using a suite of benchmarks in Section 5.5. We present conclusions and contributions resulting from our innovations in Section 5.6.

5.1 Introduction

5.1.1 Problem statement

Fiction books describe the set of entities that exist in an environment and the behaviour of these entities. We use the term behaviour to refer to the positioning of an entity and its motion within an environment. Behaviour is identified in fiction text using categories of annotation that suggest spatial constraints between entities (further details on interpreting annotations are provided in Chapter 6). We investigate the problem of automatically quantifying behaviour in a virtual environment so that it conforms to spatial constraints prescribed by annotations. In particular:

1. We define virtual environments in terms of space and time, which introduces the problem of quantifying behaviour that conforms to constraints specified over contiguous intervals of time.
2. An animated film is constructed as a sequence of clips, potentially filmed out of order. We investigate the quantification of behaviour at any time-instant in a virtual environment,

Anne didn't very much like a big brown cow who came up close and stared at her, but it went away when Daddy told it to.
...
He tiptoed by him to the table behind his uncle's chair.

Figure 5.1: Example fiction text indicating spatial positioning and behaviour from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

without simulating every instant of time in an incremental fashion until the desired state for filming is reached.

3. We believe that descriptions in fiction text imply under-constrained behaviour, the quantification of which involves a search for valid options. We investigate the problem of searching, while directing the search towards valid behaviour.

4. We speculate that the more complex the behaviour in the scene, the more difficult the task of quantifying the behaviour becomes. This translates to an increased search time. We investigate the problem of providing behaviour configurations that only approximate the desired behaviour, but which are derived in spite of bounds on computation time.

5. We anticipate that annotations in fiction books are potentially flawed, either because of inconsistencies in the original narrative or because of inconsistency in manual or automated annotation creation. Flawed annotations potentially translate to conflicting constraints, and we investigate the problem of approximating behaviour where such conflicts occur.

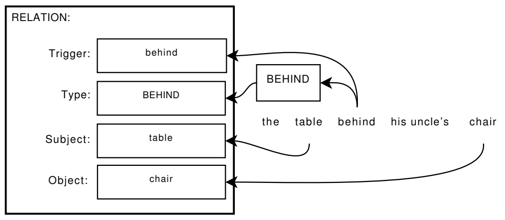
We investigate the above problems under the assumption that a process exists for translating annotations into constraints (see Chapter 6 for this aspect of the problem). This chapter is concerned only with *constraint optimization* as a method for identifying precise numerical solutions that quantify visual behaviour.

5.1.2 Problem formulation

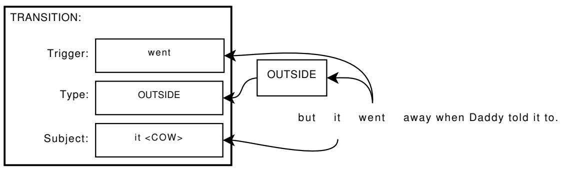
Descriptions found in fiction text imply the positioning or behaviour of entities within a scene. An example of fiction text containing such descriptions is presented in Figure 5.1. These descriptions are identified using specific categories of annotation (for example, Transition or Relation annotations, as demonstrated in Figure 5.2). The Relation annotation illustrated in Figure 5.2(a) specifies that a "table" is "behind" a "chair". The problem in this case is to determine exact coordinates for the "table" and "chair" objects so that the "behind" spatial-relation holds. The Transition annotation illustrated in Figure 5.2(b) specifies that a "cow" is "outside" the environment, indicating that an appropriate motion for the cow must be calculated so that this behaviour is visualized.

The quantification of behaviour involves calculating exact values that describe an entity's behaviour in a virtual environment. We define an environment in terms of its dimension, its boundaries, and the interval of time over which the environment exists. We name each environment a *scene*, defined as follows:

Definition 5.1. Define $\mathbb{S}\,(dimension, boundary, time)$ to be a *scene* specified as a Cartesian space in a number of *dimensions* where each dimension is bounded according to a *boundary*. The scene exists over the duration specified by *time*.



(a) Relation annotation



(b) Transition annotation

Figure 5.2: Example annotations from fiction text.

- Entity M has a trajectory defined as $\mathbf{r}^M(t) = (1-t)\mathbf{p}_0^M + t\mathbf{p}_1^M$
- Entity N has a trajectory defined as $\mathbf{r}^N(t) = \mathbf{p}_0^N$

Example system of constraints over the two trajectories:
M near N: $||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N + \alpha)^2 < 0\,\forall t \in [t_{start}, t_{end}]$
M noCollide N: $||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N)^2 > 0\,\forall t \in [t_{start}, t_{end}]$
...

Figure 5.3: Example set of time-based mathematical constraints.

For example $\mathbb{S}(2, \{[-20, 20], [-20, 20]\}, [0, 15])$ is a scene in two dimensions, where each dimension is defined over the intervals $[-20, 20]$. This scene is specified to last for 15 seconds.

A scene contains geometric models representing entities described in the fiction text. Text annotated in categories such as Relation or Transition *constrain* a model's behaviour in a scene. For example, the Relation in Figure 5.2(a) describes a constraint over the behaviour of the "chair" and "table" models, specifying that the "chair" must be *behind* the "table", no matter where the "table" model is placed in the scene. The Transition illustrated in Figure 5.2(b) specifies that the "cow" model must leave the scene, which is interpreted as a constraint on the behaviour of the "cow" that requires the model to be outside the boundaries of the scene at a certain time.

A scene is defined as an n-dimensional Cartesian space, and this allows spatial constraints to be expressed as symbolic functions. We describe the behaviour of entities in a scene using trajectories that are constrained according to the annotations. Phrased in this manner, the problem of behaviour quantification is one of constraint satisfaction. Example trajectories are presented in Figure 5.3, along with examples of the symbolic functions that constrain these trajectories.

Scenes are defined in terms of a space interval (*boundary*) and a time interval (*time*). This means that values must be found for the variables defining the trajectories that satisfy the constraints both spatially, as well as over a contiguous interval of time. This is represented in Figure 5.3 by the expression $\forall t \in [t_{start}, t_{end}]$. The inclusion of a temporal aspect to the formulation of constraints presents a *universally quantified constraint satisfaction problem* (Benhamou et al., 2004; Ratschan, 2006), where the time-dimension is said to be universally quantified.

We assume that constraint systems such as those illustrated in Figure 5.3 are created automatically from annotations (see Chapter 6 for details on this process). An automated process has the potential to create constraints that *conflict* within the same system, in which case no solution

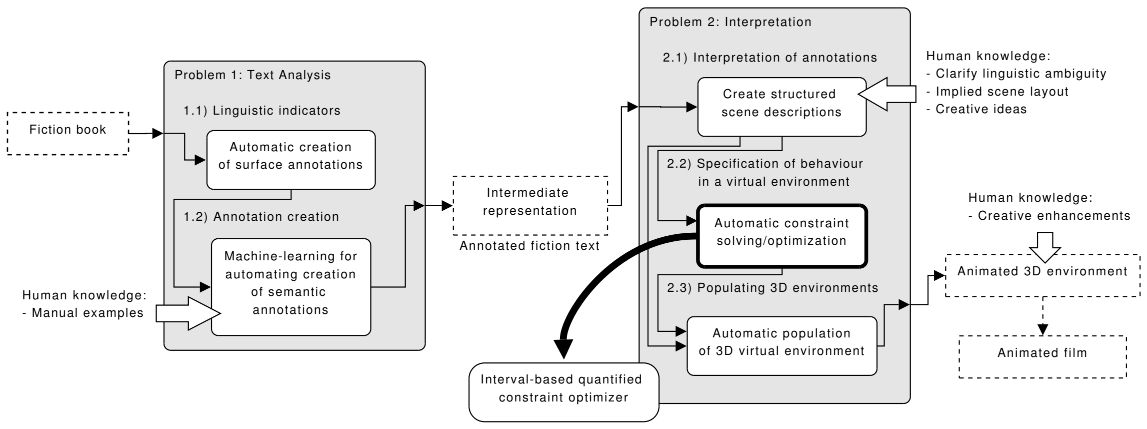

Problem 1: Text Analysis
1.1) Linguistic indicators
Fiction book
Automatic creation of surface annotations
1.2) Annotation creation
Machine-learning for automating creation of semantic annotations
Human knowledge:
- Manual examples
Intermediate representation
Annotated fiction text
Problem 2: Interpretation
2.1) Interpretation of annotations
Create structured scene descriptions
Human knowledge:
- Clarify linguistic ambiguity
- Implied scene layout
- Creative ideas
2.2) Specification of behaviour in a virtual environment
Automatic constraint solving/optimization
2.3) Populating 3D environments
Automatic population of 3D virtual environment
Interval-based quantified constraint optimizer
Human knowledge:
- Creative enhancements
Animated 3D environment
Animated film

Figure 5.4: Context of the interval-based quantified constraint optimizer with respect to the fiction-to-animation problem.

exists. We prefer an approximate solution to be presented rather than no solution, because this guarantees quantified behaviour that can be used to populate a scene (no matter how flawed).

The problem investigated in this chapter is automatically finding values for entity trajectories that satisfy a system of constraints. Constraints are expressed as non-linear, symbolic functions that include a universally quantified time variable and spatial variables bounded to intervals. The solver must produce solutions that satisfy the constraints over contiguous intervals of time (where constraint systems are consistent), or produce approximations of quantified solutions (where constraint systems are inconsistent).

5.1.3 Context

The research presented in this chapter examines one sub-problem of the interpretation task in the fiction-to-animation process. The context of this problem within the conversion process is illustrated in Figure 5.4. We examine a method that generates precise numerical values that quantify behaviour in a virtual environment. These values are used in a subsequent component for populating a virtual environment that corresponds to the fiction text. The input to this process is a set of constraints derived automatically from an annotation interpretation module (described in Chapter 6).

5.2 Related work

We name the problem investigated in this chapter *spatial reasoning*, and this term describes problem of deriving exact values that quantify an entity's behaviour in a virtual environment so that it visually corresponds to the behaviour described by annotations. Spatial reasoning is characterized by the fact that behaviour is specified using high level (non-detailed) instructions, while the precise values defining behaviour in an environment are left to be calculated by an automated process.

We categorize existing spatial reasoning technologies according to the manner in which precise values are obtained. We identify two categories, namely those that formulate these values incrementally through direct manipulation of the environment (*environment-sensitive*), and those that phrase the problem in terms of symbolic expressions that are solved independently of the environment (*environment-independent*).

5.2.1 Environment-sensitive reasoning

We broadly define environment-sensitive reasoning as the collection of techniques that progress towards acceptable behaviour by iteratively updating and evaluating behaviours of entities in the environment. These methods are characterized by a tight coupling with the target environment, usually including direct manipulation and simulation of the environment in determining values that describe behaviour. Each state in a time-based environment is the culmination of all previous states.

We examine environment-sensitive reasoning in two categories, namely those that are concerned with *static* environments (containing entities without motion), and those that are concerned with *dynamic* environments (containing moving and non-moving entities).

5.2.1.1 Static environments

Spatial reasoning in a static environment is concerned with specifying the location and orientation of entities within a finite space. Procedural methods such as those described by Parish and Muller (2001) use a suite of hand coded rules to guide the placement of roads and buildings in an environment. This method is environment-sensitive in that the placement of a road segment is affected by the placement of previous road segments in the environment.

An alternative method for spatial reasoning exists where the environment is described by a set of constraints. Values that satisfy the constraints define the correct geometric layout of entities in the environment. Environment-sensitive methods for performing the reasoning include *stochastic* and *constructive* methods (Le Roux and Gaildrat, 2003) (however, numerical methods also exist for these problems as shown in Section 5.2.2). Stochastic methods progressively refine the layout of entities in an environment by perturbing entity locations and evaluating the subsequent environment layout (according to the degree to which they satisfy the constraints) (Xu et al., 2002; Sanchez et al., 2003). Constructive methods build the environment incrementally, by placing a single entity in the environment and then enumerating and testing every possible location for the next entity (Baykan and Fox, 1991; Kwaiter et al., 1998; Bonnefoi and Plemenos, 1999; Le Roux and Gaildrat, 2003), then selecting only valid options.

5.2.1.2 Dynamic environments

Spatial reasoning in a dynamic environment is an example of the motion planning problem. Motion planning initially emerged as a field of study in robotics with the aim of determining a path through an environment that avoids collisions between an autonomous agent and possible obstacles (and is sometimes referred to as the Piano Mover's problem in the field of Artificial Intelligence (Garber and Lin, 2003)). The motion planning problem is phrased in terms of a start-location and a goal-location within a workspace that contains several obstacles. The task of an autonomous agent is to find a collision-free path from the start to the goal.

We classify motion planning techniques in two categories, namely *global* and *local* techniques, based on the information available to the agent during the creation of a path (Garber and Lin, 2002).

Global methods assume the prior existence of a populated environment, as well as global knowledge regarding the layout of obstacles within it (Foskey et al., 2001; Garber and Lin, 2002). *Probabilistic road-map* algorithms are examples of global methods and function by selecting random samples across the workspace, and then connecting samples with paths (Overmars, 1992; Kavraki and Latombe, 1994; Overmars and vSvestka, 1995; Salomon et al., 2003). The path from the start location to the goal location is determined by searching the resulting graph using algorithms such as A* (Calomeni and Celes, 2006). This algorithm assumes global knowledge in the sense that the location of obstacles across the environment is available for use by the agent in creating a graph. Paths that result in collisions with these obstacles are detected and removed. Probabilistic road-map algorithms cater for static obstacles (Kavraki and Latombe, 1994; Pettré et al., 2003) and dynamic obstacles (Gayle et al., 2005; van den Berg and Overmars, 2006), and many motion planning problems make use of some variant of this technique (Koga et al., 1994; Nieuwenhuisen and Overmars, 2003; Pettré et al., 2003). Another example of a global motion planning technique is *cell decomposition,* which breaks the workspace into a number of simple cells (Kuffner Jr., 1998; Latombe, 1999), using shortest-path algorithms to determine paths from the start-location to goal-location.

Local motion planning techniques are employed when an agent does not have global knowledge of the workspace, and repeatedly observes the environment to detect and avoid obstacles. An example is the potential-field method that associates attractive and repulsive forces to obstacles in the environment. The agent is attracted to the goal location, while repulsed from obstacles (Drucker and Zeltzer, 1994; Hong et al., 1997; Ge and Cui, 2002).

Motion planning techniques exist that specify constraints over the motion of an entity, in addition to the task of finding a path that avoids collisions. Examples of such constraints include forcing agents to adhere to physical laws such as gravity and volume preservation (Gayle et al., 2005), or for maintaining connectivity between joints in articulated models (Garber and Lin, 2002). These constraints are used in combination with local or global path planning techniques, where constraints are evaluated to determine whether the next discrete move of the entity satisfies or minimizes the set of constraints.

Global motion planning techniques are environment-sensitive because they require repeated queries to the environment for setting up a path, especially where other dynamic entities occur in a scene. Local motion planning techniques are environment-sensitive because the location of an entity in an environment is determined in relation to its previous location. If a particular time-instant in an environment is required (for filming), then every environmental state prior to the required time-instant must be simulated to that point.

5.2.2 Environment-independent reasoning

Techniques for spatial reasoning exist that phrase the problem in terms of symbolic functions that are solved analytically or numerically. These techniques remove the need for simulation or tight coupling with the environment, and transform behaviour quantification into a general constraint satisfaction problem. All that is required is a set of symbolic constraints, a set of variables, and initial domains for each variable. An independent constraint solver is applied to locate a solution. We consider approaches for quantifying behaviour in static and dynamic environments.

5.2.2.1 Static environments

Behaviour of entities in a static environment is concerned only with layout. Constraints are formulated as symbolic inequalities that define the layout of the environment, and solving techniques such as linear programming are used to determine values that satisfy these constraints (de Vries and Jessurun, 2000).

5.2.2.2 Dynamic environments

Reasoning regarding entities with motion is complicated by the time dimension. In practice, time is represented as another variable in each constraint. However, solutions must satisfy the constraint over the entire interval of time. The classical approach is to discretize the time dimension into a sequence of values, and perform constraint solving at each discrete point (Witkin and Kass, 1988). This approach is used for text-to-graphics research by the CarSim system (Johansson et al., 2005).

The discretization of time-intervals is avoided in the field of automated camera control by representing time as contiguous *intervals* rather than breaking the time dimension into discrete points (Jardillier and Languénou, 1998; Benhamou et al., 2004). Interval methods for motion planning are based on *Interval Analysis*, the field of mathematics concerned with the use of intervals of real numbers rather than finite values during calculations. This allows a time variable to be represented as a contiguous interval, for example $T = [t_{start}, t_{end}]$. An advantage of this approach is that solutions to constraint systems can be found that are guaranteed to be valid over contiguous intervals.

Interval arithmetic is used in a number of applications in computer graphics because of its usefulness in solving systems of constraints, namely ray tracing of parametric surfaces (Toth, 1985; Mitchell, 1991), contour tracing and implicit surfaces (Mitchell, 1991), collision detection (Snyder et al., 1993; Redon et al., 2002), and approximations of offsets/bisectors/medial-axes (Oliveira and De Figueiredo, 2003).

5.2.3 Interval-based constraint solving as an environment-independent spatial reasoning mechanism

Both environment-sensitive and environment-independent methods have advantages with respect to the task of spatial reasoning. Environment-sensitive methods do not require formulations of explicit expressions that require solving, and are also useful for real-time motion planning. In contrast, environment-independent methods are characterized by the following advantages:

- A solution determined by an environment-independent method describes entity behaviour at any point in the scene, without the need for prior states to be calculated first. This is the primary difference to environment-sensitive methods that require the environment to be repeatedly updated, with the effect that every state of the environment is a function of all previous states. In terms of fiction-to-animation conversion, a solution provides the ability to instantiate a scene at any point, without pre-constructing the entire duration of the environment.

- Environment-independent methods separate the problem domain from the solution strategy, unlike environment-sensitive methods in which the environment is repeatedly updated and evaluated during the reasoning process. Environment-independent methods pose the problem as a set of domain independent constraints that are solved using an arbitrary solver. This solver is independent of the problem domain, and can be improved or replaced without modifying the original problem.

Based on these observations, we employ an environment-independent method for performing spatial reasoning. This means that behaviour of entities in a scene is described by systems of constraints that require solving. Interval-based solving methods have benefits unavailable to other forms of constraint solving (for example linear programming (de Vries and Jessurun, 2000), or Newton-Rhapson solving (Witkin and Kass, 1988)):

- Time (and other quantities) is represented as contiguous intervals using interval constraint solving. This is suitable given our definition of a scene that specifies the space and time in terms of intervals (Definition 5.1 on page 107).
- Interval arithmetic avoids the need to sample the time dimension at discrete points. Techniques exist in interval-based constraint solving that are capable of providing solutions to *quantified* constraint systems, that is, constraints that are specified over contiguous intervals of time.
- Interval-based techniques solve an entire quantified system in a single operation, rather than invoking a solver at each discrete time instance. This provides for more efficient solving.
- The output from an interval constraint solving process is a set of contiguous intervals, any value from which represents a solution to the constraint system. This provides a range of valid behaviour options for a scene.

Most applications of interval arithmetic in constraint solving are concerned with locating a solution to a system of equations or inequalities, specifically in the camera control domain (Jardillier and Languénou, 1998; Benhamou et al., 2004). However, we presume that constraints are automatically generated by the fiction-to-animation system, and that some automatically generated constraint systems have the potential to be inconsistent. Existing constraint solvers perform an exhaustive search of the configuration space before concluding that no solution exists. An *optimization* mechanism is preferable, so that trajectories are specified even in the absence of a consistent solution.

Optimization techniques exist that are based on interval arithmetic (Snyder, 1992; Huyer and Neumaier, 1999; Dolgov, 2005), but these methods do not cater for universally quantified variables. This chapter describes an optimization technique for constraint systems containing universally quantified variables. This technique is based on existing universally quantified constraint solving mechanisms (Benhamou and Goualard, 2000; Benhamou et al., 2004).

5.3 Interval analysis and universally quantified constraint solving

This section provides a brief introduction to interval arithmetic, after which techniques in interval-based constraint solving are described. We focus on quantified interval-based constraint solving, primarily using techniques developed by (Benhamou et al., 2004) and (Benhamou and Goualard, 2000) and relevant portions of this research are described here. Formal descriptions and proofs of these processes are not reported in this exposition, but are located in the original sources (Benhamou et al., 1994; Benhamou, 1995; Benhamou and Older, 1997; Benhamou et al., 1999; Benhamou and Goualard, 2000; Benhamou et al., 2004).

Interval Analysis was pioneered by Moore (1966) as a method for coping with round-off errors that occur during calculations performed by machines with limited representation for floating point numbers. Real numbers are replaced by intervals that contain them and whose bounds are described using computer representable numbers. For instance, the constant π is represented as $[3.14, 3.15]$ rather than a single floating point number rounded up or rounded down (Benhamou et al., 2004). Complete theoretical expositions in interval arithmetic and analysis are provided by Moore (1966) and Neumaier (1990), and the following sections describe only those aspects necessary for investigating the quantified constraint satisfaction problem.

5.3.1 Interval analysis

This section develops the fundamentals of interval analysis that are used for subsequent constraint solving techniques. Let $\mathbb{R}$ be the set of real numbers. $\mathbb{F}$ is the set of computer representable floating point numbers, and is a subset of the set of real numbers such that $\mathbb{F} \subset \mathbb{R}$. A set of real numbers is represented on a computer by specifying floating point bounds, as described by the following definition:

Definition 5.2. *Floating point interval* (Moore, 1966): A ***floating point interval*** is a set of real numbers bounded on either side by floating point numbers. Formally[1], given $g \in \mathbb{F}$ and $h \in \mathbb{F}$, then $[g, h] = \{r \in \mathbb{R} | g \leq r \leq h\}$. Therefore, the interval $[g, h]$ contains every real number between (and including) g and h.

The set of all intervals is denoted as $\mathbb{I}$, and a single interval is denoted using a capital letter (for example, $I = [g, h]$). For the remainder of this chapter, unless otherwise stated, small letters represent floating point numbers. It is often necessary to refer to the lower or the upper-bound of an interval, and this is indicated using $lower(I)$ and $upper(I)$ respectively.

The set of primitive operations used for real numbers are extended to interval arithmetic in a conservative manner. All real numbers that could possibly occur as a result of the operation are included in the result:

Definition 5.3. *Interval Extension.* Define $\Diamond(x_1, ...x_n) \mapsto \mathbb{R}$ to be a real-valued operation consisting of n real-valued operands ($x_i \in \mathbb{R}$). An *interval extension* of operation $\Diamond$ is denoted as

[1]Similar to Moore (1966), this exposition uses the notation $\{x|P(x)\}$ for "the set of x such that the proposition $P(x)$ holds.

$\blacklozenge(X_1, ..., X_n) \mapsto \mathbb{I}$, which is the corresponding operation extended to function over n floating-point intervals ($X_i \in \mathbb{I}$), such that:

$$x_1 \in X_1, ..., x_n \in X_n \Rightarrow \lozenge(x_1, ...x_n) \subset \blacklozenge(X_1, ..., X_n)$$

Definition 5.3 means that an interval extension of a real valued operation produces an interval containing the result produced by the corresponding real-valued function if the operands of the real-valued function fall within the corresponding interval operands. *Natural interval extensions* of elementary operations are defined by Moore (1966) as follows:

Definition 5.4. Let $A = [a, b]$ and $B = [c, d]$. Natural interval extensions of real-valued elementary operations are defined as follows:

- Addition: $A \oplus B = [a + b, c + d]$
- Subtraction: $A \ominus B = [a - d, b - c]$
- Multiplication: $A \otimes B = [min(ac, ad, bc, bd), max(ac, ad, bc, bd)]$
- Division: $A \oslash B = [a, b] \otimes [1/d, 1/c]$ if $0 \notin [c, d]$, undefined otherwise

The natural interval extension of a real-valued function $f(x_1, ..., x_n)$ is the function $F(X_1, ..., X_n)$ constructed by replacing each real-valued elementary operation in f with a corresponding natural interval extension, and replacing each real value x_i with a corresponding interval X_i.

Interval extensions of real-valued functions have an important property, described by the *fundamental theorem of interval analysis* (Moore, 1966; Benhamou et al., 1994). If $F(X_1, ..., X_n)$ is an interval extension of the real valued function $f(x_1, ..., x_n)$, then the interval produced by $F(X_1, ..., X_n)$ completely contains any real-value produced by $f(x_1, ..., x_n)$ as long as any real number x_i is in the corresponding interval X_i:

$$x_1 \in X_1, ..., x_n \in X_n \Rightarrow f(x_1, ...x_n) \subset F(X_1, ..., X_n)$$

The fundamental theorem of interval analysis leads to an important property, namely that an interval extension of a real-valued function is *inclusion monotonic.* This means that the interval extension of a function is guaranteed to return an interval containing the real-valued result for any $x_i \in X_i$. This property implies that interval arithmetic is a convenient mechanism for calculating the range of a real-valued function over a specific domain. However, natural interval extensions are conservative in their approximation. While an interval returned by an interval extension of a function contains the *complete* range of the real-valued function, it potentially contains values that are not in the range. Alternatives to natural interval extensions exist that more tightly bound the range of a function, but are more computationally expensive to implement (van Hentenryck et al., 1997).

Union and intersection operators are often used in interval constraint solving. Let I_1 and I_2 be two intervals, then the union of the two intervals is the smallest interval containing both I_1 and I_2:

$$I_1 \cup I_2 = [min\,(lower(I_1), lower(I_2))\,, max\,(upper(I_1), upper(I_2))]$$

The intersection of two intervals is the largest interval common to I_1 and I_2. Let $I_U = [max\,(lower(I_1), lower(I_2))\,, min\,(upper(I_1), upper(I_2))]$, then the intersection is one of two options:

$$I_1 \cap I_2 = \begin{cases} I_U & \text{if } lower(I_U) \leq upper(I_U) \\ undefined & \text{if } lower(I_U) > upper(I_U) \end{cases}$$

An interval $I = [a, a]$ is called a *degenerate interval*, and is used to represent constants. A *canonical interval* is an interval of the form $I = [a, b]$ where b is the next floating point value after a. Canonical intervals are the smallest possible interval representable on a machine with limited floating point representation.

The following examples are provided to demonstrate interval arithmetic:

Example 5.5. Let $A = [-3, 2]$ and $B = [2, 4]$ then:

- $A \oplus B = [-1, 6]$; $A \ominus B = [-7, 0]$; $A \otimes B = [-12, 8]$; $A \oslash B = [-1\frac{1}{2}, 1]$
- $A \cup B = [-3, 4]$; $A \cap B = [2, 2]$
- if $f(a, b) = a + 2 * b$ then $F(A, B) = A \oplus 2 \otimes B = [-3, 2] \oplus [2, 2] \otimes [2, 4] = [1, 10]$

The Cartesian product of a set of n intervals is called a *box*, such that $\mathbf{B} = I_1 \times I_2 \times ... \times I_n$. Boxes are denoted using boldface capitals.

5.3.2 Solution-bounding using interval analysis

Interval analysis lends itself to constraint solving because of its ability to compute the bounds over the range of a function. For example, assume that the constraint $f(x) < 0$ must hold given a certain variable x and a finite domain for x. If the interval extension $F(X) = [a_1, a_2]$ (where X is an interval constructed using the lower bound and upper bound of the domain of x) evaluates to an interval for which a_2 (the upper bound) is less than zero, then the constraint is guaranteed to hold for all values in X. If a_1 (lower bound) is greater than zero then the constraint never holds for any value in X. These observations are a direct result of the fundamental theorem of interval analysis. However, because of the exaggerated bounds resulting from natural interval extensions, the constraint is not guaranteed to hold over the entire domain if $0 \in [a_1, a_2]$, nor is it guaranteed to violate the constraint over the entire domain.

Moore (1966) proposes a solution to the problem of exaggerated bounds. The narrower the domain of a function becomes, the more accurately the interval extension approximates the real-valued function. For example, a more accurate approximation of the function $F(X)$ is evaluated by splitting the domain represented by X into a number of smaller sub-intervals and taking the union of each resulting range: $X = [b_1, b_2] \cup [b_2, b_3] \cup ... \cup [b_{n-1}, b_n]$. Specifically, Moore (1966) proves that:

$$\lim_{n \to \infty} \bigcup_{i=1}^{n-1} F([b_i, b_{i+1}]) = \{f(x) | x \in [b_1, b_n]\}$$

If a problem is phrased so that all values of x must be found that satisfy $f(x) < 0$, then interval arithmetic can be used to locate intervals within X that satisfy the constraint. If the original domain does not conclusively satisfy the constraint, then the interval X can be recursively *split* until the interval evaluation of the function over the sub-interval conclusively verifies or violates the inequality. The method of alternating *evaluation* and *splitting* steps forms the basis of an interval-based constraint solving algorithm, which is concerned with locating all solutions to a set of non-linear constraints (including inequalities). The search space is evaluated as being a *valid solution*, *no solution*, or *indeterminate*. In indeterminate cases, the space is split and re-evaluated in a recursive fashion (Jaulin and Walter, 1993, 1996; Jardillier and Languénou, 1998). No further splits occur when the intervals become canonical, and no further splits can be represented on the floating point machine.

We use the following terminology for constraint solving, assuming that a constraint $f(x_1, ..., x_n) < 0$ is defined over n variables: the value of each variable x_i is drawn from a finite interval of real numbers X_i which is the *domain* of that variable. The *domain* of the function f is represented by the box $\mathbf{B} = X_1 \times ... \times X_n$, and this domain is loosely referred to as the *search space* or *configuration space* because it is to be searched for solutions that satisfy the constraint.

5.3.3 Constraint propagation using local consistencies

Constraint solving mechanisms benefit from propagation techniques that remove values from the search space that cannot possibly satisfy the constraints. Interval arithmetic is shown to be beneficial in this regard (Cleary, 1987; Older and Vellino, 1990; Puget, 1994), and relevant techniques for this are described in this section.

We make use of an example to illustrate the concepts used for constraint propagation in interval arithmetic. Consider the following constraint:

$$c : x + y = z$$

This constraint consists of three real valued variables, x, y, and z. Any constraint defines a *relation* between sets of real values, where values from these sets validate the constraint. In the example constraint, the relation describes the sets of real numbers for the variables x, y, and z that cause the constraint to hold (Benhamou and Older, 1997). A relation is expressed using the symbol ρ and is defined as follows:

Definition 5.6. An n-ary relation ρ_c is the set of n-tuples of real numbers that validate a constraint c (Hickey et al., 1998).

A relation differs from a *solution* to a constraint in that a relation describes every possible tuple of real values that validates the constraint. A solution is a sub-set of the underlying relation:

Definition 5.7. A solution to a constraint is a sub-set of n-tuples from the relation of the constraint.

According to Definition 5.6, the relation for constraint c is the set of $3 - tuples$ in $\mathbb{R}^3$ that validates it. Given the initial domain for the constraint represented as a box $\mathbf{B} = X \times Y \times Z$, the intersection of the relation and the box $\rho_c \cap \mathbf{B}$ represents a *solution* to the constraint over the initial domain.

An initial domain is not always a subset of the relation of the constraint. For example, assume that each variable x, y, and z have initial domains $X = [-3, 2]$, $Y = [1, 2]$ and $Z = [0, 100]$. The

Cartesian product of X, Y, and Z form a box $\mathbf{B}$ that encloses all or part of the relation. The box $\mathbf{B} = X \times Y \times Z$ does not exclusively describe the relation of c because if $x = -3$ and $y = 1$ then $x + y = -2$ which is not part of the domain of Z.

It is possible to *narrow* an initial domain so that it more accurately approximates the relation by removing portions of the box that violate the constraint. The exact values for ρ_c are not known, but $\rho_c \cap \mathbf{B}$ is approximated using *projections* of box $\mathbf{B}$ that are defined according to properties of the operators involved in the relation (defined by Hickey et al. (1998)). For example, the addition operator implies three relationships, namely that the addition of the operands equals the sum, and that the difference between the sum and one operand equals the other operand (for both operands). Values violating these properties are removed from the domains using three corresponding projection operators:

$$\begin{aligned}\pi_X(\rho_c \cap B) &= X \cap (Z - Y)\\ \pi_Y(\rho_c \cap B) &= Y \cap (Z - X)\\ \pi_Z(\rho_c \cap B) &= Z \cap (X + Y)\end{aligned}$$

The Cartesian product of π_X, π_Y, and π_Z results in $\mathbf{B}'$, a *narrowed* box that better approximates ρ_c. In the example, Z is reduced to $[0, 4]$ as a result of the π_Z projection as follows:

$$(X + Y) \cap Z \quad = \quad ([-3, 2] + [1, 2]) \cap [0, 100] = [-2, 4] \cap [0, 100] = [0, 4]$$

Further projections over elementary operations including subtraction, division, and multiplication are also defined by Hickey et al. (1998). Projections are not always sufficient to calculate exact values for $\rho_c \cap \mathbf{B}$, especially when a constraint contains more than a single operation. In cases such as these, $\rho_c \cap \mathbf{B}$ can only be approximated using propagation techniques based on *hull* and *box* consistency.

5.3.3.1 Hull consistency

Discarding all real numbers from a box that do not satisfy a constraint is not achievable in the general case, and so a coarse method called *hull consistency* is used to calculate the smallest box containing all $\rho_c \cap \mathbf{B}$ (Benhamou and Older, 1997). If r is a real number, then the function $Hull(r) \mapsto I$ returns the smallest floating-point interval I containing r. More generally, the function $Hull_\square(\mathbf{B}) \mapsto \mathbf{B}'$ returns the smallest floating-point box $\mathbf{B}'$ containing box $\mathbf{B}$ (Benhamou et al., 1994; Benhamou, 1995).

Given a constraint c and a domain expressed as a box $\mathbf{B}$, the term *hull consistency* is used to indicate that $\mathbf{B}$ represents the smallest box containing ρ_c. A real constraint c is *hull consistent* with respect to a box $\mathbf{B}$ if and only if $\mathbf{B} = Hull_\square(\rho_c \cap \mathbf{B})$ (Benhamou et al., 1999).

Hull consistency is ensured if the smallest box is found that contains all tuples that form the solution from the initial domain. The relation ρ_c is potentially comprised of multiple disjoint boxes, and in this case the hull of this relation contains tuples that are not part of the solution.

Hull consistency is achieved using a *hull consistency operator*, an algorithm that makes use of projections to determine the hull of $\rho_c \cap \mathbf{B}$. Benhamou et al. (1999) present an algorithm called $HC4$, which achieves hull consistency for non-primitive constraints. $HC4$ is a method for narrowing an initial box according to a set of arbitrary constraints, or detecting when a box does not contain any portion of the relation. The algorithm formulates an evaluation tree for each constraint, where each node in the tree describes a primitive operation. Projection operators are invoked at each node in the tree, resulting in a narrowed domain. Domains are narrowed repeatedly until no further narrowing occurs, a process called *chaotic iteration* (Apt, 1999). Algorithm $HC4$ is explained further in Appendix C, and specific detail and proofs regarding this algorithm are provided by Benhamou et al. (1999).

Hull consistency does not guarantee that the returned box contains only the solutions. Part of the cause of this is that the projection operators are not always able to narrow the domains of variables any further without the risk of losing valid solutions. Related research also acknowledges that hull consistency is limited in its ability to handle constraints in which a single variable occurs multiple times (Benhamou et al., 1994). As a result, a tighter form of narrowing is used based on the idea of *box consistency.*

5.3.3.2 Box consistency

The primary problem with hull consistency over complex constraints (constraints containing multiple instances of the same variable (Benhamou et al., 1994)) is the decomposition of the constraint expression into elementary operations (at each step in the evaluation tree). This introduces dependency problems between each instance of the same variable (Benhamou et al., 1999). To overcome this, *box consistency* avoids decomposition, and hence is able to produce tighter narrowing of an initial box.

The conditions for box consistency are more complex than for hull consistency. Let a constraint c contain k variables, where each variable v_i is defined over a domain D_i from the box $\mathbf{D}$. The constraint is rewritten as a set of k univariate constraints C_i where $1 \leq i \leq k$. Every variable in C_i is replaced with its corresponding domain, except for the variable v_i. $\mathbf{D}$ is box consistent if the following relation holds (for all $1 \leq i \leq k$):

$$D_i = D_i \cap \{v_i \in \mathbb{R} | C_i(D_1, ..., D_{i-1}, v_i, D_{i+1}, ..., D_i)\}$$

Simply stated, a box $\mathbf{D}$ is box consistent if every $D_i \in \mathbf{D}$ represents the hull of the solution to the i^{th} univariate constraint.

Intervals for D_i that exhibit box consistency are determined using an iterative algorithm. Given a constraint C_i, we replace each variable with its domain in the input box, except for one variable. This transforms the constraint into a univariate interval function. The bounds on the solutions in D_i to the univariate function are then located. Finding the bounds of D_i is achieved by using root finding methods (typically the Interval Newton Method) over the univariate function. More specifically, the left most and right most root is located.

Benhamou et al. (1994) define an algorithm called $BC4$ that achieves box consistency over a set of constraints, and that is capable of narrowing the initial domain more tightly than hull consistency. $BC4$ is also able to detect when a domain contains no solutions. $BC4$ uses an algorithm called $BC3Revise$ that achieves box consistency for a single constraint at a time. These algorithms are described in detail by Benhamou et al. (1999), as well as in Appendix C.

Both hull and box consistency operators are effective for removing portions of the search space that do not form part of a solution to a set of constraints. These narrowing operators are termed *outer contracting operators* because they contract the domain as much as possible without removing any portion of the solution space (Benhamou et al., 2004). However, neither of these techniques guarantee that a narrowed box contains *only* solutions, and so outer contracting operators alone are not sufficient for locating the solution to a set of constraints. Boxes that exclusively contain solutions are *sound*, and further methods are required to locate sound solutions to a system of constraints.

5.3.4 Sound constraint solving

Sound constraint solving techniques produce boxes that contain only solutions. We are interested in constraint systems defined over universally quantified variables, and the definition of a solution is extended to encompass this concept. Constraints with universally quantified variables are required to hold over the entire contiguous intervals defined by these variables. This means the initial domain of a universally quantified variable must be identical to its corresponding domain in the solution box:

Definition 5.8. A solution to a constraint containing a universally quantified variable u contains the entire initial domain of u.

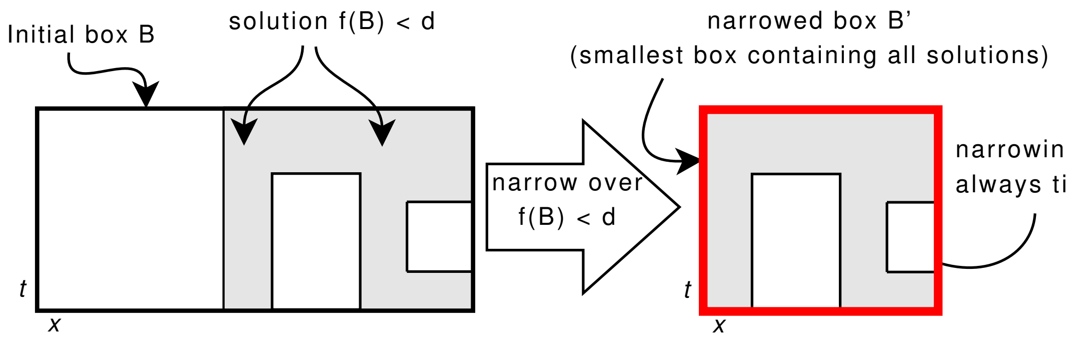
Outer contracting operators implicitly identify boxes that are not solutions to universally quantified constraints, because any narrowing that occurs over the domain of the universally quantified variable violates the universal requirement. If this occurs, then the box is guaranteed not be a universally quantified solution.

An *inner contracting operator* produces boxes that fall exclusively within the solution space (Benhamou and Goualard, 2000; Benhamou et al., 2004) and discards all values from the initial box that do not form part of the solution, as well as values that form part of the solution but cannot be enclosed in a computer representable box.

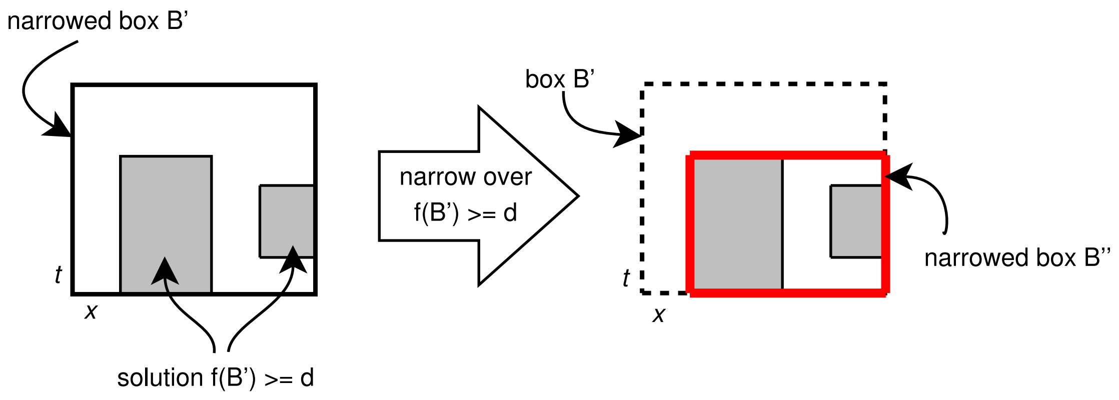
An inner contracting operator narrows a domain using the original set of constraints, and then narrows the domain over the set of *negated* constraints (that is, where relational symbols are reversed, for example from $>$ to $\leq$). Narrowing over the negated constraints removes portions from the domain that are guaranteed to be non-solutions to the negated constraint. These removed portions are then guaranteed to be solutions to the original constraints by implication. This process is illustrated in Figure 5.5 for a constraint that includes a universally quantified variable.

In Figure 5.5(a), the initial box $\mathbf{B}$ is narrowed using the outer contracting operator over a constraint and produces box $\mathbf{B}'$ eliminating some, but not all, invalid ranges of values from the initial domain. Narrowing operators never discard solution space, and if the universally quantified variables are narrowed in this step then it means that no solutions exist in $\mathbf{B}$ and this box is discarded.

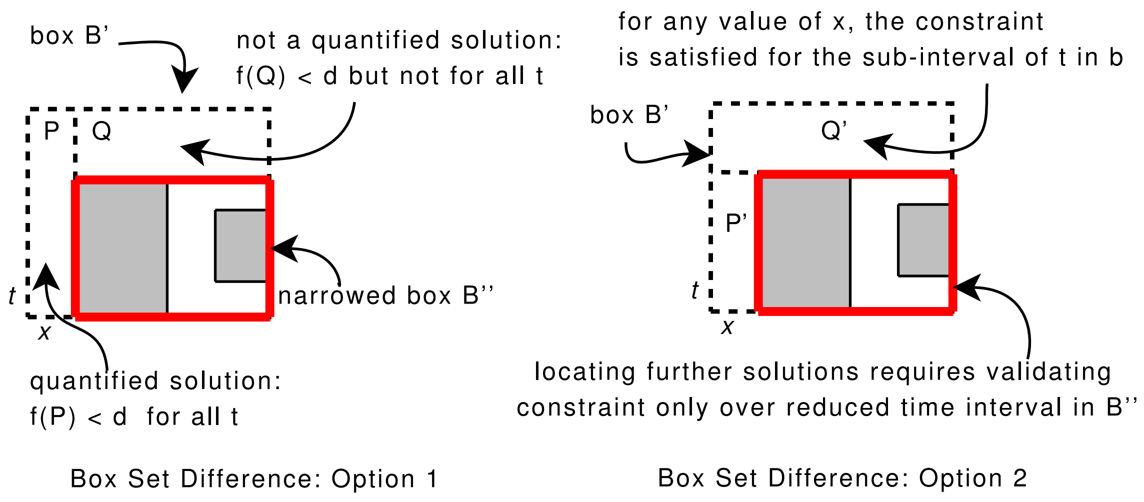
$\mathbf{B}'$ is then narrowed using the corresponding *negated* constraint, which is derived by inverting the relation operator from $>$ to $\leq$. The result of this narrowing is box $\mathbf{B}''$, as illustrated in Figure 5.5(b). This narrowed box contains all solutions and possibly non-solutions to the *negated* constraint. However, the difference between the original box $\mathbf{B}'$ and this narrowed box $\mathbf{B}''$ is guaranteed to be a solution to the original constraint, because it is guaranteed to be a non-solution



(a) Step 1: Narrow over constraint.

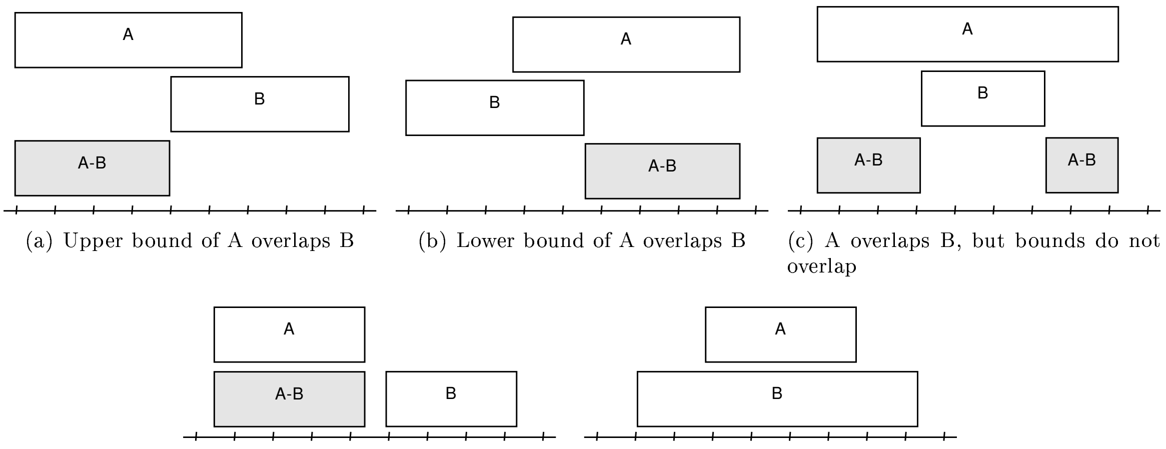


(b) Step 2: Narrow over negated constraint.


box B'
not a quantified solution:
f(Q) < d but not for all t
P
Q
narrowed box B''
t
x
quantified solution:
f(P) < d for all t
Box Set Difference: Option 1
for any value of x, the constraint
is satisfied for the sub-interval of t in b
box B'
Q'
P'
t
x
locating further solutions requires validating
constraint only over reduced time interval in B''
Box Set Difference: Option 2

(c) Step 3: Box set difference yields a solution and smaller universally quantified interval.

Figure 5.5: Graphical illustration of solution finding approach for constraints using universally quantified variables.



(a) Upper bound of A overlaps B (b) Lower bound of A overlaps B (c) A overlaps B, but bounds do not overlap

(d) A does not overlap B (e) A completely subsumed by B

Figure 5.6: Illustration of set difference between two intervals.

to the negated constraint. Using this reasoning, solutions are found without the need to evaluate the functions and risk exaggerated range bounds. In addition, a solution is identified without sampling over the universally quantified variable t.

The solution of the negated constraint is the inverted solution of the original constraint, and any box-set difference between $\mathbf{B}'$ and $\mathbf{B}''$ is guaranteed to be a solution to the original constraint. Set difference is different to the subtraction of intervals, and is illustrated graphically in Figure 5.6. In each example B is subtracted from A, and the set difference is undefined if B completely subsumes A (Figure 5.6(e)). If A completely subsumes B then the set-difference results in two intervals (Figure 5.6(c)). Box-set difference is the set difference of each domain within the two operand boxes, and potentially returns more than a single box as a result.

Figure 5.5(c) indicates two options for calculating the box set difference. The first option yields a solution $\mathbf{P}$ that spans all t and is a universally quantified solution ($\mathbf{Q}$ is not a universally quantified solution in this respect). If further solutions are required, then $\mathbf{B}''$ is searched further for solutions. This has the effect of narrowing the universally quantified domain, because the constraint is guaranteed to hold for any x over the entire sub-interval of t in $\mathbf{Q}'$.

If a solution is not encountered, then box $\mathbf{B}''$ is split along any domain except that of the universally quantified domain, and the process repeated on each sub-box. The process of contracting and splitting is repeated until boxes become canonical, or solutions are found.

The above technique is capable of locating solutions for universally quantified variables without requiring an evaluation step, and also enables the reduction in size of the universally quantified domain. Benhamou et al. (2004) formalize this technique as the $ICO2$ algorithm that acts as an inner contracting operator over a single constraint. Solving for a set of constraints is achieved by finding all solutions for each constraint in turn, and using the solutions for each constraint as initial boxes for the next constraint to be considered. The exact algorithm for a system of constraints is formally described by Benhamou et al. (2004) as the IPA algorithm. IPA is shown to be *sound,* that is, boxes returned by the algorithm contain only solutions.

Figure 5.7 illustrates the various components of a sound constraint solver. The IPA algorithm is used to locate sound solutions to a system of constraints. It uses the $ICO2$ algorithm to locate

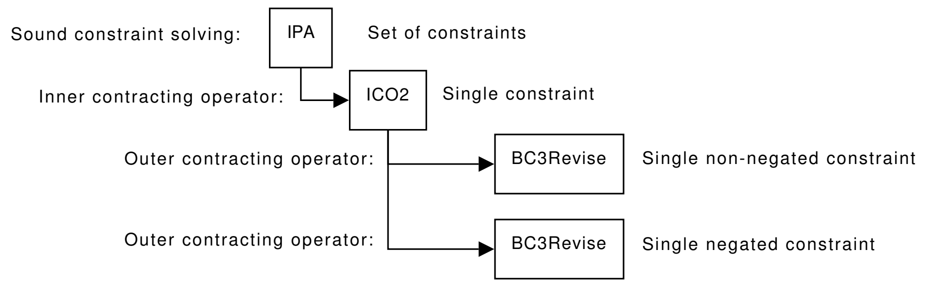

Sound constraint solving:
IPA
Set of constraints
Inner contracting operator:
ICO2
Single constraint
Outer contracting operator:
BC3Revise
Single non-negated constraint
Outer contracting operator:
BC3Revise
Single negated constraint

Figure 5.7: Illustration of the composition of a sound interval constraint solver by Benhamou et al. (2004).

sound solutions for each constraint in turn. The $ICO2$ algorithm uses a narrowing operator, namely $BC3Revise$ to narrow the domain over the constraint, as well as over the negated constraint.

5.3.5 Alternative formulations of interval constraint solving

Some interval extensions of primitive operations are more rigorously defined to provide interval evaluations with tighter bounds. For instance, Hickey et al. (2001) indicate that the division operation need not be undefined if the denominator interval contains zero, but rather produces two disjoint intervals. In the techniques described until this point, one interval is used to subsume disjoint intervals for the reason that preserving disjoint intervals is too computationally expensive (Benhamou et al., 1994). Alternative methods exist that maintain disjoint intervals however (Chabert et al., 2005), and in some instances this is shown to improve splitting strategies (Batnini et al., 2005).

Other improvements to the constraint solving technique include detecting and using cycle information between constraints to optimize chaotic iteration (Lhomme et al., 1998), removing constraints from the system as they are satisfied (Borning et al., 1996) and performing interval narrowing on a parallel architecture (Granvilliers and Hains, 2000). Ratschan (2006) proposes a more generalized quantified constraint solver that splits the universally quantified domain (unlike Benhamou et al. (2004) who never split these domains), and ensures that a solution exists for all sub-domains of the universally quantified domain. We follow the work by Benhamou et al. (2004) in this chapter.

5.4 Interval-based quantified constraint optimization

This section describes our innovative approach to locating solutions to quantified constraint systems. Interval-based constraint *solvers* (such as the one described in Section 5.3.4) fail when faced with constraint systems that are inconsistent, or are inefficient for systems in which solutions consist of small disjoint portions of the search space:

- Constraint solvers are interested in finding only global solutions. If a constraint system is inconsistent, then an extensive search is performed without any immediate results. The result is strictly binary: a solution, or no solution.

- The solver presented in Section 5.3.4 is a split and search algorithm in the worst case. Unless tailored heuristics are employed (for example alternative space traversal strategies (Benhamou et al., 2004) or "coordinate search" (Huyer and Neumaier, 1999)) there is no correlation between the amount of searching performed and the proximity to a solution.

We describe an optimization strategy that addresses both these problems, while maintaining the ability to locate universally quantified solutions. We use the term *optimizer* to refer to the proposed algorithm, and the term *solution* to refer to a box that satisfies a set of constraints. We use the term *minimizer* when referring to a box that approximates a solution.

The optimization strategy described in this section uses the quantified constraint propagation and sound constraint solving techniques presented in Section 5.3.4. This strategy is able to provide a minimizer at any point (even if the minimizer is not a proper solution to the constraint system). The longer the optimization continues, the more the minimizer resembles the actual solution of the constraint system. If the constraint system is inconsistent, then the minimizer approximates a solution even though one does not exist.

The following sections describe the concept of relaxed constraints and how these are used to achieve optimization.

5.4.1 Optimization using relaxed constraints

Our technique for optimization is based on the following conjecture:

Conjecture 5.9. *Assume a constraint c with underlying relation ρ_c can be "relaxed" in some manner, so that the underlying relation of the relaxed constraint ρ_c^δ completely contains ρ_c, that is $\rho_c \subseteq \rho_c^\delta$. We speculate that approximating ρ_c^δ is a "simpler" task than approximating ρ_c in the case where ρ_c exists, and that ρ_c^δ is a "minimizer" for the constraint in the case where the constraint is inconsistent.*

Figure 5.8 provides an illustration of Conjecture 5.9, in the case where a solution exists. Assume some constraint $F(\mathbf{B}) < 0$ is applied over a domain represented by box $\mathbf{B}$. The constraint is relaxed by rephrasing the constraint as $F(\mathbf{B}) < \delta$ given a large enough δ. Locating a solution for this relaxed constraint does not require any splitting or narrowing of $\mathbf{B}$ given a large enough δ . For a reduced value of δ, the task of locating solutions is more difficult, but still requires less splitting than the original constraint.

For this exposition we discuss constraints that are phrased in the manner $c : f(x_1, ..., x_n) \leq 0$, but all the described methods are applicable to other inequality relations. We relax a constraint by replacing the zero on the right hand side with a value δ (called the *relaxation constant*) that is greater than zero, in the following manner:

$$c^\delta : f(x_1, ..., x_n) \leq \delta$$

We assume that a large enough value of δ exists that causes the constraint to hold over the initial domains of $x_1, ..., x_n$.

Example 5.10. Let c be a constraint defined as follows: $c : 2x + y \leq 0$ where $x \in [-2, 4]$ and $y \in [-1, 2]$. Evaluating the interval extension of f over the original domain results in $F(X, Y) =$

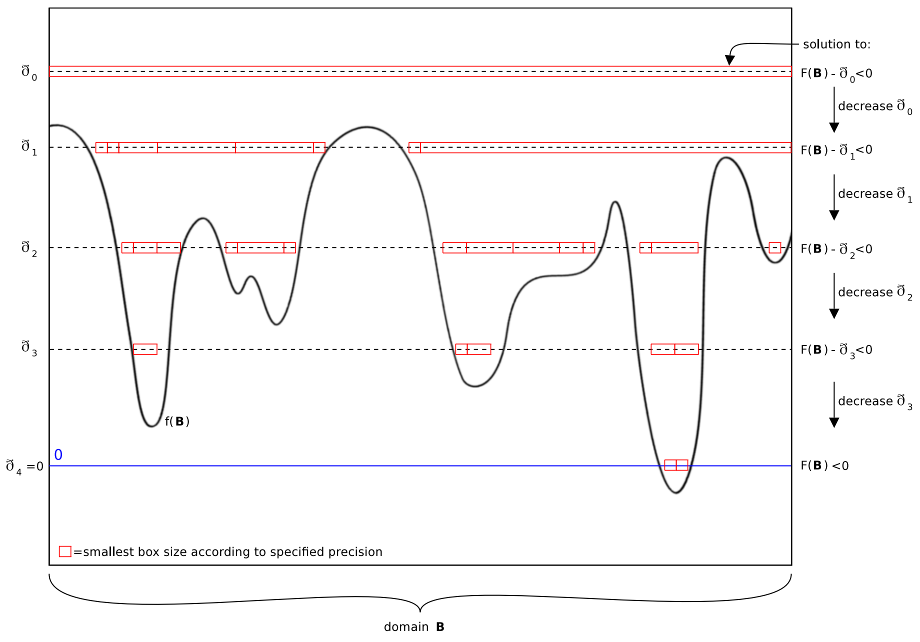


Figure 5.8: Illustration of interval optimization process.

$[2, 2] * [-2, 4] + [-1, 2] = [-5, 10]$. This interval does not verify the constraint (not all numbers are less than or equal to zero), and so a relaxation constant $\delta = 10$ is chosen so that $F(X, Y) = [-5, 10] \leq [10, 10]$.

Example 5.10 illustrates that an initial value for δ is found using the interval evaluation of the function over the initial variable domains. This method is used to determine the relaxation constant, and the initial domain is a solution to the relaxed constraint.

The value of δ is reduced and the newly *tightened* constraint is solved once again, resulting in a solution that is a sub-set of the initial domain. The process of reducing δ and solving the tightened constraint is repeated until δ reaches zero, whereby the resulting solutions are solutions to the original constraint. If no solutions are found at a specific value of δ, then no solutions exist for the constraint and the solutions for the previous value of δ are *minimizers*.

This method for constraint optimization produces a focused traversal of the search space, because portions of the space that are not solutions to a relaxed constraint are removed from the search-space for a non-relaxed constraint. These portions of space are removed for each value of δ, and as δ approaches zero the remaining portions of space become closer approximations of the solution space.

Interval-based constraint solving methods return solutions as boxes, which means that a solution describes a range of values that satisfy the constraint. Solutions to relaxed constraints provide box approximations to the actual solutions, and the value of δ indicates the upper bound of deviation from an actual solution with respect to any range of values comprising an approximate solution. In this respect, a box returned as an approximate solution potentially contains actual solutions, where the likelihood increases as δ approaches zero.

The following section describes the method used for solving relaxed constraints at any level of δ. These constraints potentially contain universally quantified variables.

5.4.2 Constraint solving over a set of relaxed constraints

The method employed for constraint solving at each value of δ is similar to the approach developed by Benhamou et al. (2004), and is based on the use of an outer and inner contracting operator. The primary difference is that the contracting operators are implemented over a set of constraints rather than a single constraint at a time.

An outer contracting operator *outerContract* takes as input a system of constraints C and a box $\mathbf{B}$ representing the domain of the variables in the system of constraints. Three results are possible, namely an unchanged box, a reduced box, or failure. These are interpreted as follows:

$$\begin{aligned} \mathit{outerContract}(C,\mathbf{B}) \subseteq \mathbf{B} &\quad\Rightarrow\quad \text{existence of a solution in } \mathbf{B} \text{ is indeterminate} \\ \mathit{outerContract}(C,\mathbf{B}) = FAIL &\quad\Rightarrow\quad \text{no solution in } \mathbf{B} \end{aligned}$$

The $BC3$ algorithm detailed by Benhamou et al. (1994) is used for the *outerContract* operator. If the algorithm returns $FAIL$, this means that no solution exists in the input domain (for instance, if any universally quantified variable is shrunk then $FAIL$ is returned). The primary function of the outer contracting operator in the optimization process is to narrow the search space using constraint propagation and to detect if the search space contains no solutions.

An inner contracting operator *innerContract* takes as input a system of constraints C and a box $\mathbf{B}$ representing the domain of the variables in the system of constraints. The result is a tuple $(S = \{\mathbf{S_1}, ..., \mathbf{S_n}\}, \mathbf{L})$ containing a set of solution boxes, and a box $\mathbf{L}$ representing a narrowed $\mathbf{B}$. These are interpreted as follows:

$$\begin{aligned} \mathbf{S_i} \subseteq \mathbf{B} &\quad\Rightarrow\quad \text{solutions exist in box } \mathbf{B}\text{, and } \mathbf{S_i} \text{ is a solution} \\ S = \{\} &\quad\Rightarrow\quad \text{existence of solution in } \mathbf{B} \text{ indeterminate,} \\ &\qquad\quad\; \text{but if it does exist, it is in the reduced box } \mathbf{L} \end{aligned}$$

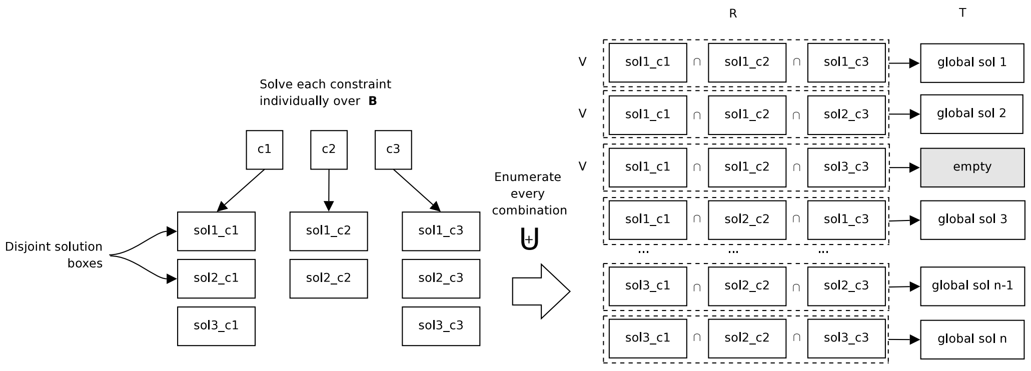
The primary function of the inner contracting operator is to detect whether an entire box is a subset of the solution space, that is, detect *sound* solutions. Leftover space $\mathbf{L}$ can neither be guaranteed to contain solutions, nor guaranteed not to contain solutions.

We design an inner contracting operator that has these properties in the manner described by Algorithm 5.1. This algorithm is similar to the $ICO2$ algorithm in that it uses the idea of negated constraints and box-set difference to locate solutions (Benhamou et al., 2004). Algorithm 5.1 is different from $ICO2$ because it applies to a set of constraints rather than a single constraint at a time. The algorithm takes as input a set of constraints, and a box $\mathbf{B}$ that is previously narrowed using the *outerContract* operator. Each constraint is negated and narrowed individually over box $\mathbf{B}$ and if a $FAIL$ is returned then $\mathbf{B}$ is the solution space to the constraint by implication. Otherwise, the box set difference between $\mathbf{B}$ and the narrowed box is a solution to the constraint.

Each constraint has the potential to produce a number of disjoint solution boxes. To find a global solution to the set of constraints, an intersection is performed for every combination of solution boxes produced by each constraint. This is illustrated in Figure 5.9, in which constraint c_1 results in three disjoint solutions, constraint c_2 results in two disjoint solutions, and constraint c_3 results in three disjoint solutions. The $\uplus$ operator is used to enumerate every possible combination of disjoint solution boxes from each constraint. The intersection of each combination is a global solution to the system of constraints.

Algorithm 5.1 Inner contracting operator over a set of constraints.

```
$innerContract$(in: constraint set $C$,
                    box $\mathbf{B}$ previously narrowed by $outerContract$;
                out: tuple $(T, \mathbf{L})$ where:
                         $T$ is set of solution boxes
                         $\mathbf{L}$ is the narrowed left-over box)
begin
  $R \leftarrow \{\}$      %% Every combination of solutions
  $\mathbf{L} \leftarrow \emptyset$      %% Initialize left-over space as empty-set
  for each $c \in C$ do
    $S \leftarrow \{\}$                  %% Solutions for this constraint
    $\mathbf{B}' \leftarrow outerContract(\{\bar{c}\}, \mathbf{B})$ %% Narrow $\mathbf{B}$ over single negated constraint
    if $\mathbf{B}' = FAIL$ then
      add $\mathbf{B}$ to $S$ as a solution to this constraint
    else
      $Q \leftarrow \mathbf{B} \boxminus \mathbf{B}'$          %% Box set difference
      for each box $\mathbf{Q} \in Q$ do
         add $\mathbf{Q}$ to $S$ if universally quantified domain not narrowed
      if $\mathbf{B}' \neq \mathbf{B}$ then
        $\mathbf{L} \leftarrow \mathbf{L} \cup \mathbf{B}'$   %% Get union of left-over box
    $R \leftarrow R \uplus S$   %% Enumerate all combinations of solutions
  $T \leftarrow \{\}$
  %% Each set in $R$ contains one solution box for every constraint
  for each set $V$ in $R$ do
    if intersection between all solutions in $V$ is defined then
      add intersection to $T$
  return $(T, \mathbf{L})$
end
```



R
T
Solve each constraint individually over B
c1
c2
c3
Disjoint solution boxes
sol1_c1
sol2_c1
sol3_c1
sol1_c2
sol2_c2
sol1_c3
sol2_c3
sol3_c3
Enumerate every combination
⊎
V
sol1_c1 ∩ sol1_c2 ∩ sol1_c3
global sol 1
sol1_c1 ∩ sol1_c2 ∩ sol2_c3
global sol 2
sol1_c1 ∩ sol1_c2 ∩ sol3_c3
empty
sol1_c1 ∩ sol2_c2 ∩ sol1_c3
global sol 3
sol3_c1 ∩ sol2_c2 ∩ sol2_c3
global sol n-1
sol3_c1 ∩ sol2_c2 ∩ sol3_c3
global sol n

Figure 5.9: Illustration of the combination of solutions performed by the inner contracting operator (defined in Algorithm 5.1).

The two inner and outer contracting operators function together as a constraint solving mechanism. The *outerContract* operator is used to remove portions of the domain that contain no solutions, and *innerContract* is used to identify sound solutions within domains previously narrowed by *outerContract*. The manner in which these operators are invoked in an optimization process is described in the following section.

5.4.3 Optimization through iterative tightening

We present an optimization process that repeatedly solves systems of progressively tightened constraints. The less relaxed a system of constraints, the more its set of solutions approximates the actual solutions to the original set of constraints.

A relaxed constraint set is created by relaxing each individual constraint within it. This is done by evaluating each individual constraint c_i over the initial domain, and selecting a δ_i that relaxes the constraint sufficiently so that it is validated over the initial domain. The sum of δ_i values represents the overall relaxation constant for the constraint set. A relaxed constraint set C with total relaxation δ is denoted as C^{δ}.

The constraint optimization algorithm over universally quantified constraints is presented as Algorithm 5.2, which consists of a pair of nested loops. The inner loop populates a set T with solutions of the constraint system (using outer and inner contracting operators) for a specific relaxation constant. Boxes for which no solutions are explicitly located are split using the *split* function (never splitting universally quantified variables). These split boxes are added to the set D for further searching and splitting. The inner loop exits when no further boxes exist in D, a state that occurs when boxes cannot be split any further on a finite floating point representation machine.

After the inner loop exits, the set of solutions at the current level of δ represent current minimizers to the constraint system. The constraint system is tightened by reducing δ_i for each constraint, and the outer loop iterates using the set of solutions at the previous step as initial domains to be searched. The outer loop exits under three conditions: if there are no domains for the next iteration, in which case no solution exists for the constraint system; if δ is sufficiently close to zero, in which case the current solution is a solution to the system; and if the execution time of the algorithm exceeds a manually specified threshold.

The problem with Algorithm 5.2 is its slowness to converge to a minimum or solution. Boxes that are indeterminate are split repeatedly at each level of δ until machine precision is reached, which means that the number of boxes in D grows very quickly. This results in a large portion of execution time being spent in the inner loop of the algorithm. The next section discusses strategies for mitigating this problem.

5.4.4 Reduction of execution time

We modify Algorithm 5.2 to reduce the amount of time spent searching for solutions at each level of δ using a pair of thresholds that are used to exit the inner loop prematurely.

The first threshold for exiting the inner loop of Algorithm 5.2 defines a maximum number of solution boxes $\tau_{solutions}$ to be found at each value of δ. If the number of solution boxes in T exceeds this threshold, then the inner loop exits. The other threshold limits the number of split

Algorithm 5.2 Quantified interval optimizer.

```
IOPT(in: constraint set $C$, box $\mathbf{B}$,
         precision level for recognizing solutions $\varepsilon$,
         time cutoff threshold $\tau_{time}$;
     out: set $S$ of solution or minimizer boxes)
begin
  $C^\delta \leftarrow$ constraint set relaxed over initial domain
  $D \leftarrow \{\mathbf{B}\}$   %% Set of solutions at previous $\delta$
  $M \leftarrow \{\}$    %% Minimizers found so far
  while $size(D) > 0$ and $\delta > \varepsilon$ and $time < \tau_{time}$ do
      $T = \{\}$  %% Solutions at this level of $\delta$
      while $size(D) > 0$ and $time < \tau_{time}$ do
          $\mathbf{B} \leftarrow removeFirst(D)$
          $\mathbf{B}' \leftarrow outerContract(C^\delta, \mathbf{B})$
          if $\mathbf{B}' \neq FAIL$ then
            $(Q, \mathbf{L}) \leftarrow innerContract(C^\delta, \mathbf{B}')$
            if $size(Q) > 0$ then
              add all solutions in $Q$ to $T$
            $D \leftarrow D \cup split(\mathbf{B}')$ %% Split for future searching
            $D \leftarrow D \cup split(\mathbf{L})$  %% Split for future searching
      if $size(T) > 0$ then  %%There are solutions at this $\delta$
        $D \leftarrow T$   %% Use these solutions as input for reduced $\delta$ value
        $M \leftarrow T$   %% Save these solutions as minimizers
        $C^\delta \leftarrow$ tighten constraint set $C^\delta$(reduce $\delta$)
  if $\delta > \varepsilon$ then   %% Outer loop exits without reaching $\delta = 0$
     return $M$   %% return minimizers
  else
     return $D$   %% return solutions
end
```

boxes created within the inner loop. If the number of boxes in D exceeds the threshold τ_{splits}, then the inner loop exits. The use of these thresholds limits the number of iterations performed by the inner loop. However, the early termination of the inner loop introduces the problem that potential solutions in D are never recognized.

We use backtracking to avoid the problem introduced by thresholds. When a threshold is reached, all boxes in D that have not yet been searched are saved, along with the current relaxed constraint system. If any subsequent searches fail to locate solutions for a tightened constraint system, the algorithm relaxes the constraints to their previous level, and continues searching the remaining boxes.

We represent D using a queue structure. At each iteration of the inner loop the first box from D is removed and searched, and if no solutions are found the box is split. Newly split boxes are appended to the end of D. This process is analogous to a breadth-first traversal of a tree. To reduce the problem introduced by thresholding, we only enable thresholds if each depth in the implicit search tree is searched to a sufficient degree. In practice, we find that searching 200 boxes at each depth in the tree (before thresholds are enabled) provides adequate performance gains. These boxes are chosen in a distributed manner so that they are sampled from across the entire range at a particular depth.

The enhanced interval optimizer is presented as Algorithm 5.3, in which the enhancements from Algorithm 5.2 are highlighted. Algorithm 5.3 makes use of a variable called δ_{best}, which records the lowest level of δ for which solutions are found. This is required in the event that backtracking occurs to distinguish future minimizers from the best solution located until that point. A set called $States$ is also maintained for backtracking purposes, containing tuples of the form $(C^{\delta} = \{c^{\delta_1}, ..., c^{\delta_n}\}, D)$, each of which associates a set of relaxed constraints with a set of boxes that still require searching.

The outer loop of Algorithm 5.3 remains unchanged, but the inner loop is modified in a number of ways. Instead of removing the first box from the set D for searching, a box is removed at a specific index to ensure that boxes are sampled from across the current depth of the tree. Newly split boxes are not immediately added to D for future processing, but are rather added to an intermediate set N representing the next depth in the tree to be searched. Only in the event that D becomes empty are the elements in N appended to D. The inner loop only exits if the current depth has been adequately sampled according to the manually specified $factor$ value (number of boxes searched at each level) and one of the two thresholds are reached. The inner loop also exits if a time limit is reached, or no remaining boxes are left for searching at the current value of δ.

The exit of the inner loop results in either a set of solutions for the current value of δ (the set T), or no solutions ($T = \{\}$). If solutions are found and the current δ is less than δ_{best} then these solutions are minimizers, and the constraint set can be tightened and solved over these solutions. If no solution is found ($T = \{\}$), then a backtrack must occur, relaxing the constraint set to a previous level of δ that still has unsearched boxes. The decision as to which constraint set and set of boxes to search at the next iteration of the outer loop is encapsulated in the $switchState$ function, which either tightens the constraint set, or backtracks to a relaxed constraint set:

$$switchState(C^{\delta}, T, States) = \begin{cases} (\text{tightened } C^{\delta}, T) & \text{if}\, size(T) > 0 \\ removeFirst(States) & \text{if}\, size(T) = 0 \end{cases}$$

Algorithm 5.3 Interval optimizer updated for efficiency.

```
    $IOPT_2$(in: constraint set $C$, box $\mathbf{B}$,
                precision level for recognizing solutions $\varepsilon$,
                time cutoff threshold $\tau_{time}$,
                solution cutoff threshold $\tau_{solutions}$,
                split cutoff threshold $\tau_{splits}$
                number of boxes before thresholds enabled $factor$;
            out: set $S$ of solution or minimizer boxes)
    begin
      $C^\delta \leftarrow$ constraint set relaxed over initial domain
      $D \leftarrow \{\mathbf{B}\}$   %% Set of solutions at previous $\delta$
      $M \leftarrow \{\}$   %% Minimizers found so far
  ▶   $\delta_{best} \leftarrow \delta$   %% Lowest $\delta$ found with solutions
  ▶   $States \leftarrow \{\}$ %% Backtrack states
      while $size(D) > 0$ and $\delta > \varepsilon$ and $time < \tau_{time}$ do
          $T = \{\}$  %% Solutions at this level of $\delta$
      ▶   $distribution \leftarrow size(D)/factor$  %% Calculate how to select search boxes
      ▶   $removeIndex \leftarrow 0$              %% Index for removing search boxes
      ▶   $N = \{\}$                            %% Next level of split boxes
          while $size(D) > 0$ and $time < \tau_{time}$ and
      ▶           not($removeIndex > size(D)$ and
      ▶           ($size(N) > \tau_{splits}$ or $size(T) > \tau_{solutions}$)) do
              ▶ $\mathbf{B} \leftarrow$ remove box at index $removeIndex$ from $D$
              ▶ $removeIndex \leftarrow removeIndex + distribution$
                $\mathbf{B}' \leftarrow outerContract(C^\delta, \mathbf{B})$
                if $\mathbf{B}' \neq FAIL$ then
                  $(Q, \mathbf{L}) \leftarrow innerContract(C^\delta, \mathbf{B}')$
                  if $size(Q) > 0$ then
                     add all solutions in $Q$ to $T$
              ▶   $N \leftarrow N \cup split(\mathbf{B}')$ %% Split for future searching
              ▶   $N \leftarrow N \cup split(\mathbf{L})$  %% Split for future searching
              ▶   if $size(D) = 0$ then
              ▶      $D \leftarrow N$ %% If level is complete, search next level
          add all boxes in $N$ to $D$
      ▶   if $size(T) > 0$ and $\delta < \delta_{best}$ then
      ▶      $M \leftarrow T$
      ▶      $\delta_{best} \leftarrow \delta$
      ▶   if $size(D) > 0$ then add $(C^\delta, D)$ to $States$   %% Save state
      ▶   $(C^\delta, D) \leftarrow switchState(C^\delta, T, States)$       %% Reduce $\delta$ or backtrack
      if $\delta > \varepsilon$ then   %% Outer loop exits without reaching $\delta = 0$
         return $M$   %% return minimizers
      else
         return $D$   %% return solutions
    end
```

Parameter	Description
ε	Level of precision for relaxation constant δ
$factor$	Number of boxes to be searched at each depth before thresholds are enabled
τ_{time}	Threshold for execution time
$\tau_{solution}$	Threshold for solutions found in inner loop
τ_{splits}	Threshold for split boxes created in inner loop

Table 5.1: Summary of parameters for the interval-based quantified constraint optimization algorithm.

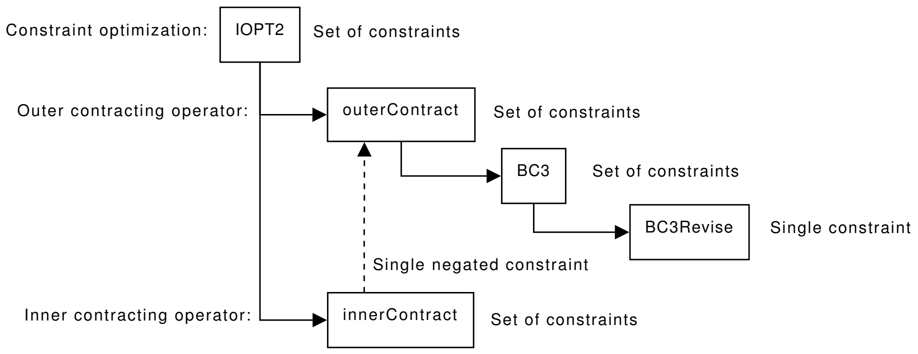


Figure 5.10: Illustration of the components of the interval optimization algorithm.

The outer loop iterates until δ reaches a sufficient proximity to zero (defined by the constant ε), or until no further backtracking states exist. The outer loop also exits if the time threshold is reached, in which case the minimizer at the current δ_{best} value is returned as an approximate solution.

The set of parametrization values for the $IOPT_2$ algorithm is summarized in Table 5.1.

5.4.5 Implementation

We implement the $IOPT_2$ algorithm using components sourced from existing research. Figure 5.10 illustrates that the primary two components of $IOPT_2$ algorithm are the *outerContract* and *innerContract* operators. The *outerContract* operator is implemented using the $BC3$ algorithm defined by Benhamou et al. (1994). This algorithm in turn incorporates the $BC3Revise$ (Benhamou et al., 1994, 1999) component for achieving box consistency. The *innerContract* algorithm (Algorithm 5.1 on page 127) is based on the $ICO2$ algorithm defined by Benhamou et al. (2004), in which a single negated constraint is narrowed at a time using the *outerContract* operator.

We implement interval arithmetic operations in Java, using natural interval extensions of primitive operators described by Moore (1966) and Hickey et al. (2001). Conservative outward rounding is provided in the implementation using the `BigDecimal` class available in the Java 1.5 API, which performs rounding at manually specified levels of precision.

Our implementation of box consistency is different to the method used by Benhamou et al. (1994). We use a divide and conquer approach for finding the right-most and left-most roots of the univariate functions (instead of the Newton method). The Newton method is faster in locating roots, but it requires the calculation of the derivative of a function, as well as an implementation of the interval division operator, both of which are non-trivial. For example, the handling of division by zero in interval arithmetic is a topic with disputed solutions (Moore, 1966; Hickey et al., 1998). The divide and conquer approach avoids this issue, at the cost of slower convergence however.

The $IOPT_2$ algorithm performs an initial evaluation of the set of constraint functions to determine the starting value of δ. Evaluation of an interval function $F(\mathbf{B})$ is performed using an interval evaluation technique based on *centered forms* for producing less exaggerated bounds, described by Moore (1966).

Input constraints are expressed symbolically (examples of which appear in Appendix D) and are parsed using the Java Expression Parser[2]. An initial, non-infinite box **B** is specified that represents the search domain for a system of constraints.

5.5 Analysis of the interval-based quantified constraint optimizer

We examine the properties of the interval-based quantified constraint optimizer to determine if it is effective in quantifying behaviour specified by systems of constraints. These properties are investigated in terms of the following questions:

1. *What parametrization values are appropriate for the interval-based quantified constraint optimizer?*
   We investigate the two threshold values $\tau_{solutions}$ and τ_{splits} to determine the effect these parameters have on the time taken to locate a solution to a system of constraints.

2. *Does the implementation of underlying algorithms compare to reported implementations in terms of execution time and scalability?*
   The $BC3$ algorithm is an important component of both the constraint solving (Section 5.3.4) and constraint optimization algorithms (Section 5.4). We investigate if our implementation of this algorithm is comparable to reported implementations.

3. *Is the interval-based quantified constraint optimizer able to locate solutions for standard universally quantified constraint solving benchmarks?*
   One requirement that we place on the interval-based quantified constraint optimizer is that it be capable of locating solutions to constraint systems that are consistent. We investigate the success of the algorithm over standard benchmarks, and compare the optimization algorithm with an implementation of an existing constraint solver.

4. *Is the constraint optimizer capable of locating solutions to constraint systems that describe relations between stationary and moving entities in a virtual environment?*
   The fiction-to-animation system is concerned with moving multiple entities around a Cartesian space with a time dimension, as prescribed by our definition of a scene in Section 5.1. We investigate benchmarks that perform this task, and determine which of the optimizer or solver is more applicable for solving these constraint systems.

[2]JEP: `http://www.singularsys.com/jep/` [accessed on 16 June 2008]

Benchmark	Source
CLPRevisited{a,b,c}	A toy benchmark described by Benhamou et al. (1994).
Broyden Banded Functions {5,10,20,40,80,160}	Used as benchmarks by Benhamou et al. (1994, 1999) to show scalability to increasing number of variables and constraints.
More-Cosnard {10,20,40,80}	Used as benchmarks by Benhamou et al. (1994, 1999) to show scalability to increasing number of variables and constraints.

Table 5.2: Benchmarks for verifying underlying narrowing and solving algorithms.

Benchmark	Source
Parabola Fitting, Circle, Robot, Point-path and Satellite	Described by Benhamou et al. (2004) as benchmarks for constraint solving over universally quantified variables.
Robust 1	Used by Ratschan (2006) from the bibliography available from his website (Ratschan, 2008).

Table 5.3: Benchmarks for verifying ability to solve universally quantified constraint systems.

This section presents a suite of experiments for answering the above questions, and describes benchmarks used in these experiments. Metrics for measuring success are defined, and possible sources of experimental error are identified.

5.5.1 Benchmarks

We divide benchmarks into three categories: those used to verify underlying algorithm implementation, those used to validate and compare the ability to solve standard universally quantified constraint benchmarks, and those that contain constraint systems likely to be created by the fiction-to-animation process.

5.5.1.1 Non-quantified benchmarks to verify underlying algorithms

The constraint optimizer relies on an implementation of outer and inner contracting operators that make use of the $BC3$ algorithm defined by Benhamou et al. (1994). The implementation of this algorithm is evaluated using standard constraint solving benchmarks in related research. These benchmarks are listed in Table 5.2. The Broyden and Cosnard functions are chosen as benchmarks because the functions can be increased in terms of the number of variables and constraints, and provide an indication as to the scalability of the constraint solving implementation.

The benchmarks listed in Table 5.2 are all non-quantified root-finding problems. Full formulations are listed in Appendix D.

5.5.1.2 Quantified benchmarks to verify solving ability

The goal of the interval-based quantified constraint optimizer is the ability to locate solutions for constraint systems that include a universally quantified variable. Standard benchmarks exist for validating this ability, listed in Table 5.3.

All benchmarks defined by Benhamou et al. (2004) are concerned with solving constraint systems in which each constraint contains a single universally quantified variable. Ratschan (2006)

Benchmark	Description
FRONT	Four objects, each constrained to appear *inFrontOf* and *near* one of the others. *noCollide* constraints over all objects.
SCENE	Six objects arranged using *toRightOf*, *toLeftOf*, *inFrontOf*, *behind*, *noCollide* and *near* constraints.
LAYOUT3	Three objects arranged with the *noCollide* constraint.
WAYPOINTS	One object constrained to pass through 3 fixed way-points, using the *near* constraint over 3 different time-intervals.
DYNAMIC1 STATIC1	One object is static, the other dynamic with trajectories of increasing degree in each dimension. *near* and *inFrontOf* are applied over a sub-interval of time. *noCollide* is applied over the entire interval of time.
DYNAMIC2	Both objects are dynamic, having trajectories of increasing degree in each dimension. *near* and *inFrontOf* are applied over sub-interval of time. *noCollide* is applied over entire interval of time.
COLLISION	n objects, each constrained to be *near* and *noCollide* with every other object. Increases in complexity with addition of each object, and for $n > 3$ is inconsistent.

Table 5.4: Benchmarks for fiction-to-animation constraints.

defines a suite of benchmarks in which each constraint contains more than one quantified variable. Our implementation of the interval-based quantified constraint optimizer currently handles a single quantified variable per constraint (although many quantified variables per system) because we anticipate that *time* is the only universally quantified value in our behaviour quantification problem. Therefore, out of the six possible "Robust" experiments defined by Ratschan (2008) we only use "Robust 1", the only benchmark from this suite in which constraints contain at most one universally quantified variable. Full formulations of these benchmarks are listed in Appendix D.

5.5.1.3 Quantified fiction-to-animation constraint systems

We use a suite of benchmark constraint systems that define the motion of entities in a d-dimensional Cartesian space. We limit the number of dimensions to three, because greater dimensionality is not required for visual representation. Entity trajectories are represented as n-degree Bezier splines defined over a universally quantified time variable. The constraint solving task is concerned with finding values for the control points of the Bezier splines so that the resulting trajectories satisfy the defined relational constraint over a specified time interval.

These benchmarks are divided into those describing static and those describing dynamic scenes, summarized in Table 5.4. We phrase constraints in terms of spatial relations (such as *inFrontOf* and *near*) that are motivated and formulated in Chapter 6. For these experiments, we assume that these benchmarks are representative of constraint systems to be produced by an automated process. For repeatability, the exact constraint formulations for these experiments are detailed in Appendix D.

The FRONT, SCENE, LAYOUT3, and COLLISION benchmarks are used as non-quantified constraint solving benchmarks and only contain entities without motion. The WAYPOINTS, DYNAMIC1STATIC1, and DYNAMIC2 benchmarks define scenes containing moving entities, and are used to evaluate constraint optimization for systems containing a universally quantified variable representing time.

5.5.2 Metrics

Time-to-solution is the most common metric used for evaluating constraint solving techniques (Benhamou et al., 2004; Ratschan, 2006). We measure this metric as the number of seconds between the invocation of the solving/optimization process and the detection of a solution.

We also report the δ value for the system of constraints during the constraint optimization process. This value is recorded once every ten seconds, and in these experiments we are interested in the following observations:

- δ_{start}: the largest relaxation constant for the system of constraints. The initial domain is a solution to this system of relaxed constraints.
- δ_{best}: the lowest value of δ achieved by the optimization process at the time of termination. This value indicates the remaining quantity of tightening required before a solution is obtained.

5.5.3 Sources of experimental error

A source of experimental error in this investigation is the computer system on which the implemented algorithm is executed. We perform our experiments on a Pentium 4 dual core 1.86GHz machine with 2GB of memory. The operating system permits multiple simultaneous processes, which means that the time-to-solution value for the solving/optimizing process is potentially affected by other executing processes. We mitigate this source of error by executing each process multiple times, and taking the average execution time as the time-to-solution.

Another source of experimental error is the type of rounding performed on a particular machine. We mitigate this problem using the `BigDecimal` class, and for all experiments enforce outward rounding at a precision level of 10^{-2}.

5.5.4 Results

This section describes the individual experiments performed during the investigation of the questions posed.

5.5.4.1 Parametrization and behaviour of the optimizer

This experiment examines the effect of the threshold parameters on the solving and optimizing ability of the $IOPT_2$ algorithm, and also provides initial insight into the nature of the relaxation constant. We investigate the following questions:

- *What values of τ provide the best compromise between execution time and low values for δ (where τ refers to both $\tau_{solutions}$ and τ_{splits} simultaneously)?*
- *What is the relationship between the value of the relaxation constant and the amount of execution time of the optimization process?*

We use three benchmarks for this experiment. The SATELLITE benchmark is used to represent a common universally quantified solving task, while the SCENE benchmark is used as an example

ε	$factor$	$\tau_{solution}$	τ_{splits}
10^{-2}	200	120	120

Table 5.5: Optimizer parameters used for experiments.

of a fiction-to-animation task. The COLLISION7 benchmark is also used as an example of an inconsistent constraint system.

The experiment is conducted as follows: a value of 5 is chosen for τ and the optimization process is invoked for a benchmark. The value of δ is recorded at 10 second intervals, until either a solution is found, or until the total execution time exceeds 90 minutes. This experiment is repeated ten times, doubling the value of τ for each experiment. This experiment does not investigate the independent effects of $\tau_{solutions}$ and τ_{splits}.

The level of precision ε is fixed at 10^{-2} for all experiments, and the distribution factor at each level in the search tree is fixed at $factor = 200$. τ_{time} is fixed at 90 minutes.

We plot the δ value as a function of time in Figure 5.11 for each experiment. For the SATELLITE problem time-to-solution is minimal using thresholds of 5, as shown in Figure 5.11(a). However, using a threshold of 5 the optimization process never reaches a solution in the time allotted for the SCENE benchmark, as shown in Figure 5.11(b). Instead, higher thresholds (of 20 and 80) are more successful. Thresholds of 10 and 160 produce the two lowest values for δ for the COLLISION7 benchmark in Figure 5.11(c).

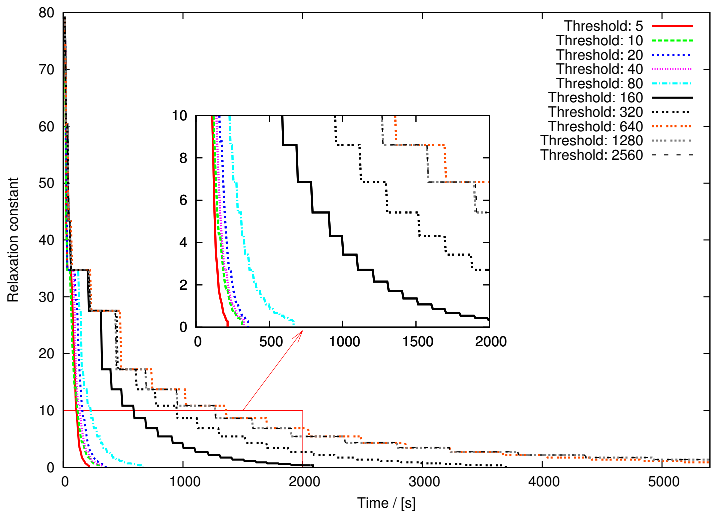
The above observations indicate that a dependence exists between the type of constraint system being solved and the most appropriate threshold values for minimal execution time. We choose a threshold value of 120 for the majority of experiments in subsequent sections, because this is an intermediate value within the range of successful thresholds observed in this experiment. However, we suggest a rule-of-thumb based on personal experience with the solver stating that this threshold value should be increased as the number of variables in a constraint system increases.

All three graphs in Figure 5.11 indicate that the value of δ decreases with the progression of time. The reduction in δ is step-wise in nature, reflecting the fact that reduction only occurs when solution boxes are located. The more difficult the solution finding process for a particular δ, the longer the overall relaxation value stays constant. However, the graphs indicate that given sufficient execution time, solutions or minimizers at low δ levels are located eventually.

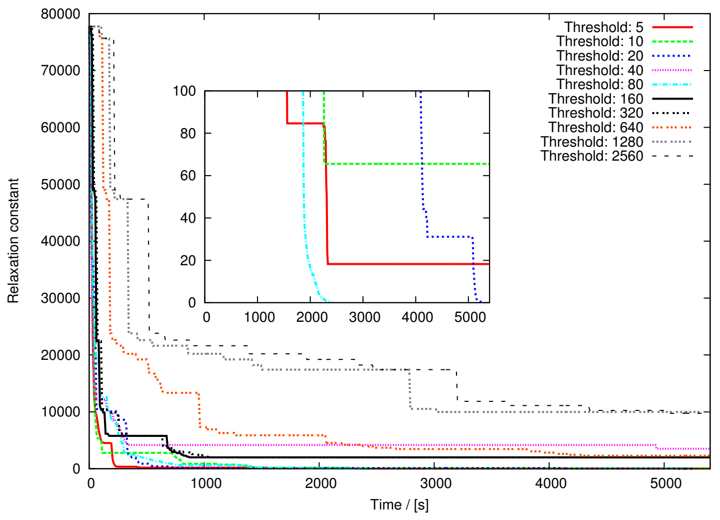
These experiments demonstrate that appropriate threshold values for the optimization algorithm are dependent on the type of constraint system being solved, but values reflected in Table 5.5 are generally successful. All subsequent experiments use these parameter values unless otherwise indicated. We also conclude that an inverse relationship holds between the amount of time spent optimizing a constraint system and the value of δ. This means that better approximations are expected given additional execution time of the algorithm, but the degree of improvement diminishes the longer the process runs.

5.5.4.2 Implementation of underlying algorithms

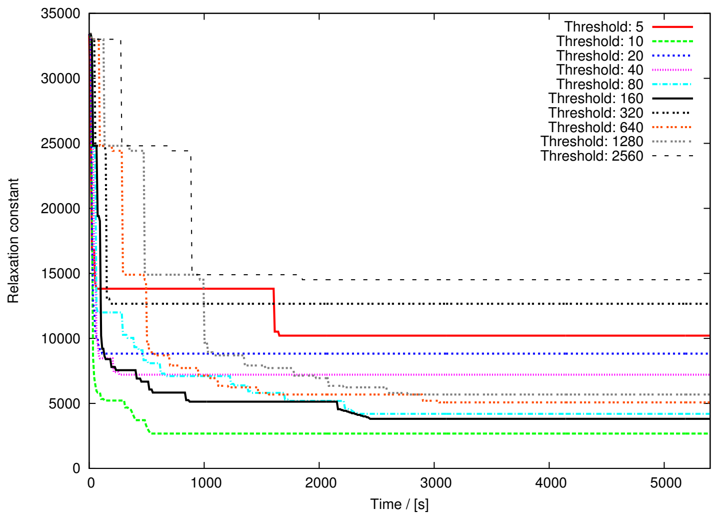
This experiment investigates whether the underlying algorithms are implemented to a comparable level to existing implementations. The reason for this experiment is that our implementation contains differences to existing methods (for example using a divide and conquer approach for



(a) Standard benchmark (consistent): Satellite



(b) Custom benchmark (consistent): Scene



(c) Custom benchmark (inconsistent): Collision7

Figure 5.11: Effect of threshold size on solving time.

Benchmark	Consistent	Correct	Time / [s]	Growth	Reported Growth
CLPRevisited(a)	Yes	✓	1.096	-	-
CLPRevisited(b)	Yes	✓	0.908	-	-
CLPRevisited(c)	No	✓	0.911	-	-
Broyden 5	Yes	✓	1.253	-	-
Broyden 10	Yes	✓	2.330	1.860	7.59 (Benhamou et al., 1994)
Broyden 20	Yes	✓	5.041	2.164	2.94 (Benhamou et al., 1994)
Broyden 40	Yes	✓	12.669	2.513	2.38 (Benhamou et al., 1994)
Broyden 80	Yes	✓	38.309	3.024	2.07 (Benhamou et al., 1994)
More-Cosnard 10	Yes	✓	2.414	-	-
More-Cosnard 20	Yes	✓	11.332	4.694	6.333 (Benhamou et al., 1999)
More-Cosnard 40	Yes	✓	79.811	7.043	3.052 (Benhamou et al., 1999)
More-Cosnard 80	Yes	✓	698.898	8.757	4.724 (Benhamou et al., 1999)

Table 5.6: Performance benchmarks for non-quantified constraint solving.

finding roots as opposed to the Newton method). We evaluate whether these differences impact scalability, and by doing so, affect the ability of an implemented universally quantified constraint solver and optimizer. We investigate the following questions:

- *Can our implementation of underlying algorithms be used to locate solutions to non-quantified benchmarks?*
- *Is our implementation comparable to existing implementations in terms of scalability (with regards to the number of variables and constraints)?*

We conduct this experiment using the $BC3$ algorithm for constraint propagation (Benhamou et al., 1994) over a number of benchmarks. The first benchmark is the CLPRevisited problem described by Benhamou et al. (1994), which we use to verify that the implemented algorithm is capable of detecting inconsistent constraint systems. Scalability is investigated using the Broyden-banded functions and More-Cosnard functions, both of which are systems of constraints with doubling numbers of variables and constraints. The time-to-solution is recorded for each benchmark.

We evaluate the scalability of the implemented algorithms using the ratio between consecutive instances of increasingly complex constraint systems. This ratio represents *growth*, a metric used by Benhamou et al. (1994) to evaluate the penalty in time that exists when increasing the complexity of the constraint systems.

The recorded time-to-solution values over the aforementioned benchmarks are listed in Table 5.6. A solution is found for all benchmarks using the implemented algorithm, and the inconsistent constraint system is correctly identified.

The growth ratio observed for consecutive Broyden and Cosnard functions is also presented in Table 5.6, along with the corresponding growth figures observed for related implementations (Benhamou et al., 1994, 1999). The growth figures reported for the related implementations are varied, and do not present constant growth for either the Broyden or the Cosnard functions. This variability is also reflected in our implementation, although we report growth figures within the same order of magnitude as the reference implementation. We report reduced growth ratios for some Broyden and Cosnard functions indicating that, up to a point, our implementation produces better scalability than the related implementation.

Benchmark	Reported	Solver	Optimizer	Relaxation constant	
($\varepsilon = 10^{-2}$)	Time / [s]	Time / [s]	Time / [s]	δ_{first}	δ_{best}
Parabola Fitting	0.02*	0.097	4.292	3.0	0
Circle	0.01*	0.500	13.452	4891.09	0
PointPath	7.85*	1.151	50.296	146.07	0
Robot	0.01*	0.954	5.271	10.98	0
Satellite	0.99*	2.860	653.308	79.19	0
Robust 1	$< 1^{+}$	0.053	2.561	197.80	0

*As reported by Benhamou et al. (2004).
+As reported by Ratschan (2006)

Table 5.7: Performance benchmarks for non-linear, universally quantified constraint solving.

These experiments demonstrate that our implementation of constraint propagation and solving algorithms are comparable to existing implementations, and that our implementation is capable of locating solutions to standard constraint solving benchmarks. The scalability of our implementation is also comparable to existing implementations with regards to the number of variables and constraints in the system.

5.5.4.3 Benchmarks in universally quantified constraint solving

This experiment investigates whether solutions can be found for standard benchmarks in universally quantified constraint solving using the optimization algorithm. We compare our implementation of the universally quantified *solving* process with an implementation reported in related research, and also investigate how an optimization approach compares over these problems. In particular:

- *Is the interval-based quantified constraint optimizer able to locate solutions for standard universally quantified constraint solving benchmarks?*
- *Is our implementation of the universally quantified constraint* solver *comparable to a related implementation reported by Benhamou et al. (2004)?*
- *Does the optimization method increase or decrease time-to-solution over standard benchmarks, and what is the nature of the relaxation constant for these benchmarks?*

This experiment is conducted as follows. Each benchmark is solved using our implementation of the *IPA* algorithm described in Section 5.3.4 and by Benhamou et al. (2004) and the time-to-solution recorded. The same benchmark is solved using the optimization algorithm, recording the time-to-solution, the initial relaxation constant, and the final relaxation constant.

The reported solving time (for finding the first solution) for existing systems over the set of benchmarks is listed in Table 5.7. Our implementation of the *IPA* solver successfully locates solutions to all the benchmarks.

The time-to-solution of our implemented solver is greater than the reported time-to-solution of the existing implementation in all cases except for the PointPath benchmark. We attribute this fact to a number of factors: our implementation is penalized in execution efficiency due to the virtual machine-based execution; our implementation uses a simple depth-first traversal across the search space, as opposed to the heuristic traversal strategies used by Benhamou et al. (2004); and the use of the divide and conquer approach for locating roots affects the convergence to solutions (as demonstrated in Section 5.5.4.2). In spite of these differences, the reported time-to-execution is of the same order (both within a range of seconds) in both implementations of the solver, indicating that the implementations are comparable.

The reported solving time for the constraint optimizer is presented in Table 5.7. We observe that the optimizer locates solutions to all benchmarks, but the time-to-solution in all cases is larger than that observed using the solver. This indicates that there is a penalty in execution time for using an optimization approach rather than a solving approach. We believe that this penalty is offset by the availability of an approximate solution at any point during the execution.

The initial values of the relaxation constant for each benchmark are also presented in Table 5.7. These values vary according to the benchmark, indicating that the interpretation of the constant is dependent on the formulation of the constraints. However, in all cases the constant reaches zero.

We conclude that our implementation of the constraint solver and optimizer is capable of locating solutions to benchmarks in universally quantified constraint solving. Our implementation of the solver is comparable to a related implementation, which means that further evaluations can be performed using this solver as a representative of the solving strategy (see Section 5.5.4.4, which compares solving and optimization strategies for fiction-to-animation benchmarks).

5.5.4.4 Benchmarks formulated for virtual environments

This experiment investigates the problem of finding solutions to constraint systems that are formulated for specifying time-based spatial relations in n-dimensional scenes. We investigate this problem to determine if the interval optimizer is capable of locating solutions to the types of constraints that are formulated by an automatic fiction-to-animation process. We investigate the following questions:

- *Is the interval-based quantified constraint optimization algorithm capable of locating solutions to constraint systems formulated for scenes without motion (non-quantified) as well as scenes with motion (quantified)?*
- *Does the constraint optimizer focus the search with the result of locating solutions in less time?*
- *Is the interval optimizer capable of producing approximate solutions for inconsistent constraint systems?*

This experiment is conducted using the set of fiction-to-animation benchmarks discussed in Section 5.5.1.3, which include constraint systems that specify scenes with and without motion, as well as constraint systems that are inconsistent. These benchmarks are solved using the implementation of the solver, and also using the interval optimizer. Time-to-solution is recorded using both techniques, and we fix an upper limit of 90 minutes for the solving/optimizing process.

Time-to-solution for each fiction-to-animation benchmark is presented in Table 5.8 for both the solver and the interval optimizer. The ∞ symbol is used when no solution is found within the allocated 90 minute period. The interval solver performs poorly over the set of consistent benchmarks, only finding a solution for two benchmarks in the set. The interval optimizer locates

Benchmark				Time/[s]		Relaxation constant		
Name	d	n	τ	Solver	Optimizer	δ_{first}	δ_{best}	Time-to-δ_{best}/[s]
Consistent benchmarks without motion:								
FRONT	2	0	120	∞	1929.28	3620.63	0	1929.282
SCENE	2	0	120	∞	2928.03	19339.99	0	2928.03
LAYOUT3	2	0	120	1364.438	60.78	9648.00	0	60.787
Consistent benchmarks with motion:								
WAYPOINTS	2	2	120	∞	578.768	63871.8	0	578.768
	2	3	500	∞	2071.92	56615.68	0	2055.775
DYNAMIC1STATIC1	1	1	120	218.493	22.53	594	0	22.533
	2	1	500	∞	173.95	1197.00	0	173.95
	2	2	600	∞	∞	927.71	24.20	232.154
DYNAMIC2	1	1	500	∞	337.35	81197.01	0	337.35
	2	1	500	∞	471.97	324007.02	0	471.97
	2	2	600	∞	∞	761070.10	25.08	833.07
Inconsistent benchmarks with and without motion:								
COLLISION4	2	0	120	∞	∞	9546.00	219.54	3882.076
	2	1	120	∞	∞	9642.24	4639.15	3213.20

d =dimension; n =degree of trajectory; τ =threshold value for both $\tau_{solutions}$ and τ_{splits}

Table 5.8: Performance benchmarks for fiction-to-animation constraints.

solutions to nine of the eleven benchmarks, including benchmarks with and without motion. This indicates that constraint systems exist where the interval optimizer locates solutions in less time than the interval solver. We believe that this is because the relaxation process guides the search towards solutions.

The starting relaxation constant δ_{first} and the lowest relaxation constant achieved by the interval optimizer δ_{best} are recorded in Table 5.8. In cases where solutions are located, δ_{best} reaches zero. Minimizers are found for benchmarks for which no solutions are located, and the relaxation constant δ_{best} provides an indication of how closely the minimizer approximates a solution through its distance to zero. We observe that the two consistent benchmarks for which no solutions are located result in minimizers that are close to zero in comparison with δ_{first}.

As expected, solutions to inconsistent benchmarks are neither located by the solver nor are they located by the optimizer. However, the optimizer provides minimizers to both benchmarks, and in the case of the scene without motion, produces a relaxation constant that is reduced by 97.70%. This result highlights the strength of the optimization approach, in that an approximate solution is provided even where no solution exists.

Table 5.8 presents the time-to-δ_{best} value, which is the time required to reach the lowest relaxation constant for a benchmark. In all cases where solutions are located, this value is equal to time-to-solution. However, for all inconsistent benchmarks, this value is less than 5400 (90 minutes), indicating that minimizers are located earlier than the cutoff time threshold.

The interval optimizer is effective at locating solutions for constraint systems that specify time-based behaviour. The optimizer focuses the search for solutions such that it locates solutions faster than the solving technique for these types of constraint systems, and also produces approximate solutions for inconsistent constraint systems.

5.5.5 Summary of findings

The experiments presented in Section 5.5.4 on page 136 provide insight into the properties of the interval-based quantified constraint optimizer with respect to the questions posed at the beginning of this section:

1. Threshold values for parametrizing the interval-based quantified constraint optimizer (to reduce execution time) vary according to the constraint system to be solved. The relaxation constant decreases as a function of time, and given sufficient time, the optimizer eventually locates a solution (for consistent systems).

2. Our implementation of the underlying algorithm is comparable to an existing implementation in terms of its ability to locate solutions to standard benchmarks, as well as in terms of its scalability with respect to the number of variables and constraints in a system.

3. The interval-based quantified constraint optimizer locates solutions to consistent constraint systems that contain universally quantified variables. There is a penalty for using an optimization process for finding solutions to universally quantified constraint systems, which is offset by the availability of an approximate solution at any point.

4. The constraint optimizer locates solutions to constraint systems that describe relations between stationary and moving entities in a virtual environment. It also derives approximate solutions for inconsistent constraint systems. Systems exist specifically in the fiction-to-animation domain where the trade-off experienced using the optimizing strategy (as opposed to a solving strategy) is nullified, and the optimizer locates solutions faster than the solver.

5.6 Conclusion

The interval-based quantified constraint optimizer provides a solution to the problem of quantifying behaviour in a virtual environment. If behaviour is phrased as symbolically formulated constraints, this mechanism is guaranteed to produce quantified values that satisfy the constraints or approximate valid behaviour (in cases where constraints conflict). We conclude the following with regards to the initial problem statement at the beginning of this chapter:

1. Constraints formulated over contiguous intervals of time and space are represented effectively using interval arithmetic. This enables direct solving of symbolically phrased constraints, and removes the need to transform systems into discrete representations.

2. Environment-independent spatial reasoning is achieved using an analytical formulation of constraints and their solutions using interval-based methods. This means that behaviour is quantified prior to the instantiation of the environment, and provides for the construction of environments at any point in the duration of a scene (for non-sequential filming).

   (a) The interval-based representation provides for the identification of sound quantified solutions over a contiguous interval (this conclusion is also reached by Benhamou et al. (2004)). This ensures that no erratic behaviour is produced in a scene that could potentially result from aliasing when using discrete solving methods.

   (b) Solutions returned by the interval-based solving/optimizing process are also phrased as contiguous intervals. This provides a range of options for valid behaviour in a scene.

3. The interval-based quantified constraint optimizer directs the search for solutions with the result that it locates solutions faster than a regular solving algorithm (for fiction-to-animation benchmarks). Relaxed constraints and iterative tightening successfully prune the search space posed by the original constraints. This is significant because it reduces search time, and also establishes a correlation between search time and the proximity of the process to an actual solution.

4. The interval-based quantified constraint optimizer addresses the trade-off between locating valid behaviour configurations and the amount of time spent searching for these configurations.

   (a) The optimization method effectively maintains a solution approximation that is refined with further execution of the optimizer. This suggests that a human with limited time is capable of continuing the fiction-to-animation process with an approximate solution, but is free to use a more refined solution at a later stage if the optimization process continues.

   (b) Optimization time is affected by the type of constraint system being solved. The use of thresholds reduces optimization time, but custom threshold values must be derived depending on the type of system. We use a rule-of-thumb stating that the greater the number of variables, the larger these thresholds should be for shorter time-to-solution.

5. A null solution is not the only possibility for inconsistent constraint systems that are created automatically. The interval-based quantified constraint optimizer produces approximate solutions even for inconsistent constraint systems. This guarantees quantified behaviour (regardless of how flawed), despite inconsistencies in the text or annotations.

The interval-based quantified constraint optimizer provides a mechanism for automatically quantifying scene behaviour. The derivation of analytical constraint systems, as well as the conversion of constraint solutions into corresponding visuals is discussed in Chapter 6.

This chapter contributes innovative work with regards to the text-to-graphics task and interval-based constraint solving:

- We present the first use of interval-based constraint solving in the text-to-graphics domain.
- To the knowledge of the author, this work is the first to employ quantified constraint solving techniques from an optimization perspective rather than a solving perspective. In particular:
  - This work presents the first method for optimizing universally quantified constraint systems where a solution is not guaranteed to exist.
  - This work is the first to provide a mechanism that allows early termination of the universally quantified solution-finding process while still providing values that can be used in subsequent applications.

- We contribute to the field of interval-based constraint solving through the creation of an inner contracting operator that functions over a set of constraints (rather than a single constraint at a time).
- Our method of using a divide and conquer approach for locating roots over univariate functions contributes an alternative strategy for implementing the $BC3$ algorithm that does not require the implementation of division or root operators. While resulting in slower convergence, this method does not seriously impact the performance or scalability of solving methods.

Future work includes investigating further enhancements to the $IOPT_2$ algorithm to speed up convergence to a minimum or solution. This is particularly important when scaling to the types and quantity of constraints automatically generated from annotated text. This issue is investigated further in Chapter 6.

Chapter 6

Population of virtual worlds

This chapter describes the creation of multi-modal animated 3D environments and films from annotated fiction text. The interpretation process consists of deriving high-level scene descriptions from the annotations that include the list of scenes to visualize and the contents and behaviour in each scene (Section 6.2). High-level behaviour is expressed using abstract constraints, and is quantified in a virtual environment through the optimization of corresponding analytical expressions (Section 6.3). The final step in the process automatically populates virtual environments so that they visually represent the descriptions in the original text (Section 6.4). We evaluate the interpretation process in terms of the degree of consistency of the automatically generated content, and in terms of the degree to which the visualized scene corresponds to the original text (Section 6.5). We provide conclusions regarding the automated interpretation process in Section 6.6.

6.1 Introduction

6.1.1 Problem statement

Annotated text forms the intermediate representation of the fiction-to-animation process. Given the existence of annotations identifying visual descriptions in fiction text, we investigate the interpretation of these annotations for creating corresponding virtual environments. This problem is characterized as follows:

1. Annotations (in categories such as those described in Chapter 4) identify scene related aspects of the fiction text. These annotations must be interpreted for creating scene descriptions (such as: the list of scenes to be visualized; the contents of each scene; and the behaviour of entities in each scene) in a structured manner.

2. Given the presence of structured scene descriptions, corresponding virtual environments must be instantiated and populated, a problem that includes: choosing appropriate visual icons to represent entities described in the text; creating geometry that visualizes appropriate background scenery; and visualizing the behaviour specified by the annotations. This problem also includes the automatic construction of multi-modal presentations of the fiction text.

Room:
Anne slept in the next room. Julian ran in and shook her. "Wake up! It's Tuesday! And the sun's shining." Anne woke up with a jump and stared at Julian joyfully. "It's come at last!" she said. "I thought it never would. Oh, isn't it an exciting feeling to go away for a holiday!"
Outside:
They started soon after breakfast. Their car was a big one, so it held them all very comfortably. Mother sat in front with Daddy, and the three children sat behind, their feet on two suitcases. In the luggage-place at the back of the car were all kinds of odds and ends, and one small trunk. Mother really thought they had remembered everything.
London:
Along the crowded London roads they went, slowly at first, and then, as they left the town behind, more quickly.
Country:
Soon they were right into the open country, and the car sped along fast. The children sang songs to themselves, as they always did when they were happy.

Figure 6.1: Example decomposition of fiction text into scenes, from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

The virtual environments created from the interpreted text should conform to the descriptions in the original text. The above problems are investigated under the assumption that annotated text is available (produced with the aid of the machine learning algorithm in Chapter 4).

6.1.2 Problem formulation

The interpretation of annotated fiction text for creating virtual environments is subjective to the human performing the task. Previous work interprets a fiction book as an account of a *virtual universe*, where events in the book are reported out of order for dramatic purposes (Glass and Bangay, 2007a). The creation of a virtual universe means "unraveling" the order of events so that they occur in the correct order in the 3D environment. Fiction text does not always provide explicit indications of event order, which means that knowledge-rich reasoning must be used to automate the unraveling process (Ma and McKevitt, 2004a; Ma, 2006).

We use an alternative interpretation that divides a book into a number of *scenes* (the definition of which is provided in Chapter 5, Definition 5.1 on page 107), each of which is independent and considered its own virtual universe. Sequences of tokens in a fiction book describe one particular scene, examples of which are presented in Figure 6.1. Textual triggers for Setting, Object, Avatar, Transition, and Relation annotations within each segment of text are interpreted only in the scene in which they occur.

The identification of which scenes to instantiate forms the first part of the interpretation process. Subsequent tasks include interpreting annotations for identifying the entities that occur in each scene, and how these entities behave. We first specify these scene details on a high level, recognizing that ambiguity manifests during the interpretation of annotations. This allows for human intervention if required. For example, behaviour in a scene is first expressed using high-level abstract constraints that are human readable and conducive to review and modification if necessary.

We design abstract constraints to allow direct conversion into symbolic analytical equivalents. Solutions to these constraints quantify behaviour in a virtual environment, and are determined using the interval-based quantified constraint optimizer (described in Chapter 5).

Given structured scene descriptions and quantified scene behaviour, the final step in the interpretation process instantiates animated 3D virtual environments. This includes selecting geometry

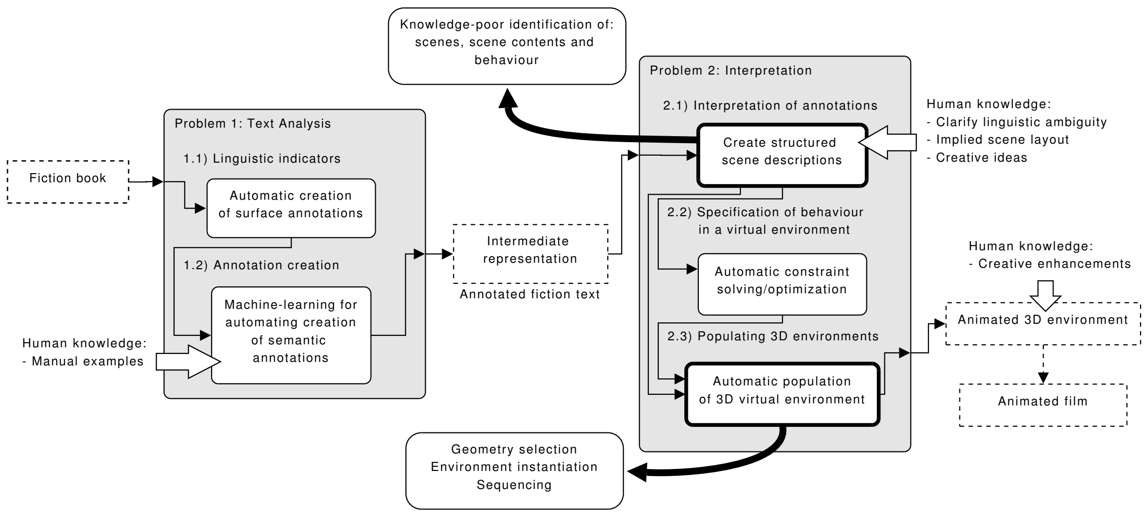

Knowledge-poor identification of: scenes, scene contents and behaviour
Problem 1: Text Analysis
1.1) Linguistic indicators
Fiction book
Automatic creation of surface annotations
1.2) Annotation creation
Machine-learning for automating creation of semantic annotations
Human knowledge:
- Manual examples
Intermediate representation
Annotated fiction text
Problem 2: Interpretation
2.1) Interpretation of annotations
Create structured scene descriptions
Human knowledge:
- Clarify linguistic ambiguity
- Implied scene layout
- Creative ideas
2.2) Specification of behaviour in a virtual environment
Automatic constraint solving/optimization
Human knowledge:
- Creative enhancements
Animated 3D environment
2.3) Populating 3D environments
Automatic population of 3D virtual environment
Animated film
Geometry selection
Environment instantiation
Sequencing

Figure 6.2: Illustration of the constraint creation and scene creation modules in the fiction-to-animation process.

to represent entities in each environment as well as applying the correct behaviour to entities in each scene.

Each of the above three problems are investigated in this chapter, namely the interpretation of annotations for specifying scene descriptions, the quantification of behaviour in a scene, and the instantiation and population of virtual environments.

6.1.3 Context

The interpretation task of the fiction-to-animation process is investigated in this chapter. The context of this problem within the conversion process is illustrated in Figure 6.2). We assume the existence of annotated fiction text, the automatic creation of which is described in Chapter 4. In this chapter we investigate the interpretation of these annotations in creating quantified constraint systems, solutions to which are found using the interval-based quantified constraint optimizer described in Chapter 5.

Techniques for automatically populating a virtual environment from interpreted annotations and quantified behaviour are described in this chapter. The output of these processes includes multi-modal animated 3D virtual environments, and corresponding animated films. The work presented in this chapter is a more detailed description of research by the same author (Glass and Bangay, 2008).

6.2 High-level scene descriptions from interpreted annotations

We use the term *scene description* to collectively describe: the list of scenes to be instantiated as virtual environments, the contents of each scene, and the behaviour of entities in each scene. This information is expressed using high-level (but structured) descriptions. This ensures that

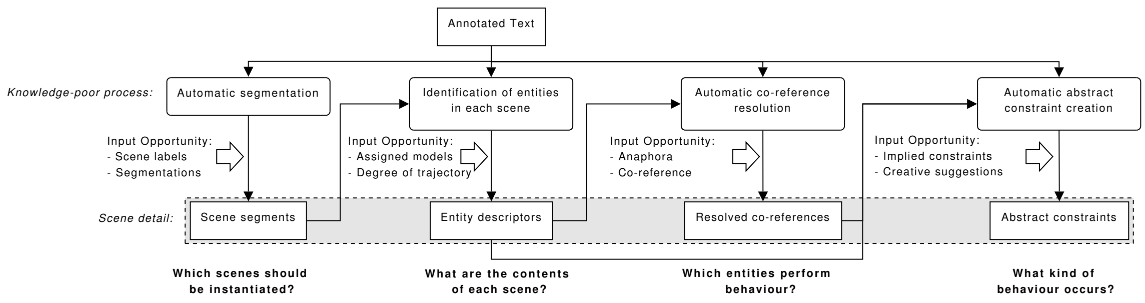

Annotated Text
Knowledge-poor process:
Automatic segmentation
Identification of entities in each scene
Automatic co-reference resolution
Automatic abstract constraint creation
Input Opportunity: - Scene labels - Segmentations
Input Opportunity: - Assigned models - Degree of trajectory
Input Opportunity: - Anaphora - Co-reference
Input Opportunity: - Implied constraints - Creative suggestions
Scene detail:
Scene segments
Entity descriptors
Resolved co-references
Abstract constraints
Which scenes should be instantiated?
What are the contents of each scene?
Which entities perform behaviour?
What kind of behaviour occurs?

Figure 6.3: Illustration of the interpretation module for creating constraints from annotated fiction text.

"Mother, have you heard about our summer holidays yet?" said Julian, at the breakfast-table. "Can we go to Polseath as usual?"
"I'm afraid not," said his mother. "They are quite full up this year."
The three children at the breakfast-table looked at one another in great disappointment. They did so love the house at **<setting>Polseath</setting>**. The **<setting>beach</setting>** was so lovely there, too, and the bathing was fine.
"Cheer up," said Daddy. "I dare say we'll find somewhere else just as good for you. And anyway, Mother and I won't be able to go with you this year. Has Mother told you?"

Figure 6.4: Example of text containing Setting annotations, from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

scene descriptions are human readable (providing for manual modification or for the injection of creativity), but are also conducive to further processing using automated techniques.

We present a work-flow of automated knowledge-poor techniques for creating high-level scene descriptions from annotated fiction text. The different categories are illustrated in Figure 6.3, as are the knowledge-poor processes used for their creation. This work-flow permits human intervention for handling exceptions that occur during interpretation not handled by the knowledge-poor techniques.

The knowledge-poor techniques for producing each category of scene description are discussed in the following sections.

6.2.1 Scene segmentation

As described in Section 6.1.2, each scene corresponds to a single physical setting. Annotation categories such as Setting (defined in Chapter 4) identify physical settings, and we use these annotations to segment text into scenes.

Scene descriptions are not always explicitly mentioned in fiction writing, an example of which is presented in Figure 6.4. The initial setting in this example is not explicitly stated, and every token up until "Polseath" is automatically assigned a DEFAULT setting. Human intuition indicates that the described scene is likely to be occurring in a KITCHEN based on evidence from described objects such as "breakfast-table". A human is able to specify the correct setting manually for exceptional cases such as this.

The next step in the abstract constraint creation process is concerned with identifying entities in each scene.

6.2.2 Identification of entities

The entities that appear in a scene are identified using annotation categories such as Avatar and Object, and we construct a list of entities identified by these annotations for each scene. We represent entities using *entity descriptors,* which are scene-independent structures that associate information (regarding instantiation in a virtual environment) with each entity mentioned in a book. Descriptors contain information unique to an entity, including the geometric model to represent the entity graphically in a virtual environment, and the type of motion associated with the entity.

An entity descriptor is created for every unique entity in the fiction book. Entities such as avatars occur in many different scenes, and the same entity descriptor for an entity is used across different scenes.

When an entity descriptor is created, a geometric model is automatically selected to represent the entity in a virtual environment. We source models from a library of geometric models. Each model in the library is annotated with descriptive keywords that are matched against the annotated tokens. The types of model returned by the library depend on the category of annotation, and for Avatar and Object categories are as follows:

- **Avatars:** Avatars are assumed to be human, and only humanoid models are appropriate for this category. The token annotated in this category is likely to be the name of the avatar and we use a gazetteer of names to determine the gender, and select the appropriately gendered model.

- **Objects:** The token annotated in this category is used as a search term when querying the model library, matching the search term with the descriptive keywords associated to each model. In cases where no matching keywords are found, we use synonyms of the annotated token as search terms (provided by WordNet Fellbaum (1998)). We also use hypernyms of the annotated token as a search term, abstracting the tokens until no further abstraction is possible. If no matching models are found, we use a default placeholder object (a cube) to represent the object visually.

The accurate selection of geometric models depends on the annotation conventions adopted for creating Object and Avatar annotations. For example, in our annotated version of the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942), the character Timothy is annotated as an avatar, but Timothy is actually a dog. This exception is corrected manually.

We also use the annotation category to determine whether a model is static or dynamic in a virtual environment. Avatars are assigned trajectories that permit motion in the virtual environment, while the trajectories assigned to objects permit no movement.

6.2.3 Co-reference resolution

Ambiguous references to entities exist in the annotated text. For example, anaphora is often used to refer to entities (the use of "he" or "it"). Annotation categories such as Transition and Relation potentially identify anaphoric tokens as the *subject* or *object* of the behaviour, and these must be resolved to determine the entities that are involved in the behaviour described by the annotations.

```
    Entities:
      Avatars:                                        Objects:
        ANNE                                            COW
        DADDY

Anne/ANNE didn't very much like a big brown cow/COW who came up close and stared at her/ANNE, but
it/COW went away when Daddy/DADDY told it/COW to.
```

Figure 6.5: Illustration of an entity list and resolved co-reference.

We resolve instances of personal pronominal anaphora (such as “he” and “she”) by keeping track of the last explicitly mentioned male and female avatars, and matching the gender of the anaphoric token with the corresponding gender-matched avatar (avatar gender is determined using the method described in Chapter 3). Currently, we do not resolve non-gender specific pronouns such as “it” and we do not cater for general anaphoric cases (for example, in the case where the word “boy” indirectly refers to a specific male avatar). Human intervention is permitted to resolve these exceptions should they occur. Future enhancements to this process include the use of more sophisticated anaphora resolution techniques (Lappin and Leass, 1994; Nasukawa, 1994; Kennedy and Boguraev, 1996; Mitkov, 1998; Mitkov et al., 2002; Castaño et al., 2002; Dimitrov, 2002) and co-reference resolution methods (Baldwin, 1997; Dimitrov, 2002).

An example of an entity list created for a scene from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942) is presented in Figure 6.5, which contains entities in the category of Avatar and Object. The sentence contains examples of resolved co-references with regards to the entities listed above. The pronoun “her” is automatically linked to the entity “Anne”. Pronouns such as “it” are resolved manually to the “cow” entity.

6.2.4 Abstract constraints for specifying behaviour

We create a human readable summary of behaviour in a scene using abstract constraints. Abstract constraints are phrased in terms of the entities involved, the type of behaviour that occurs, as well as the interval of time over which the behaviour is to take place. They are structured to allow for automatic conversion into equivalent analytical expressions:

Definition 6.1. An abstract constraint specifies a time-quantified relation between two entities in an environment. Abstract constraints are defined in terms of the following fields:

- **Subject:** The entity responsible for the behaviour.
- **Relation:** The type of behaviour described by the constraint.
- **Object:** The reference entity or anchor point of the behaviour.
- **Start-time:** The time in the scene at which the behaviour begins.
- **End-time:** The time in the scene at which the behaviour terminates.

Annotation categories that describe behaviour (such as Transition and Relation) map directly to abstract constraints. The token in the *subject* text-reference of the annotation is linked to an entity descriptor (this link is created during the co-reference resolution step), and the identifier

Annotated text:

He/JULIAN stole <**transition subject=“he” type=“INSIDE”**>**in**</**transition**>. ...

He tiptoed by him to the table/TABLE <**relation subject=“table” object=“chair” type=“BEHIND”**>**behind**</**relation**> his uncle's chair/CHAIR.

Corresponding abstract constraints:

```
CONSTRAINT 1:            CONSTRAINT 2:
 Subject:    JULIAN       Subject:    TABLE
 Relation:   INSIDE       Relation:   BEHIND
 Object:     ROOM         Object:     CHAIR
 Start-time: 5            Start-time: 0
 End-time:   30           End-time:   30
```

Figure 6.6: Example abstract constraints derived from annotated behaviour in a fiction extract.

for the descriptor is used for the subject field of the constraint. The same applies for the *object* text-reference and corresponding field in the constraint. The *type* field for both Transition and Relation annotations is used in the relation field of the abstract constraint. An example of a set of abstract constraints is presented in Figure 6.6.

Abstract constraints are created from annotation categories specifying behaviour (explicit constraints), or are created automatically to ensure physical believability in the scene (implicit constraints). Implicit *noCollide* constraints are added for every pair of entities in the scene to ensure that entities do not interpenetrate. The one exception is the case where one entity is *inside* another entity (specified by a Relation annotation). We also create an implicit *near* constraint for certain types of Relation annotation, including $inFrontOf$, *behind*, $toLeftOf$, $toRightOf$, $onTopOf$ and *below*.

Temporal information is required for the start-time and end-time fields. Assume a fiction book is narrated in the audio modality (corresponding to being read aloud) using, for example, a speech synthesizer. Language units in the book use a finite interval of time when verbalized. Each scene consists of a sequence of language units, and the total time taken for the play-back of the corresponding audio provides an indication of scene duration, and a reference against which the time-fields of abstract constraints can be specified. We describe a time-line derived from audio narrations as a *presentation time-line* because it represents the original order of presentation intended by the author (and avoids the event “unraveling” problem described in Section 6.1.2).

Time information for abstract constraints is derived by calculating the time-value of an annotation trigger with respect to the presentation time-line. Speech synthesis produces audio at the sentence level, and the timing for an annotation trigger is estimated in terms of the token's offset in the sentence and the starting time of the sentence in the scene:

$$\begin{aligned} offset\,of\,trigger\,in\,sentence &= \frac{position\,of\,token\,in\,sentence}{number\,of\,tokens\,in\,sentence} * audio\,length\,of\,sentence \\ start\,time\,of\,trigger &= start\,time\,of\,sentence + offset\,of\,trigger\,in\,sentence \end{aligned}$$

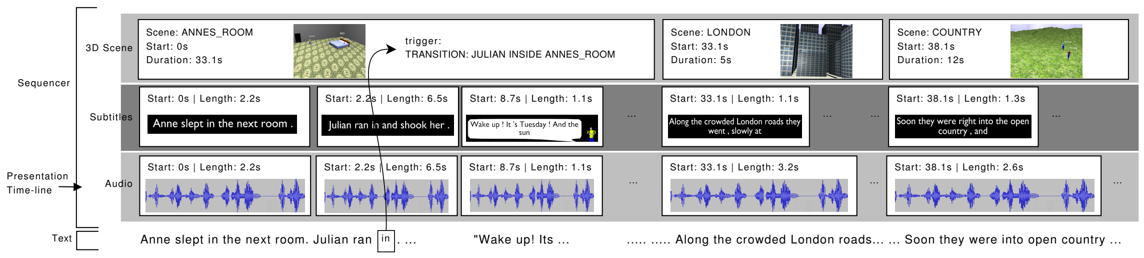

Sequencer
Presentation Time-line
Text
3D Scene
Subtitles
Audio
Scene: ANNES_ROOM
Start: 0s
Duration: 33.1s
trigger:
TRANSITION: JULIAN INSIDE ANNES_ROOM
Scene: LONDON
Start: 33.1s
Duration: 5s
Scene: COUNTRY
Start: 38.1s
Duration: 12s
Start: 0s | Length: 2.2s
Anne slept in the next room .
Start: 2.2s | Length: 6.5s
Julian ran in and shook her .
Start: 8.7s | Length: 1.1s
Wake up ! It 's Tuesday ! And the sun
...
Start: 33.1s | Length: 1.1s
Along the crowded London roads they went , slowly at
... ...
Start: 38.1s | Length: 1.3s
Soon they were right into the open country , and
...
Start: 0s | Length: 2.2s
Start: 2.2s | Length: 6.5s
Start: 8.7s | Length: 1.1s
...
Start: 33.1s | Length: 3.2s
...
Start: 38.1s | Length: 2.6s
...
Anne slept in the next room. Julian ran in. ...
"Wake up! Its ...
.... Along the crowded London roads... ... Soon they were into open country ...

Figure 6.7: Illustration of the relationship between the presentation time-line and behaviour in a 3D virtual environment.

An illustration of the relationship between fiction text, the audio files, and their interpretation as a presentation time-line is illustrated in Figure 6.7. The manner in which abstract constraints are synchronized is also indicated.

The derivation of a start-time and end-time for an abstract constraint depends on the category of annotation from which the constraint is derived:

1. **Transition:** By default, the start-time of the constraint corresponds to the time of the trigger of the annotation, while the end-time corresponds to the end of the scene. A sequential list is maintained for each entity that records all the constraints that apply to it. Once all constraints are added, each entity's list is traversed from start to end, setting the end-time of each constraint to the start-time of the following constraint. If the first constraint for an entity is of type *inside* at a time greater than zero, then an implicit *outside* constraint is inserted from time 0 to the start-time of the first constraint, to ensure that the entity is initially outside the scene. Any entities that are not constrained by a transition constraint are automatically assigned an implicit *inside* constraint for the duration of the scene to ensure that they appear in the scene.

2. **Relation:** The start-time and end-time for relation constraints depend on the trajectory of the entities involved. If both entities are static, then the constraint is set to last the duration of the scene (because the entity is unable to move to satisfy the constraint). If a dynamic entity is involved then the start-time of the constraint is set to the time of the trigger token of the annotation, while the end-time is set to the end-time of the scene. If a relation constraint applying to an entity overlaps with an *outside* constraint for that entity, then the end-time of the relation constraint is set to be the start-time of the *outside* constraint for that entity. This avoids potential conflicts in terms of relations that cannot hold while an entity is outside the scene.

High-level scene descriptions that are created using the knowledge-poor methods in this section include: a list of scenes to be instantiated; the contents of each scene (specified using descriptors); and the behaviour of entities in each scene (specified using abstract constraints). Before a virtual environment can be instantiated for each scene, precise values that quantify the behaviour in each virtual environment are required.

6.3 Quantified behaviour in virtual environments

Abstract constraints describe behaviour on a high level, but do not provide precise numerical values for visualizing behaviour in a virtual environment. If abstract constraints are converted into symbolic analytical expressions, then the interval-based quantified constraint optimizer described in Chapter 5 can be used to find these values.

We define entity behaviour in terms of location, which is expressed as a function of time so that entities are capable of moving in the virtual world. We represent this behaviour using *trajectories* that are defined in terms of a set of variables. These trajectories should conform to the behaviour summarized in the set of abstract constraints.

6.3.1 Constrained model trajectories

We assume a scene exists over the time interval $[t_0, t_1]$. Each model M in the scene of dimension d is associated with a trajectory $\mathbf{r}_d^M(t) = [r_1(t), ..., r_d(t)]$ parametrized according to time t. Trajectories are defined as parametric curves over time, using for example, a Bezier curve of degree n (Buss, 2003):

$$\mathbf{r}^M(t) = \sum_{i=0}^{n} B_i^n(t)\mathbf{p}_i$$

with the blending function $B_i^n(t) = t^i(1-t)^{n-i}$. The control points $\mathbf{p}_i$ specify the shape of the curve, and consequently the trajectory of the associated model in the scene. The problem of quantifying behaviour is transformed into finding values for the control points so that the set of constraints is satisfied over the duration of the scene.

Abstract constraints express spatial relations between two entity models over an interval of time. We define an equivalent analytical constraint as follows:

Definition 6.2. *Quantified Constraint*: Let $\mathbf{r}^M(t)$ and $\mathbf{r}^N(t)$ be trajectories for two models M and N respectively. A quantified constraint $c(\mathbf{r}^M, \mathbf{r}^N, [t_{start}, t_{end}])$ is a symbolic expression that expresses a relation between the two trajectories that exists for all time between t_{start} and t_{end}.

Quantified constraints are derived for each abstract constraint. The formulation of the expression depends on the relation field of the abstract constraint. In this exposition, the Transition and Relation annotation categories are used to specify behaviour (and create abstract constraints), and expressions must be derived for the different *types* of behaviour in these categories. Transition annotations are defined as one of two semantic *types*, namely *inside* or *outside*, indicating whether the entity is inside or outside the scene (described in Chapter 4). A Relation annotation is one of several *types*, including $inFrontOf$, *behind*, $toLeftOf$, $toRightOf$, $onTopOf$, or *below*. Behaviour types are interpreted as spatial relationships between two entities.

6.3.1.1 Spatial constraints

Let M and N be the object models for two items in a scene with dimension d. Assume that each model is bound using a bounding sphere of radius a_M and a_N respectively. Let $\mathbf{r}^M(t)$ and $\mathbf{r}^N(t)$ be trajectories for the two models M and N respectively. We define the spatial relations in terms of three canonical quantified constraints, namely *near*, *noCollide*, and *directionRelation*.

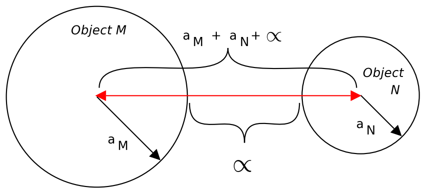


Figure 6.8: Illustration of the spatial relationship described by the *near* constraint.

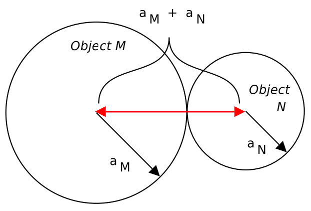


Figure 6.9: Illustration of the spatial relationship described by the *noCollide* constraint.

Near The constraint $near(\mathbf{r}^M(t), \mathbf{r}^N(t), [t_{start}, t_{end}])$ specifies that two models must be close to one another over time interval $t \in [t_{start}, t_{end}]$, expressed in terms of the Euclidean distance as follows:

$$\begin{aligned} ||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 &< (a_M + a_N + \alpha)^2 & \forall t \in [t_{start}, t_{end}] \\ ||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N + \alpha)^2 &< 0 & \forall t \in [t_{start}, t_{end}] \end{aligned}$$

where α is the minimum distance considered to verify the term "near" (for instance, 1 meter). The *near* scenario is illustrated in Figure 6.8.

Abstract constraints with the relation *near* or *inside* are converted into expressions of this type. For *near* constraints, α is chosen as 1 meter, but for *inside* constraints there must be no distance between the two models, and so we use a value of $\alpha = 0$ for constraints of this type.

NoCollide The constraint $noCollide(\mathbf{r}^M(t), \mathbf{r}^N(t), [t_{start}, t_{end}])$ specifies that two models must not interpenetrate over time interval $t \in [t_{start}, t_{end}]$, expressed in terms of the Euclidean distance as follows:

$$\begin{aligned} ||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 &> (a_M + a_N)^2 & \forall t \in [t_{start}, t_{end}] \\ ||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N)^2 &> 0 & \forall t \in [t_{start}, t_{end}] \end{aligned}$$

The *noCollide* scenario is illustrated in Figure 6.9. Abstract constraints with the relation *noCollide* are converted into expressions of this type.

DirectionRelation The constraint $directionRelation(\mathbf{r}^M(t), \mathbf{r}^N(t), [t_{start}, t_{end}])$ specifies the location of M with reference to N. We define $\mathbf{u}(t)$ as a direction vector of N, and $\mathbf{v}(t)$ as the vector indicating the direction from N to M, calculated as $\mathbf{v}(t) = \mathbf{r}^M(t) - \mathbf{r}^N(t)$. For simplicity, the time parameter is not shown in the following formulations.

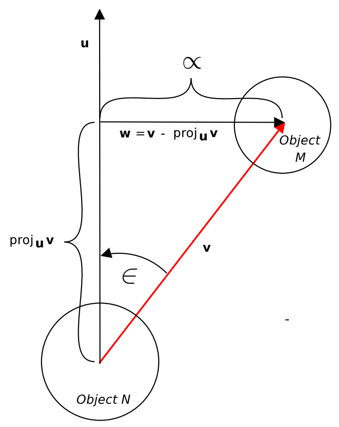


Figure 6.10: Illustration of the spatial relationship described by the *directionRelation* constraint.

Consider Figure 6.10. For model M to be related to N according to the direction $\mathbf{u}$, the angle ϵ should be less than a certain value. This is also phrased in terms of the length of the vector $\mathbf{w}$, which should be less than an amount α, that is $||\mathbf{w}|| < \alpha$ or:

$$\begin{aligned} ||\mathbf{w}||^2 &< \alpha^2 \\ ||\mathbf{w}||^2 - \alpha^2 &< 0 \end{aligned}$$

We calculate $\mathbf{w}$ using the projection of $\mathbf{v}$ on $\mathbf{u}$: $\mathbf{w} = \mathbf{v} - proj_{\mathbf{u}}\mathbf{v} = \mathbf{v} - \frac{\mathbf{u}}{||\mathbf{u}||^2}(\mathbf{u} \cdot \mathbf{v})$.

Let ϵ be the angle between $\mathbf{u}$ and $\mathbf{v}$, then α is expressed in terms of ϵ by the trigonometric relation: $sin(\epsilon) = \alpha/||\mathbf{v}||$. The constraint above is rephrased as follows:

$$\begin{aligned} ||\mathbf{w}||^2 - \alpha^2 &< 0 \\ \mathbf{w}\cdot\mathbf{w} - (\mathbf{v}\cdot\mathbf{v})sin^2(\epsilon) &< 0 \\ \left(\mathbf{v}||\mathbf{u}||^2 - \mathbf{u}(\mathbf{u}\cdot\mathbf{v})\right)\cdot\left(\mathbf{v}||\mathbf{u}||^2 - \mathbf{u}(\mathbf{u}\cdot\mathbf{v})\right)/||\mathbf{u}||^4 - (\mathbf{v}\cdot\mathbf{v})sin^2(\epsilon) &< 0 \\ \left((\mathbf{v}\cdot\mathbf{v})||\mathbf{u}||^4 - 2||\mathbf{u}||^2(\mathbf{u}\cdot\mathbf{v})^2 + ||\mathbf{u}||^2(\mathbf{u}\cdot\mathbf{v})^2\right)/||\mathbf{u}||^4 - (\mathbf{v}\cdot\mathbf{v})sin^2(\epsilon) &< 0 \\ (\mathbf{v}\cdot\mathbf{v})(1 - sin^2(\epsilon)) - (\mathbf{u}\cdot\mathbf{v})^2/||\mathbf{u}||^2 &< 0 \\ ||\mathbf{u}||^2||\mathbf{v}||^2cos^2(\epsilon) - (\mathbf{u}\cdot\mathbf{v})^2 &< 0 \end{aligned}$$

The reference direction of $\mathbf{u}$ depends on the type of directional relation being specified. For example, if an $inFrontOf$ relation is specified, then $\mathbf{u}$ is the forward facing direction of the model N. If $\mathbf{r}^N(t)$ is of a degree greater than zero (dynamic entity), then the forward direction is the tangent of the trajectory, calculated using the derivative of $\mathbf{r}^N(t)$. If N is a static model, then $\mathbf{u}$ is an arbitrary vector (default direction, or specified by a human).

Constraints of type $inFrontOf$, *behind*, $toLeftOf$, $toRightOf$, $onTopOf$, or *below* are defined using the same formulation of the *directionRelation* constraint, but varying in the choice of $\mathbf{u}$ for determining the reference direction.

6.3.1.2 Constraint systems

Every abstract constraint for a particular scene is converted into an equivalent quantified constraint, the result of which is a system of constraints. The mapping used to convert abstract constraints to

Abstract relation	Quantified constraint	Parameters
M *near* N	$near(\mathbf{r}^M(t), \mathbf{r}^N(t), [t_0, t_1])$	$\alpha = 1$
M *inside* N	$near(\mathbf{r}^M(t), \mathbf{r}^N(t), [t_0, t_1])$	$\alpha = 0$
M *noCollide* N	$noCollide(\mathbf{r}^M(t), \mathbf{r}^N(t), [t_0, t_1])$	-
M *inFrontOf* N M *behind* N M *toRightOf* N M *toLeftOf* N M *onTopOf* N M *below* N	$directionRelation(\mathbf{r}^M(t), \mathbf{r}^N(t), [t_0, t_1])$	$\epsilon = 45°$

Table 6.1: Direct mapping between abstract constraints and quantified constraints.

quantified constraints depends on the relation field of the abstract constraint and is summarized in Table 6.1.

We provide an example to illustrate the conversion of abstract constraints into quantified constraint systems. Consider the following set of abstract constraints:

```
CONSTRAINT 1:             CONSTRAINT 2:           CONSTRAINT 3:
 Subject:     JULIAN       Subject:     TABLE      Subject:     CHAIR
 Relation:    NEAR         Relation:    NOCOLLIDE  Relation:    INFRONTOF
 Object:      TABLE        Object:      JULIAN     Object:      TABLE
 Start-time: 0             Start-time: 0           Start-time: 0
 End-time:    5            End-time:    5          End-time:    5
```

For the sake of illustration, we assume that entity descriptors exist for Julian, Table, and Chair and that geometric models J, T, and C are associated with these respective entities. Each of these entities is assigned a trajectory, and as described in Section 6.2.2, the avatar is dynamic (trajectory of degree greater than 0), while the two objects are static (degree 0):

- $\mathbf{r}^J(t) = (1-t)\mathbf{p}_0^J + t\mathbf{p}_1^J$
- $\mathbf{r}^T(t) = \mathbf{p}_0^T$
- $\mathbf{r}^C(t) = \mathbf{p}_0^C$

In this example, we assume that the scene is defined in two dimensions, that is, each control point is of the form $\mathbf{p}_i = (x_i, z_i)$. The behaviour for this scene is therefore expressed in terms of eight variables $V = \{x_0^J, z_0^J, x_1^J, z_1^J, x_0^T, z_0^T, x_0^C, z_0^C\}$. Constraint 1 is phrased in terms of these variables using the formulation of the *near* expression:

$$\begin{aligned}
& c_1 : near(\mathbf{r}^J(t), \mathbf{r}^T(t), [0,5]) & \\
\Leftrightarrow\quad & ||\mathbf{r}^J(t) - \mathbf{r}^T(t)||^2 - (a_J + a_T + \alpha)^2 < 0 & \forall t \in [0,5] \\
\Leftrightarrow\quad & ||(1-t)\mathbf{p}_0^J + t\mathbf{p}_1^J - \mathbf{p}_0^T||^2 - (a_J + a_T + \alpha)^2 < 0 & \forall t \in [0,5] \\
\Leftrightarrow\quad & ((1-t)x_0^J + tx_1^J - x_0^T))^2 + ((1-t)z_0^J + tz_1^J - z_0^T))^2 - (a_J + a_T + \alpha)^2 < 0 & \forall t \in [0,5]
\end{aligned}$$

Constraint 2 is phrased using the *noCollide* expression:

$$\begin{array}{lll}
 & c_2 : noCollide(\mathbf{r}^J(t), \mathbf{r}^T(t), [0,5]) & \\
\Leftrightarrow & ||\mathbf{r}^J(t) - \mathbf{r}^T(t)||^2 - (a_J + a_T)^2 > 0 & \forall t \in [0,5] \\
\Leftrightarrow & ||(1-t)\mathbf{p}_0^J + t\mathbf{p}_1^J - \mathbf{p}_0^T||^2 - (a_J + a_T)^2 > 0 & \forall t \in [0,5] \\
\Leftrightarrow & ((1-t)x_0^J + tx_1^J - x_0^T))^2 + ((1-t)z_0^J + tz_1^J - z_0^T))^2 - (a_J + a_T)^2 > 0 & \forall t \in [0,5]
\end{array}$$

Constraint 3 is phrased in terms of the *inFrontOf* expression. The trajectories for both Table and Chair are of degree zero, which means that we cannot use the derivative of $\mathbf{r}^T(t)$ for the forward facing direction $\mathbf{u}$. We assume that $\mathbf{u} = (u_x, u_z)$ is a manually assigned default value.

$$\begin{array}{lll}
 & c_3 : directionRelation(\mathbf{r}^T(t), \mathbf{r}^C(t), [0,5]) & \\
\Leftrightarrow & ||\mathbf{u}||^2||\mathbf{v}||^2 cos^2(\epsilon) - (\mathbf{u}\cdot\mathbf{v})^2 < 0 & \forall t \in [0,5] \\
\Leftrightarrow & ||\mathbf{u}||^2||\mathbf{r}^C(t) - \mathbf{r}^T(t)||^2 cos^2(\epsilon) - \left(\mathbf{u}\cdot(\mathbf{r}^C(t) - \mathbf{r}^T(t))\right)^2 < 0 & \forall t \in [0,5] \\
\Leftrightarrow & (u_x^2 + u_z^2)||\mathbf{p}_0^C - \mathbf{p}_0^T||^2 cos^2(\epsilon) - \left(\mathbf{u}\cdot(\mathbf{p}_0^C - \mathbf{p}_0^T)\right)^2 < 0 & \forall t \in [0,5] \\
\Leftrightarrow & (u_x^2 + u_z^2)\left((x_0^C - x_0^T)^2 + (z_0^C - z_0^T)^2\right) cos^2(\epsilon) - \left(u_x(x_0^C - x_0^T) + u_z(z_0^C - z_0^T)\right)^2 < 0 & \forall t \in [0,5]
\end{array}$$

A *system of constraints* C is the conjunction of these constraints:

$$C \Leftrightarrow c_1 \wedge c_2 \wedge c_3$$

Simultaneous solving is required to locate values for the control points that verify all three of these equations, a technique for which is described in the following section.

6.3.2 Constraint system solving and optimization

A mechanism for solving systems of simultaneous quantified constraints using interval arithmetic is described in Chapter 5, namely the interval-based quantified constraint optimizer.

Some scene-related benchmarks evaluated in Chapter 5 cannot be solved using either a solving or an optimization technique. In this respect, the capability of the solution finding technique dictates the level of complexity of the automatically created constraint systems. In this section we examine what characteristics a constraint system should exhibit so that its solutions are found in a feasible time using the optimizer presented in Chapter 5. These characteristics influence the manner in which constraint systems are created from abstract constraints.

Solution approximations are provided for benchmarks in Chapter 5 when no solutions are found. In cases such as these, it is useful to have a measure of the quality for the approximate solution. In this section we show how a quality metric is obtained during the optimization process.

6.3.2.1 The relaxation constant as a quality metric

The interval-based quantified constraint optimizer uses the concept of relaxed constraints in searching for a solution. Each constraint c_i is relaxed by a quantity δ_i, which we call the relaxation constant. If the constraint expression is formulated correctly, then δ_i indicates the level of quality of the approximate solution.

For quality to have meaning, it must be expressed in terms of a unit that is relevant to the scene and that is common across all constraints. Distance is an example of such a measure, where the level of relaxation indicates how far (for example, in meters) an approximate solution is from an actual solution. Our formulation of spatial constraints in Section 6.3.1 is expressed in terms of distance for this reason:

- *near*: this constraint is expressed as the squared Euclidean distance between two entities (see Section 6.3.1.1). A relaxed constraint of this form replaces the zero on the right hand side with a value represented by δ:

  $$||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N + \alpha)^2 < \delta$$

  In this case, δ is the squared distance of the approximate solution from the actual solution, and $\sqrt{\delta}$ indicates the distance in whichever unit the scene is specified. The closer the value of $\sqrt{\delta}$ is to zero, the closer the approximation is to the actual solution. Therefore, quality for constraints of this type is:

  $$quality = \sqrt{\delta}$$

- *noCollide*: this constraint is also expressed as the squared Euclidean distance between two entities (see Section 6.3.1.1), and has a relaxation constant of the same formulation:

  $$||\mathbf{r}^M(t) - \mathbf{r}^N(t)||^2 - (a_M + a_N)^2 > \delta$$

  Quality for constraints of this type is also defined as:

  $$quality = \sqrt{\delta}$$

- *directionRelation*: this constraint restricts the spatial distance between a model and the direction vector of the reference model. A relaxed constraint appears as follows:

  $$||\mathbf{u}||^2||\mathbf{v}||^2 cos^2(\epsilon) - (\mathbf{u} \cdot \mathbf{v})^2 < \delta$$

  The above constraint was formulated using zero on the right hand side, which allowed multiplication throughout by the $||\mathbf{u}||^2$ value. Squared distance is expressed by $\delta/||\mathbf{u}||^2$, which means that quality is defined as:

  $$quality = \sqrt{\frac{\delta}{||\mathbf{u}||^2}}$$

Each of the above quality values measures the distance of an approximate solution from an actual solution over a time interval.

Quality for all three constraint types is measured in the same unit (distance). The quality of an approximate solution to a system of constraints is evaluated as the sum of the quality for each individual constraint in the system. The closer the total sum is to zero, the better the quality of the approximate solution.

6.3.2.2 Analysis of quality in fiction-to-animation constraint systems

We investigate the idea of quality with respect to the fiction-to-animation benchmarks used in Chapter 5, for answering the following questions:

1. *Does the quality metric reflect when an actual solution is located?*
2. *Does quality improve as the optimization progresses?*
3. *What effect does the degree of the trajectory and the dimensionality of the scene have on the quality of an approximate solution?*
4. *What effect does the number of constraints have on the quality of an approximate solution?*
5. *Is there evidence to support the hypothesis that quality reflects the correctness of behaviour in a scene?*

We investigate these questions with respect to scenes that contain static entities as well as scenes containing dynamic entities.

Quality in scenes with static entities

We use the Front, Scene, and Layout3 benchmarks to determine the relationship between quality and execution time for scenes without motion (tested in Chapter 5 and described in detail in Appendix D). We record the quality metric every ten seconds and plot it against time, limiting the total execution time to 90 minutes (5400 seconds).

The quality of intermediate approximate solutions is plotted against the execution time of the optimizer in Figure 6.11. These benchmarks are all consistent, and quality reaches zero at the same point at which a solution is located for each benchmark. The quality reflects the point at which solutions are located, that is, when quality reaches zero as is expected from the design of these constraints.

Improvement in quality with the progression of time is observed from the graph in Figure 6.11, indicating that approximate solutions get closer to actual solutions with added optimization time. The greatest improvement in quality occurs early in the optimization process for both the Front and Scene benchmarks. This shows that termination of the optimization process is possible before solutions are located, resulting in solutions of improved quality in relation to the solutions located earlier.

The solutions to the Front and Scene benchmarks are visualized in Figure 6.12, in which the forward facing directions manually assigned to each model are indicated. The visualized layout corresponds to the constraints specifying each scene.

The Collision benchmark is a constraint system specifying a scene that contains an increasing number of models, where every pair of models is constrained to be *near* and *noCollide*. If more than three models are in the scene, then the constraint system is inconsistent. We optimize constraint systems describing scenes that contain two to eight models, and record the quality at ten second intervals for each system.

The quality achieved is plotted against optimization time in Figure 6.13 for each Collision benchmark. Solutions are quickly located for the two consistent systems (containing two and three

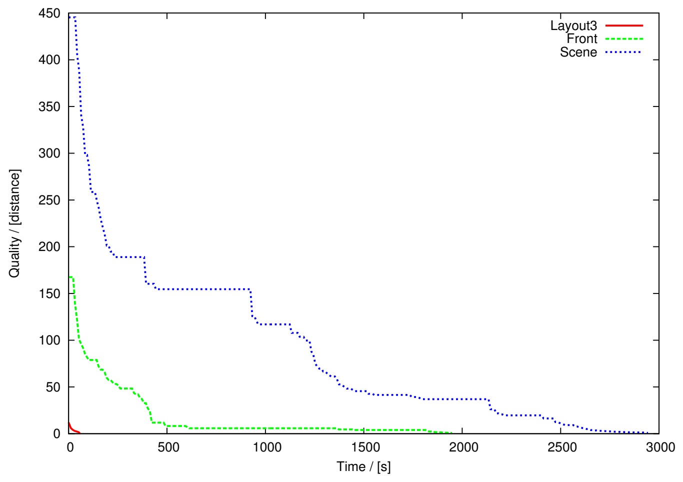
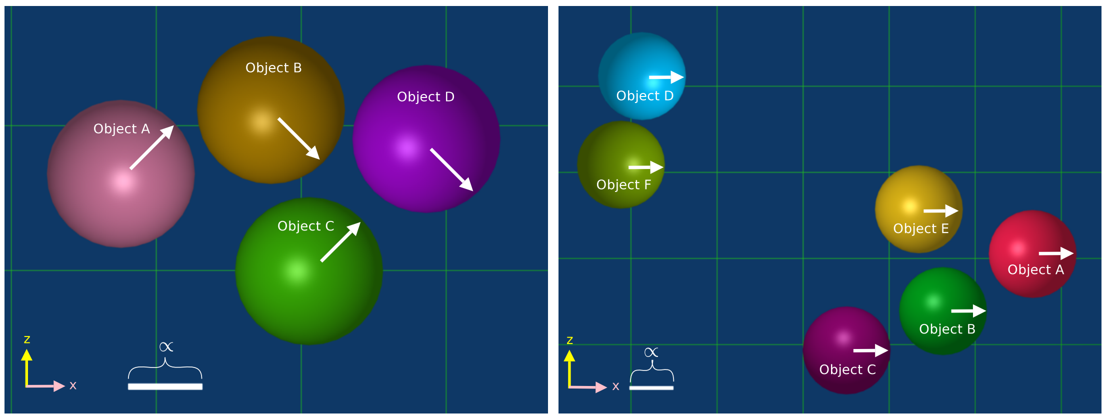


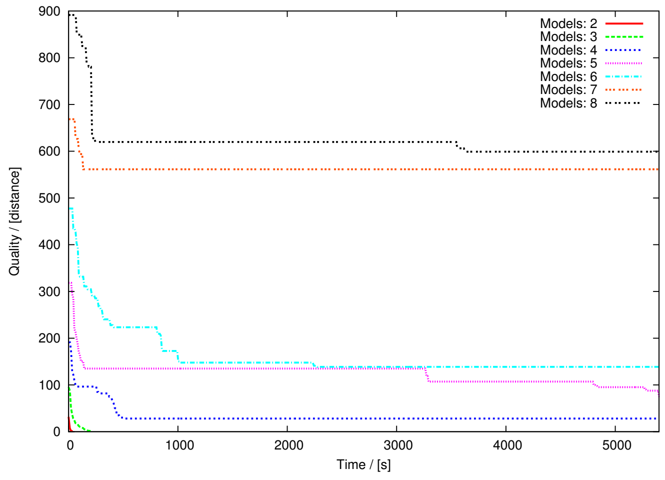
Figure 6.11: Quality of intermediate solutions to benchmarks (without motion) as a function of execution time.



(a) Front (b) Scene

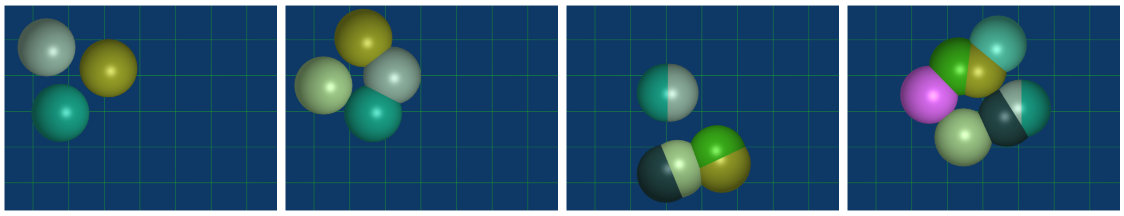
B *inFrontOf* A	B *near* A	A *inFrontOf* B	A *near* B
C *inFrontOf* B	C *near* B	C *behind* B	B *near* B
D *inFrontOf* C	D *near* C	E *toLeftOf* B	E *near* B
∗ *noCollide* ∗		F *toRightOf* D	F *near* D
		∗ *noCollide* ∗	

Figure 6.12: Solutions to Front and Scene benchmarks depicted graphically.



$$* \ near \ *$$
$$* \ noCollide \ *$$

Figure 6.13: Quality of approximate solutions to inconsistent constraint systems as a function of execution time.



(a) 3 Models (consistent): Quality = 0

(b) 4 Models: Quality = 101.23

(c) 6 Models: Quality = 138.71

(d) 8 Models: Quality = 599.04

Figure 6.14: Visualization of approximate solutions to inconsistent constraint systems.

models respectively). Only approximate solutions can be produced for inconsistent constraint systems, and the maximum improvement in quality is achieved early in the optimization process. Minor improvements are made given further execution time.

Figure 6.14 is a visualization of some of the approximate solutions found for the Collision benchmarks. More interpenetration occurs as models are added to the scene, which corresponds to worsening levels of quality for each scene. This evidence supports the hypothesis that quality reflects the visual correctness of behaviour in a scene.

Quality in scenes with dynamic entities

To determine the relationship between quality and execution time for scenes with motion, we use the Dynamic1Static1 and Dynamic2 benchmarks tested in Chapter 5 and described in detail in Appendix D. We use trajectories of increasing degree, and the constraint systems are phrased in scenes of increasing dimensionality. Every variable in these benchmarks begins with the range $[-10, 10]$. We record the quality metric every ten seconds and plot it against time, limiting the

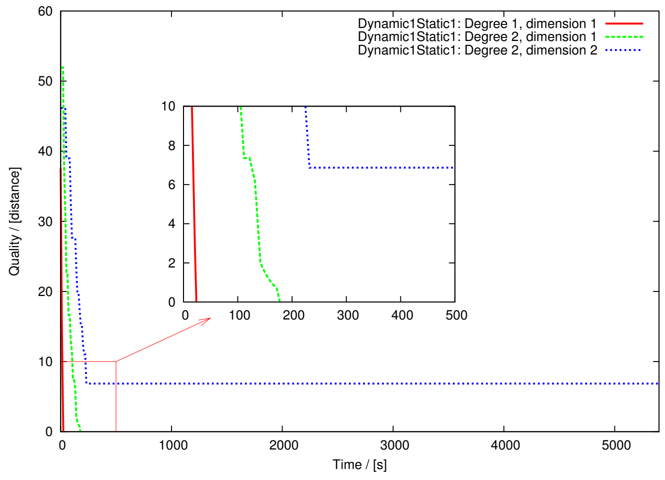
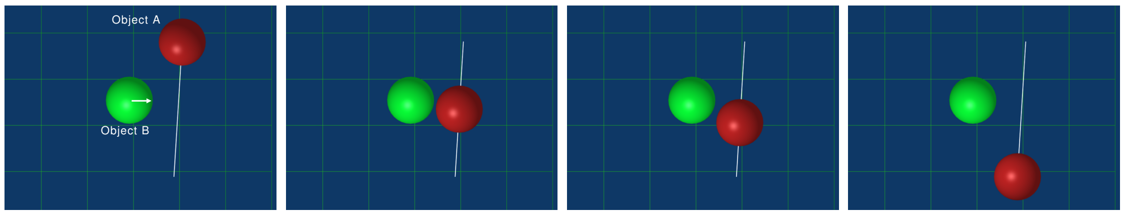


Figure 6.15: Quality of intermediate solutions to the DYNAMIC1STATIC1 benchmark as a function of execution time.



(a) 0 seconds (b) 5 seconds (c) 6 seconds (d) 10 seconds

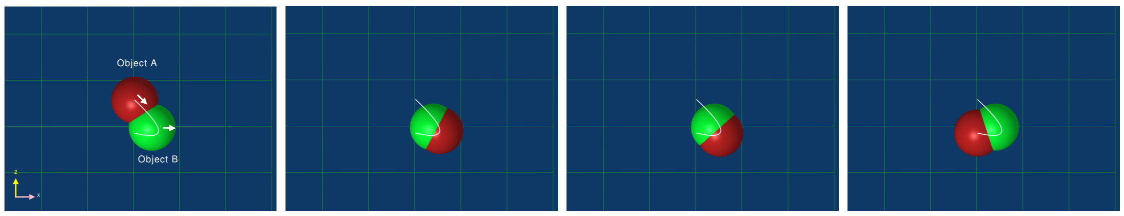
$$\texttt{A}\ \ inFrontOf\ \ \texttt{B}\ \ \forall t \in [5,6]$$
$$\texttt{A}\ \ near\ \ \texttt{B}\ \ \forall t \in [5,6]$$
$$\texttt{A}\ \ noCollide\ \ \texttt{B}\ \ \forall t \in [0,10]$$

Figure 6.16: Visualization of the solution to the DYNAMIC1STATIC1 benchmark using a trajectory of degree 1.

total execution time to 90 minutes (5400 seconds). We provide animated films that visualize the solutions found for each benchmark in Appendix F.

The recorded quality for the DYNAMIC1STATIC1 benchmark is plotted against time in Figure 6.15. No solution is located for the constraint system phrased over trajectories of degree 2 in a two-dimensional scene. This indicates that the larger the system becomes in terms of degree and dimension (and consequently the number of variables), the more difficult the solution finding process becomes. In spite of this, a high quality approximate solution is located for this benchmark early in the optimization process.

The solution of the DYNAMIC1STATIC1 benchmark using a trajectory of degree 1 is visualized in Figure 6.16. The *inFrontOf* constraint is satisfied visually at the correct point in the resulting motion. An approximate solution of this benchmark using a trajectory of degree 2 is visualized in Figure 6.17. Errors are visible in these scenes (interpenetration of the objects), reflecting the use of approximate solutions (and not actual solutions) to create these scenes. This demonstrates a correlation between quality and the visual correctness of the derived behaviour.



(a) 0 seconds (b) 5 seconds (c) 6 seconds (d) 10 seconds

$$\texttt{A}\ \textit{inFrontOf}\ \texttt{B}\ \forall t \in [5,6]$$
$$\texttt{A}\ \textit{near}\ \texttt{B}\ \forall t \in [5,6]$$
$$\texttt{A}\ \textit{noCollide}\ \texttt{B}\ \forall t \in [0,10]$$

(the forward direction of B is indicated in the first image)

Figure 6.17: Visualization of the approximate solution to the Dynamic1Static1 benchmark using a trajectory of degree 2.

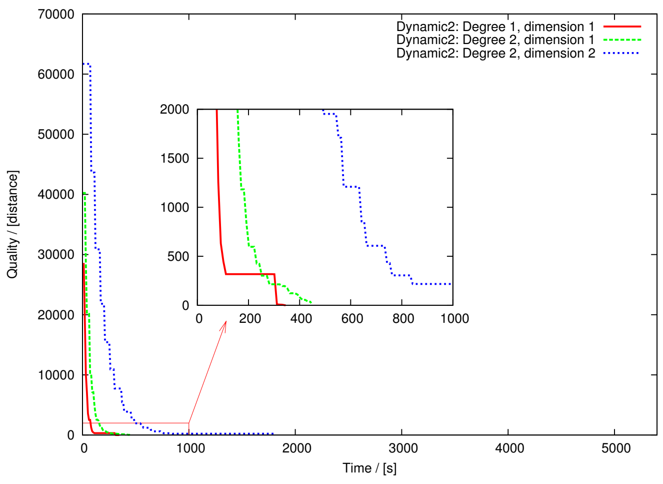
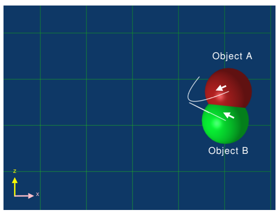


Figure 6.18: Quality of intermediate solutions to the Dynamic2 benchmark as a function of execution time.

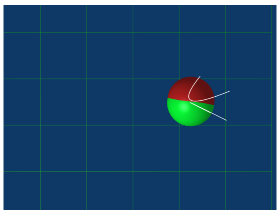
Quality is plotted against time for the Dynamic2 benchmark in Figure 6.18. The quality levels achieved for this benchmark do not come as close to zero compared to the quality achieved with the Dynamic1Static1 benchmark for trajectories of higher degree and dimension (minimum quality of 6.86 for Dynamic1Static1, 217.10 for Dynamic2). We attribute this to a greater number of variables in the Dynamic2 benchmark (more trajectories of a higher degree).

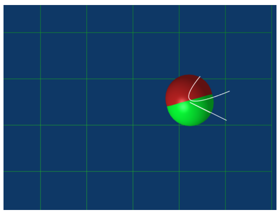
Two visualizations of the approximate solutions of differing quality are presented in Figure 6.19 for the Dynamic2 benchmark. The interpenetration is less pronounced in the visualization produced from the approximation of better quality. This demonstrates the correlation between quality and the visual correctness of the derived behaviour.

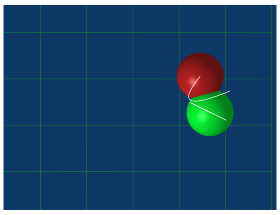
The above experiments demonstrate that the number of constraints does not directly influence the difficulty of the solution finding problem (as is the case with the number of variables). This is demonstrated by the fact that the Dynamic2 benchmarks contain fewer constraints than the



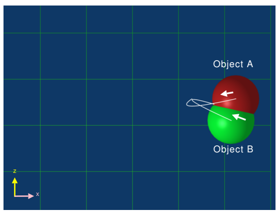
(a) 0 seconds


(b) 5 seconds

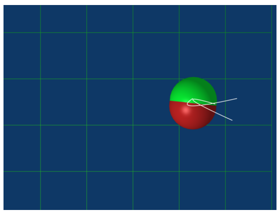

(c) 6 seconds

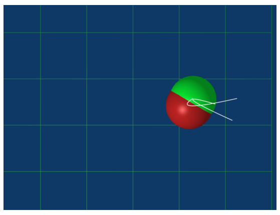

(d) 10 seconds

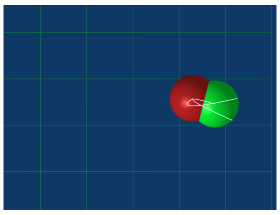
Quality = 161.32



(e) 0 seconds


(f) 5 seconds


(g) 6 seconds


(h) 10 seconds

Quality = 25.08

$$\begin{gathered}\texttt{A}\ \ inFrontOf\ \ \texttt{B}\ \ \forall t \in [5,6] \\ \texttt{A}\ \ near\ \ \texttt{B}\ \ \forall t \in [5,6] \\ \texttt{A}\ \ noCollide\ \ \texttt{B}\ \ \forall t \in [0,10]\end{gathered}$$

(the forward direction is the direction of motion of both models, indicated in the first image)

Figure 6.19: Visualization of approximate solutions of increasing quality for the DYNAMIC2 benchmark.

FRONT benchmark, but no solution is found for the former because of the greater number of variables.

Summary of findings and suggestions

The experiments detailed in this section provide evidence to support the following conclusions (with respect to the questions posed at the beginning of this section):

1. The quality metric reflects the discovery of an actual solution, that is, when a quality of zero is encountered.

2. Quality improves as the optimization progresses.

3. Quality reflects the correctness of visual behaviour in a scene.

4. The greater the number of variables (as a result of higher degree curves or scenes of higher dimensionality) in a constraint system, the longer the time required for locating a solution using the optimizer.

5. The number of constraints in a system does not impact the solution finding process as does the number of variables.

Conclusions 1, 2, and 3 support the use of the quality metric for evaluating behaviour automatically quantified for a scene, especially when the optimization process is terminated prematurely due to lack of time. Conclusions 4 and 5 influence the manner in which we convert abstract constraints into constraint systems for solving. Based on these conclusions we formulate the following guidelines:

- Trajectories of degree no higher than 2 should be used, to maximize the likelihood of finding actual solutions. Otherwise, we expect to terminate the optimization process prematurely and make use of an approximate solution.
- Scenes must be specified in a maximum of two dimensions, otherwise an approximate solution is expected.
- No restriction is placed on the number of constraints in a system.

These guidelines disqualify potential constraint systems that are automatically created from abstract constraints. For example, a model that moves back and forth and between other objects requires a trajectory with a very high degree curve. It is unlikely that the constraint optimization process will find a solution to a constraint system over this trajectory within a feasible time-limit.

The following section describes methods for overcoming these shortcomings of the optimization process, while still enabling complex scene behaviour.

6.3.3 Constraint system formulation for feasible optimization

Section 6.3.2 indicates that constraint systems with a large number of variables require a large amount of optimization time to locate solutions of good quality. We reduce the optimization time by expressing motion as sequences of low degree curves, and solving constraint systems in an incremental fashion.

6.3.3.1 Sequences of low degree trajectories

We define *complex behaviour* to be entity motion that includes a number of way-points in a scene. For example, the story *"The man entered the room and approached the table. .... He then moved towards the chair."* describes complex behaviour. For illustration, we assume that this story is converted into the following abstract constraints:

`man` *noCollide* `room` $\forall t \in [0, 5]$
`man` *inside* `room` $\forall t \in [5, 30]$
`man` *near* `table` $\forall t \in [9, 14]$
`man` *near* `chair` $\forall t \in [14, 30]$
`man` *noCollide* `{table;chair}` $\forall t \in [5, 30]$
`table` *noCollide* `chair` $\forall t \in [0, 30]$

A trajectory of a high degree must be attributed to the "man" object to ensure that the model reaches all the way-points at the correct time. As observed in Section 6.3.2, the higher the degree the more difficult the constraint optimization process becomes.

We avoid using trajectories of high degree by constructing a trajectory as a chain of low degree curves (Christie et al., 2002). We find that there is a satisfactory trade-off between optimization performance and solution quality using chains of first degree curves. We segment trajectories by

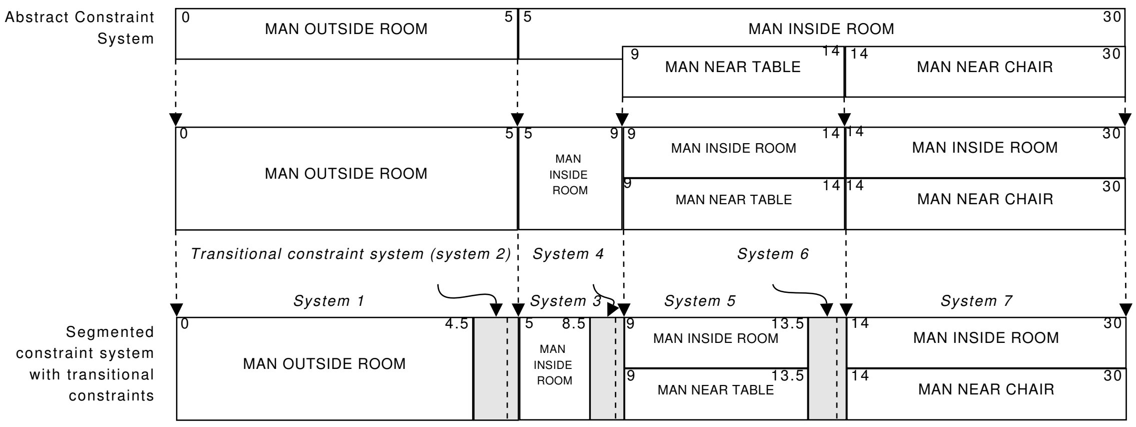
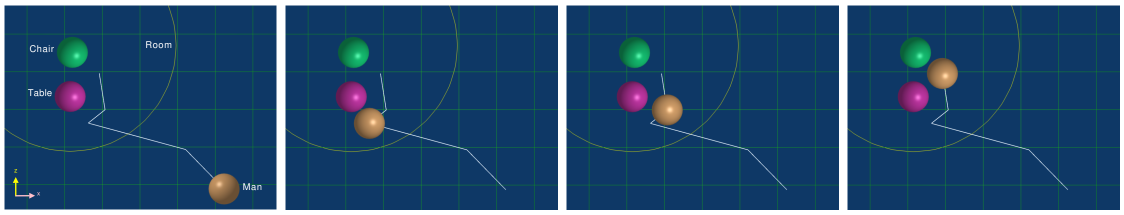


Figure 6.20: Illustration of constraint system segmentation.



(a) 5 seconds (b) 8 seconds (c) 12 seconds (d) 18 seconds

Figure 6.21: Visualization of a complex trajectory as the sequence of low degree trajectories.

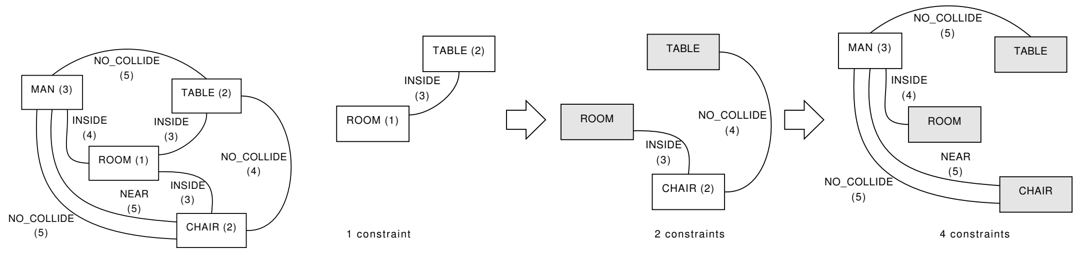
identifying contiguous segments of time in which the behaviour of an entity is described by a single low-degree trajectory.

A graphical illustration of a subset of the abstract constraints above is presented in Figure 6.20, as well as an illustration of how the constraints are segmented into a chain of 7 constraint systems. The duration of the scene is divided into contiguous time intervals during which no new constraints apply. For each interval a single set of constraints is applicable over the full duration of that interval. A *transitional* system is inserted between adjacent intervals allowing a period during which both sets of constraints must be satisfied so that model locations blend smoothly from one interval to the next.

To facilitate efficient constraint solving, the ending locations for each interval are used as starting locations for solving the following constraint system. If the solution renders the subsequent constraint system inconsistent, then we use backtracking to locate alternate solutions to the previous constraint system.

Figure 6.21 presents a visualization of a final solution found by the optimizer for the set of segmented constraint systems derived in Figure 6.20. A solution is located for every segmented constraint system, meaning that quality value zero is obtained in all cases.

The segmentation method presented in this section is a compromise between context-free behaviour quantification and optimization performance. In Chapter 5 we argue that an analytical approach to constraint solving provides a non-incremental method for quantifying scene behaviour, whereas the idea of segmentation is inherently incremental. However, the constraint optimization



(a) Constraint system illustrated as a weighted graph.

(b) Three incrementally solved constraint systems

Figure 6.22: Illustration of incremental system optimization.

process occurs once for an entire scene. Once a solution is found for all the constraint systems, the context-free advantage is maintained.

6.3.3.2 Incremental system optimization

Some constraint systems contain a large number of variables despite the use of trajectory chaining. To improve optimization efficiency, we solve systems in an incremental fashion by finding solutions to only a subset of the constraints at a time.

Incremental system optimization is performed by placing static objects first, and thereafter quantifying the motion for dynamic objects. Each entity is assigned a weight based on its trajectory type, as illustrated in Figure 6.22(a). The setting (room) is assigned weight of 1 (because settings are assumed to be located at the origin), while static objects are assigned weight of 2. Dynamic objects (man) are assigned a weight of 3.

Each edge in the graph shown in Figure 6.22(a) represents a constraint, and is assigned a weight calculated as the sum of the involved entities' weights. The edge with the lowest weight is selected as the first constraint to be solved, as illustrated in Figure 6.22(b), and the edge is removed from the graph. In the example, the solution to the first constraint defines the location of "table" for the remaining constraint systems, removing its variables from any subsequent optimization process.

The process continues iteratively, selecting the edge with the lowest weight as the next constraint to be optimized until there are no more edges in the graph. At each iteration a single edge is selected that identifies the two entities for which trajectories will be quantified in the current iteration. To avoid inconsistency, any other constraint involving both entities is also added to the constraint system to be optimized at the current iteration. This process is illustrated in Figure 6.22(b).

This method is a compromise between context-free and incremental behaviour quantification. We find that each constraint system produced using the incremental method is solved rapidly using the optimizer. Once all constraints are solved, the overall solution is one that can be used to query the scene behaviour in a context-free manner.

6.3.3.3 Combination of segmentation and incremental optimization

The restrictions imposed on the complexity of constrain systems by the interval-based quantified constraint optimizer are overcome using constraint trajectory chaining and incremental optimization. A set of abstract constraints is first segmented into a chain of constraint systems as described

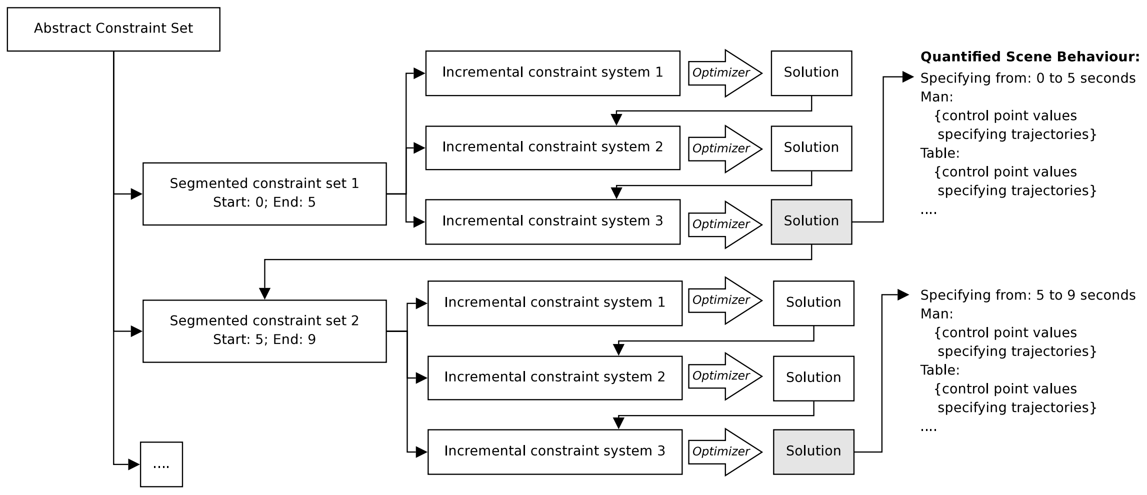

Abstract Constraint Set
Segmented constraint set 1
Start: 0; End: 5
Incremental constraint system 1
Incremental constraint system 2
Incremental constraint system 3
Optimizer
Solution
Segmented constraint set 2
Start: 5; End: 9
....
Quantified Scene Behaviour:
Specifying from: 0 to 5 seconds
Man:
{control point values specifying trajectories}
Table:
{control point values specifying trajectories}
....
Specifying from: 5 to 9 seconds
Man:
{control point values specifying trajectories}
Table:
{control point values specifying trajectories}
....

Figure 6.23: Illustration of the process used to create quantified trajectories from an abstract constraint set.

in Section 6.3.3.1 and then each segmented constraint system is solved in an incremental fashion as described in Section 6.3.3.2. This process is illustrated in Figure 6.23.

In summary, this section demonstrates that the interval-based constraint optimizer is limited in its ability to find solutions to scene-related constraint systems. We overcome these limitations by formulating and optimizing constraint systems in a careful manner, allowing us to exploit the strengths of the analytical formulation of behaviour in a scene while maintaining the ability to quantify complex behaviour in a virtual environment.

6.4 Population of animated, multi-modal 3D environments

This section describes the strategies employed for configuring a three dimensional environment using scene descriptions derived from the annotations. We investigate strategies for instantiating virtual environments and combining audio and text-based modalities into a final multi-modal presentation.

6.4.1 Strategies for instantiating 3D virtual environments

The population of a 3D environment for each identified scene involves instantiating models in the environment, placing or animating these models according to their quantified behaviour, and instantiating background scenery in the environment.

6.4.1.1 Object models

Each entity that appears in a scene (identified using the process described in Section 6.2.2) is represented visually in the 3D virtual environment using the model specified in the entity descriptor.

Models sourced from the library are standardized with respect to size and orientation, ensuring that all objects are scaled to fit within the unit cube (using automatic normalization). All objects

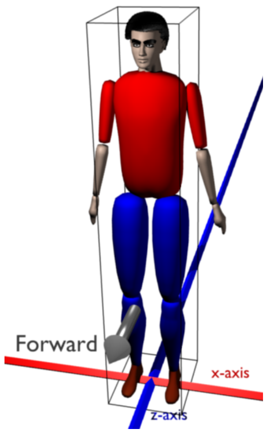


Figure 6.24: Default humanoid model from the library with orientation, location and bounding box illustrated.

are oriented to face the positive z-axis, with the up direction facing the positive y-axis. Future orientations of a model are calculated based on this fact.

Each model in the library is associated with a manually defined bounding box from which a radius is derived for specifying a bounding sphere in the creation of quantified constraints. This bounding box need not be the hull of the model, because cases exist where the hull produces an exaggerated radius. For example, the humanoid model illustrated in Figure 6.24 is associated with a bounding box that is not the hull of the model, but rather approximates the *body* of the humanoid because the arms exaggerate the width of the model.

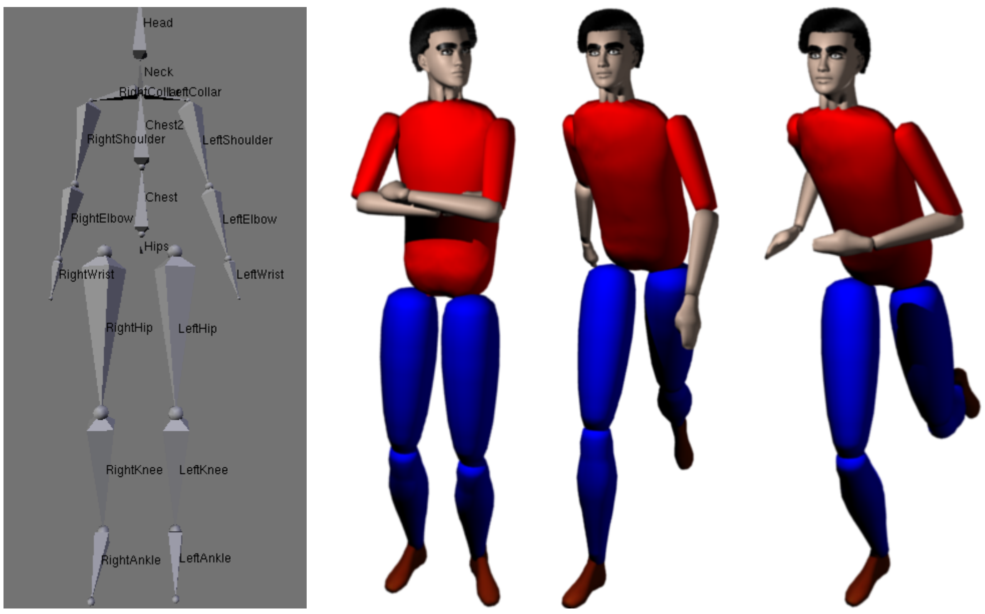
All objects in the library are pre-textured. The current implementation of the library contains only a limited number of humanoid models, and we use variations in texturing for visual distinctiveness between avatars. Vertices comprising the model are grouped in terms of items of clothing, including *shirt* and *trousers*. The selection of different materials for these vertex groups provides a simple method for modifying the visual appearance of unique avatars. Descriptors contain the individual colour and scaling factor for each entity. At present, clothing colour is chosen at random for each avatar, as is the scaling factor.

6.4.1.2 Model positioning and articulation

Solutions to the quantified constraint systems specify trajectories for every entity, and placement in a virtual environment consists of ensuring that the models follow the paths dictated by the corresponding trajectories.

All models face the positive z-axis by default, and currently no constraints are implemented that specify the orientation of models in a scene. Orientations for dynamic models are derived using the tangent of the trajectory at each frame in the scene, automatically ensuring that these models face the direction of motion.

Local articulation is applied to human models in the form of motion capture for a set of three typical human poses. This requires the following data to be associated with the models extracted from the library:



(a) Rest position (b) Stand pose (c) Walk pose (d) Run pose

Figure 6.25: Armature configuration and illustration of the default avatar in the three standard poses.

- **Skeleton:** models are rigged with an appropriate armature, labeled in a standardized manner so that multiple motion capture configurations can be applied. The standard bone structure and labeling system we use is illustrated in Figure 6.25. Currently, each bone is assigned to a vertex group on the avatar mesh. If no skeleton is assigned to a model, then no articulation is applied to the model in the scene.

- **Motion Capture Data:** Currently, the library of motion capture data contains only three poses for humanoid models, namely *stand*, *walk,* and *run*, illustrated in Figure 6.25. These are listed as optional motion capture assignments in the description of a model in the model library. The selection of an appropriate pose for an avatar at a point in the scene is based on the velocity of a model at that point. If the velocity is above an upper threshold, the *run* pose is selected. If velocity is below a lower threshold, then the *stand* pose is selected. Otherwise the *walk* pose is selected.

The addition of the three basic poses for humanoid models greatly improves the visual richness of the final scene, enhancing the recognition of the humanoid models as representations of people. The final result is a scene containing visual objects and articulated avatars, an example of which is provided in Figure 6.26. The trajectory of each entity is represented by a coloured curve.

6.4.1.3 Setting

The Setting annotation is used to construct surrounding geometry in the 3D environment. The challenge with interpreting the Setting annotation is that only a single token is annotated (future annotation categories potentially identify descriptive phrases regarding the setting).

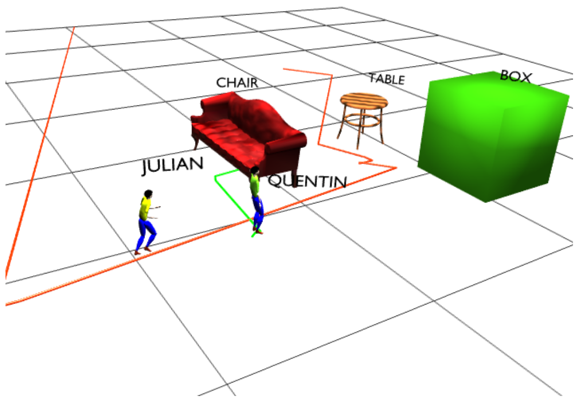


Figure 6.26: Illustration of a 3D environment containing models with assigned motion and articulation.

We use procedural modeling approaches for creating background scenery because they are able to construct geometry of unlimited size and scale. This is opposed to a predefined model that is not large or detailed enough for every scene. Procedural methods also create geometry that appears unique at each invocation, allowing for variation in the appearance of scenery between different settings.

Currently, we provide a procedural method for three categories of setting, namely *terrain*, *room,* and *city*. The annotated Setting token must be associated with one of these categories for the invocation of the appropriate procedure.

WordNet (Fellbaum, 1998) provides a mechanism for categorizing Setting annotations into one of the three groups. If a Setting annotation is a hyponym of the term "geological formation" or "geological area", it is classified as a *terrain*. Any term with the hypernym "urban area" is classified as a *city*, while any term with the hypernym "room" is classified as a *room*. For example, the word "study" is a hyponym of "room", while the terms "London" and "city" are hyponyms of "urban area" and are classified as *cities*.

Entity positioning and locomotion is determined before the creation of the setting geometry, and so a procedural method must adapt to the behaviour of entities in a scene.

Procedural terrain generation Procedural terrain generation is inspired by the work of Musgrave et al. (1989) who pioneered this technique using fractional Brownian motion and erosion simulation techniques. Recent work by Belhadj (2007) provides alternative methods for the procedural terrain generation that allows for the pre-definition of ridge and peak points, around which terrain is constructed automatically. This allows terrain to be created at the correct height to "support" the entities within the scene wherever they move, while the rest of the terrain is constructed in a natural-looking manner.

At present, terrain geometry is textured with a "grass" material, but continued research in this area is underway in terms of populating a terrain with rivers and flora, and providing high levels of detail for close-up shots[1].

[1]See the research page on this topic available at `http://delta.ru.ac.za/vrsig/currentprojects/058proceduralterrain/index.html`

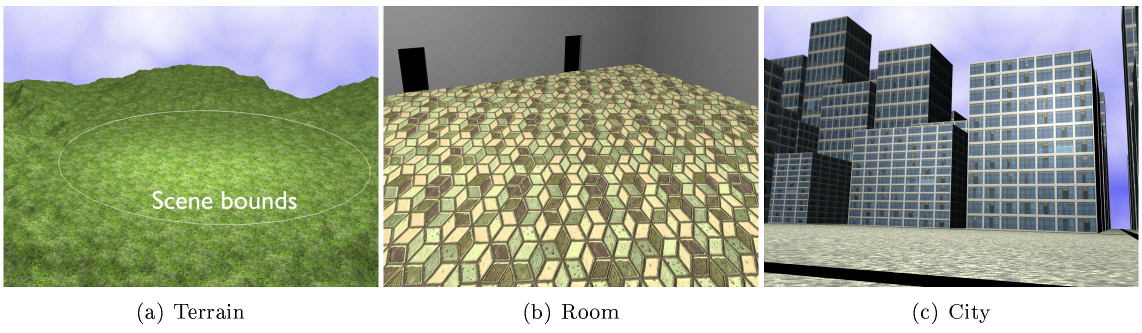


Figure 6.27: Example settings generated procedurally.

Procedural city generation Procedural city generation is inspired by the work of Parish and Muller (2001) and Muller et al. (2006), who describe methods for the automatic generation of urban road networks and buildings. We provide a city generation method that produces only a grid-iron road-network, representing buildings as textured cubes. The procedure ensures that no buildings are placed within the radius specifying the dimensions of the scene, so that the entities appear to fall within a square in the centre of a city. The simple geometry produced by this module provides visual indication of an urban environment, and continued research is underway in terms of the creation of autonomous agents for simulating pedestrians. This potentially adds visual richness to a procedurally generated city.

Procedural room generation A procedure for creating scenery indicating the inside of a room is constructed by creating geometry for four walls that surround the radius of the scene. Textures for the walls and floor are chosen at random from a library. Entities enter or leave a room through doors that are created wherever an avatar crosses the boundary of the scene. This approach functions well in most cases because we observe that few transitions occur within a single scene. In future, constraints could apply to the trajectory of a model that restrict entry and exit to a single point.

Future enhancements to procedural room generation include inserting models into the scene that typically would occur in a specific type of room. For instance, a “kitchen” would generally contain a table and some chairs, while a “bedroom” might contain a bed and a cupboard. This requires further research into the use of databases that provide this type of link between terms (Sproat, 2001), although such methods would not be considered knowledge-poor.

Visual examples of the three types of procedural setting for the 3D visualization system are provided in Figure 6.27.

6.4.2 Strategies for audio

Two options for creating audio representations of fiction text are investigated. The first option generates an audio version of the narrative using a text-to-speech synthesizer, while the second option includes sound effects, or foleys, described in the text.

He gave an excited yelp and rushed full-tilt at the surprised rabbit.

Figure 6.28: Example foley description from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

6.4.2.1 Audio narrations

Audio versions of text are generated using standard text-to-speech technology, such as eSpeak[2]. However, we observe that the speech generated is often monotonous and difficult to follow. While improvement in the prosody of the speech synthesizer can address this to an extent, we believe that this problem is alleviated by choosing voices appropriately for each avatar in the story (Zhang et al., 2003). This adds an element of variation to the audio narrative.

Fiction text contains quoted text representing speech emitted by different characters in the story. This information is provided in the form of Quote annotations in the fiction text, as identified in Chapter 4. Each unique avatar identified in the book requires a different voice. Most speech synthesizers provide a number of voices, usually including a male and female voice. While entirely new voices can be created for a speech-synthesizer, experience has shown this to be a tedious and time consuming process (Hood, 2004). Instead, a range of voices is created by choosing a different pitch for each avatar, within a range that is still appropriate to the specified gender. Each section of quoted speech is rendered with the appropriate voice, and recorded to an audio file. Non-quoted speech is rendered using a default *narrator* voice.

The use of avatar specific voices adds valuable variation to the narrated output, and allows avatars to be identified by the listener even in dialogue where they are not explicitly named.

6.4.2.2 Foleys

Sound effects are often described in fiction text, as illustrated in Figure 6.28. These descriptions are potentially identified using annotations created by the hierarchical rule-based learning system in Chapter 4. However, we currently create foley annotations using a specially designed knowledge-poor method.

We identify nouns such as "yelp" automatically by examining the hypernym (abstraction) tree of the word provided by WordNet (Fellbaum, 1998). Any word related to "sound" is identified as a foley, and a corresponding audio file matching the word is located from an audio library. We use a foley library structured in the same manner as the model library, where each foley is associated with a number of keywords. A query to the library containing the triggering word is compared against foley keywords, returning foleys where positive matches occur.

Sound effects further improve the quality of the overall presentation by adding richness to the audio output.

6.4.3 Strategies for subtitles

We minimize the problem of mispronunciation in synthesized speech by providing concurrent subtitles. The text for each segment is rendered as an image that is superimposed over the currently

[2]eSpeak: `http://espeak.sourceforge.net/` [accessed on 10 July 2008]

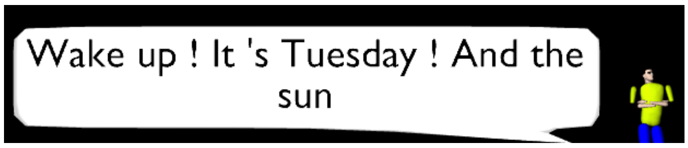


Figure 6.29: Example subtitle with speech bubble, representing a quote from talking avatar Julian.

displayed graphics. The times at which subtitles appear are automatically derived from the location of the corresponding audio files in the presentation time-line.

We enhance subtitles by wrapping the text in a speech-bubble whenever quoted speech is encountered, an example of which is presented in Figure 6.29. A graphical representation of the appropriate avatar provides a visual identifier unique to the talking character. The use of a visual avatar simplifies the identification of the speaking entity, and provides a visual mechanism for differentiating between narration and speech.

The visual component of the subtitles could also serve to benefit viewers with hearing deficiencies.

6.4.4 Construction of a multi-modal animated film

Virtual environments are instantiated in the Blender modeling package [3], which is an open source 3D modeling and animation tool that contains a Python scripting interface as well as a video sequencing editor. The scripting interface allows for the automatic conversion of annotations and quantified trajectories into 3D environments. The advantage of using a package such as Blender is the availability of a number of pre-implemented manipulation tools for computer graphics that are used to automate the scene creation process.

Positioning and motion within a 3D environment is controlled using Blender's interpolation (IPO) curve structure (Blender Foundation, 2008), which defines a model's translation in a scene in terms of an independent curve (channel) for each dimension. Each channel corresponds to a component of an entity's trajectory, and each curve is constructed by tracing the quantified trajectories for each entity.

Blender allows for the creation of multiple 3D environments, conveniently called *scenes*, matching our definition of the concept in Section 5.1 on page 106 in Chapter 5. Every scene is a separate 3D environment in Blender, from which segments are rendered and placed into the video sequencer along with subtitles and audio. Timings in the 3D scenes are all based on the audio rendering of the fiction text. Transitions between different scenes are realized using a basic "cut" technique.

A example sequence produced automatically from annotated fiction text is presented in Figure 6.30. Rendering the final sequence results in a multi-modal animated 3D film corresponding to the annotated fiction text.

[3]Blender: `http://www.blender.org/` [accessed on 12 July 2008]

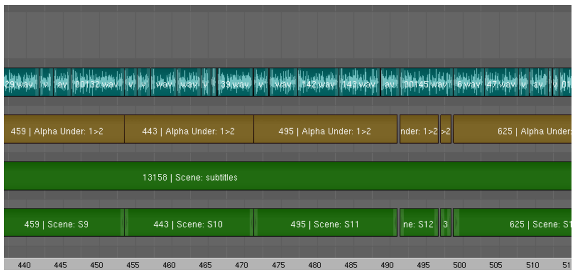


Figure 6.30: Example sequence in Blender constructed automatically from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

6.5 Analysis of the scene creation process

We investigate whether corresponding multi-modal animated 3D virtual environments and films are created using the processes described in this chapter. Evidence validating the creation process is supported by answers to the following questions:

1. *Are consistent high-level scene descriptions derived from annotated text?*
   This question determines whether consistent high-level descriptions of the scenes, their contents and behaviour are created using the automated processes we describe.

2. *Are virtual environments populated consistently using high-level scene descriptions?*
   Once a virtual environment is populated a human has the opportunity to directly modify a scene in a 3D modeling package. We report on the type and quantity of modifications made by a human as an indication of the consistency with which these environments are populated.

3. *Are the automatically populated 3D environments representative of the corresponding text?*
   This question cannot be answered using quantified methods, due to the subjectivity of language and visual interpretation. However, we provide examples of automatically created environments, and provide a subjective evaluation of each, which is guided in terms of the following questions:

   (a) *Is behaviour specified by annotations reflected in the visual representation?*

   (b) *Can complex behaviour (multiple way-points in a scene) be reflected in a visual representation?*

   (c) *Does the process support sequencing of multiple scenes?*

   (d) *Is appropriate media generated for representing the text?*

   (e) *Are the conversion processes applicable to fiction books of different type (author, series, readability)?*

This section presents a suite of experiments for answering the above questions, and describes test cases used. Metrics for measuring success are defined, and possible sources of experimental error are identified.

				Readability	
	Description	Series: Book	Author (year)	Fog	Flesh
1	Cow scene	*Famous Five 1: Five of a Treasure Island*	Enid Blyton (1942)	5.2	96.2
2	Rabbit scene	*Famous Five 1: Five of a Treasure Island*	Enid Blyton (1942)	5.2	96.2
3	Study scene	*Famous Five 1: Five of a Treasure Island*	Enid Blyton (1942)	5.2	96.2
4	Travel sequence	*Famous Five 1: Five of a Treasure Island*	Enid Blyton (1942)	5.2	96.2
5	Follow scene	*World of Tiers 7: More than Fire*	Philip Jose Farmer (1993)	6.6	87.8
6	House sequence	*Chronicles of Narnia 2: The Lion, the Witch and the Wardrobe*	C.S. Lewis (1950)	6.9	90.5

Table 6.2: Descriptions of the source books and chosen scenes.

6.5.1 Test corpus

We use six extracts from three fiction books of different type (according to the definition in Chapter 4, where books from different series and authors are considered to be of different type), listed in Table 6.2. We present the actual extracts with the visual results in Section 6.5.4.

Each extract is annotated using categories described in Chapter 4, namely with Avatar, Object, Setting, Transition, and Relation annotations. The annotations are assumed to be correct according to one human's perception, but need not be correct according to another's, nor are they guaranteed to be consistent.

The suite of extracts is chosen to illustrate features of the conversion process:

- Extracts 1, 2, 3, and 5 demonstrate the automatic quantification of behaviour in a virtual world.
- Extract 4 and 6 are extended extracts describing a number of different scenes. We use these to demonstrate the creation of corresponding sequenced animated films with multiple scenes.
- All the extracts are used to demonstrate the automatic insertion and placement of appropriate geometry and audio material.

The objective evaluation of the above processes is a poorly defined problem, which we discuss in the next section.

6.5.2 Metrics

The measurement of the success of an automatically generated virtual world is a subjective process because of different human interpretations of fiction text. The results produced by the scene creation processes are visual in nature, and it is unclear which features in a visual scene should be measured for quantitative evaluation.

The problem of evaluating automatically generated scenes is encountered in other research in the text-to-graphics domain. Most related research performs subjective evaluations through the use of visual examples (Coyne and Sproat, 2001; Lu and Zhang, 2002; Zeng et al., 2003; Joshi et al., 2004). In these cases, a small set of example images produced from the original text is provided, leaving evaluation to the discretion of the reader.

The only other evaluation method we encounter in related text-to-graphics research is user evaluation, where a group of human subjects is asked to rate the visual content produced by the automatic system (Johansson et al., 2005; Ma, 2006). However, Johansson et al. (2005) acknowledge that such studies do not provide a complete reflection on the capabilities of a system, and results vary greatly from one human to the next. We believe that this is because such studies do not objectively evaluate the system. Rather, they evaluate both the human as well as the system. Subjectivity is involved in comprehending natural language and visual images. The processes we describe are designed to improve in quality given additional time and patience of the human. These factors introduce errors into a user-evaluation that are not necessarily caused by the automated process, and are therefore not measurable.

We perform our evaluations using a subjective (but quantitative) evaluation of the manual interventions needed to ensure consistency in automatically generated content. The degree to which automatically produced content matches the text is justified through the use of visual examples.

6.5.2.1 Consistency of automatically generated content

We evaluate whether the content generated at each step in the conversion process requires modification and consider content to be consistent if the total amount of generated content exceeds the amount of manual modifications.

We calculate the following metric to determine the level of consistency achieved in the creation of high-level scene descriptions:

$$consistency = \frac{total\,content\,items - manual\,modifications}{total\,content\,items} * 100$$

The above metric, while providing numeric values, is subjective in nature because the manual modifications are performed at the discretion of an individual human.

We evaluate the consistency of automatically produced virtual environments according to the type of intervention performed. We categorize manual intervention as *deletions*, *modifications,* and *insertions,* each of which vary in terms of manual effort. Insertions are considered the most difficult operation because they require the most effort and expertise in a 3D modeling environment. Modifications are considered less difficult because only attributes of the scene are modified, requiring an intermediate level of expertise and effort. Deletions are considered the least difficult operation because little expertise is required for this operation. We present the number of each type of manual operation as an indicator of the degree of consistency.

6.5.2.2 Correctness of visual content

We follow the methods used by WordsEye (Coyne and Sproat, 2001), Swan (Lu and Zhang, 2002), and other text-to-graphics systems (Zeng et al., 2003; Joshi et al., 2004) in using visual examples as evidence regarding the correctness of the automated content. We critically evaluate each visual example. However, we maintain that the true level of correctness varies from human to human, or according to the adage is “in the eye of the beholder”.

6.5.3 Sources of experimental error

As in Chapter 4, we acknowledge the complexity of the English language and its understanding as a source of experimental error. The manual annotations produced for the corpus of test data cannot be guaranteed to be consistent or correct, and this potentially produces extraneous or surprising artifacts in the automatically generated scenes.

The subjectivity of image and film comprehension is also a source of experimental error. Similar to language comprehension, comprehension of the visual and audio modalities is defined by human experience. As such, manual modifications and critical evaluations provided for each result are independent to the evaluator, and need not correlate with another human's opinion.

Our implementation is restricted to particular visual features that avoid potentially biasing the perceived success of the automation process. We provide an indication of the extent to which certain features are implemented:

- The set of annotation categories we use are restricted to Avatar, Object, Setting, Transition, and Relation. Further visual descriptions including grasping of objects, poses, expressions, and colours are not yet implemented. These factors are not considered during the critical evaluations of the visual results.
- Following the knowledge-poor theme of this research, certain semantic types associated with annotations such as Transition and Relation are interpreted literally. For instance, the relation type *inside* is interpreted literally in examples such as "in his chair" and "in bed". These examples require detailed world-knowledge for correct interpretation, and we leave this to the human.
- Camera control is not considered during the scene generation process, leaving this to future work which will take into account the work already performed in this field (Drucker and Zeltzer, 1994; He et al., 1996; Christie et al., 2002; Nieuwenhuisen and Overmars, 2003).
- We produce graphical visualizations that are of intermediate quality in terms of richness, so that the important aspects of the visualizations are not obfuscated. We avoid the use of special effects and other graphical enhancements in these evaluations.

6.5.4 Results

We present experiments that investigate the questions posed at the beginning of this section.

6.5.4.1 Consistency of high-level scene descriptions

We investigate the consistency with which high-level scene descriptions are created from annotations using the process described in Section 6.2. In particular, we investigate the following questions:

- *Is the majority of the scene descriptions correctly created from interpreted annotations?*
- *What is the nature of manual intervention in this process?*

Extract	Scene segmentation			Model descriptors			Co-reference			Abstract constraints		
	Total	Modified	Consistency	Total	Modified	Consistency	Total	Modified	Consistency	Total	Modified	Consistency
1	1	0	100%	12	1	91.67%	12	2	83.33%	68	1	98.52%
2	1	0	100%	8	2	75.0%	52	15	71.5%	41	1	97.56%
3	1	1	0%	6	0	100%	40	8	80.0%	45	0	100%
4	6	2	66.67%	10	0	100%	23	0	100%	41	9	78.0%
5	1	0	100%	2	0	100%	11	0	100%	4	0	100%
6	2	0	100%	8	0	100%	28	1	96.43%	44	0	100%
Summary	12	3	83.33%	46	3	93.47%	166	26	84.33%	243	11	95.47%

Table 6.3: Degree of consistency for the automatic abstract constraint creation process over six extracts.

We investigate these questions using each of the six annotated extracts. We record the amount of automatically generated content and compare this with the amount of content that is modified by a human.

The type and degree of manual intervention with respect to each step of the abstract constraint creation process is listed in Table 6.3. Exact descriptions of the modifications are provided in Appendix E.

Scene segmentation is consistent in the majority of cases. Extracts in which modifications are made do not explicitly mention the scene, and human discretion is used to infer the setting.

The creation of model descriptors is consistent in the majority of cases, and only 3 of the 46 automatically created descriptors are modified manually. The majority of these modifications are concerned with assigning a trajectory of higher degree to objects that are defined as static by default.

Co-reference resolution is achieved at a high level of accuracy. Two scenarios require manual intervention. The pronoun "it" is not handled by the current implementation, and must be manually resolved. General co-references, for example "boy", are also not handled, and must be manually matched to the correct avatar is some cases. However, these scenarios represent only a small portion of the total consistent co-reference items created.

Only a small number of the automatically generated abstract constraints require manual modification. The greatest number of modifications are made for extract 6. In this case, constraints are added manually to cater for details not explicitly described in the fiction text. For example, in extract 6 the character Anne is described as "sleeping", an activity that a human associates with a "bed". This item of furniture is not explicitly stated in the extract, and no corresponding annotation is created that identifies it. A human manually inserts an abstract constraint: "Anne *inside* bed". This results in the automatic placement of a bed model in the scene. In spite of these manual modifications, the majority of automatically produced constraints are consistent.

Scene descriptions are created consistently using the knowledge-poor interpretation processes described in Section 6.2. The types of manual intervention cater for exceptions that require additional world-knowledge from a human.

	Insertion	Modification	Deletion
1	0	1	3
2	0	1	0
3	0	1	1
4	0	8	1
5	0	1	0
6	0	1	2
Summary	0	13	7

Table 6.4: Summary of manual modifications to automatically populated virtual environments.

6.5.4.2 Consistency of virtual environment population

We investigate the consistency with which the population of virtual environments is achieved using the process described in Section 6.4. We investigate the following questions:

- *Do instantiated and populated virtual environments require significant manual modification?*
- *What is the nature of manual intervention in this process?*

We investigate these questions by populating virtual environments for each of the six annotated extracts. We indicate the number and type of each manual intervention in the virtual environment.

The extent of human intervention in creating virtual environments for each extract is listed in Table 6.4. No insertions are made to any of the virtual environments, but some modifications and deletions are performed. This indicates that our process is capable of producing a minimum amount of visual content for a scene, but in some cases it produces more visual content than is necessary.

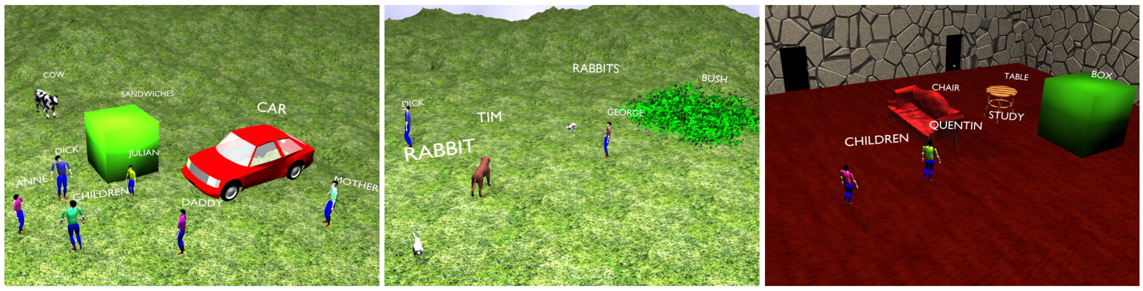
The majority of the modifications made to the virtual environments are concerned with camera placement, a function not catered for by the current implementation. The other modifications are discretionary, such as modifying the material of entities in the scene.

The primary cause for deletions in a virtual environment is the creation of content that should not be visually represented. In all these cases, extraneous content is the result of annotations that are inconsistent. In this respect, the visualization process is correctly representing the annotated text, but is not creating a correct visualization according to one human's opinion. This point is illustrated by the inclusion of a model for "children" in the scene shown in Figure 6.31(a) and (c). This entity is marked as an avatar in the extract, but it does not make sense in the visual scene, and is removed by hand. Not all scenes require deletions, an example of which is illustrated in Figure 6.31(b).

Automatically populated virtual environments do not require significant manual modifications, indicating that they are created consistently. Manual intervention occurs only in the form of modifications and deletions in the virtual environment.

6.5.4.3 Visual representation of behaviour

We conduct two experiments to determine if behaviour described by Transition and Relation annotation categories is visualized correctly in a scene. We investigate the following questions:



(a) Extract 1 (b) Extract 2 (c) Extract 3

Figure 6.31: Illustration of automatically generated virtual environment without manual intervention.

- *Is the behaviour described by Transition and Relation annotations visualized in the virtual environment?*
- *Is quantified behaviour (behaviour defined over an interval of time) visualized in a virtual environment?*

We use two extracts from different books in the investigation of the above questions.

Cow scene The cow scene, drawn from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942) is annotated as follows:

The picnic was lovely. They had it on the top of a hill, in a sloping field that looked down into a sunny **<setting>valley</setting>**. **<avatar>Anne</avatar>** didn't very much like a big brown **<object>cow</object>** who **<transition type="INSIDE" subject="cow">came</transition>** up **<relation type="near" subject="cow" object="her">close<relation>** and stared at her, but it **<transition type="OUTSIDE" subject="it">went</transition>** away when **<avatar>Daddy</avatar>** told it to. The **<avatar>children</avatar>** ate enormously, and Mother said that instead of having a tea-picnic at half-past four they would have to go to a tea-house somewhere, because they had eaten all the tea **<object>sandwiches</object>** as well as the lunch ones!
"What time shall we be at Aunt Fanny's?" asked **<avatar>Julian</avatar>**, finishing up the very last **<object>sandwich</object>** and wishing there were more.
"About six o'clock with luck," said **<avatar>Daddy</avatar>**. "Now who wants to stretch their legs a bit? We've another long spell in the car, you know." The **<object>car</object>** seemed to eat up the miles as it purred along.

A sequence of images taken from the resulting animated film is presented in Figure 6.32 (which is available for viewing in Appendix F). The cow enters and exits the scene at the correct moments according to the concurrent subtitles and audio, indicating the successful conversion of the Transition annotations to visual behaviour. The setting is interpreted correctly in providing a background suitable for the description "valley", and the appropriate geometric models appear in the virtual environment.

Follow scene The following scene, taken from the *World of Tiers 7: More than Fire* by Philip Jose Farmer (1993), provides an example indicating the strength of phrasing constraints in terms

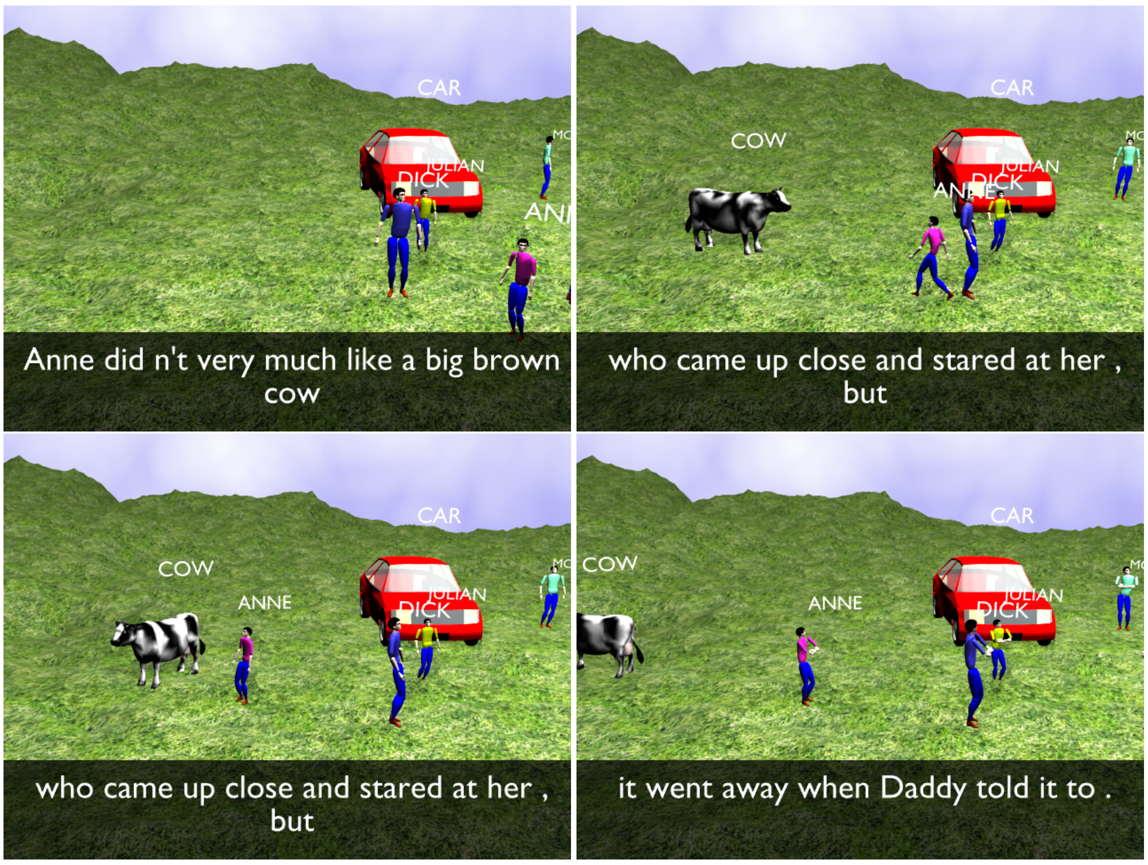

CAR
JULIAN
DICK
ANNE
Anne did n't very much like a big brown cow
COW
CAR
JULIAN
DICK
ANNE
who came up close and stared at her , but
COW
ANNE
CAR
JULIAN
DICK
who came up close and stared at her , but
COW
ANNE
CAR
JULIAN
DICK
it went away when Daddy told it to .

Figure 6.32: Cow scene from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

of contiguous intervals of time:

"THIS'LL BE IT!" <**avatar**>**KICKAHA**</**avatar**> SAID. "I KNOW IT, KNOW IT! I CAN feel the forces shaping themselves into a big funnel pouring us onto the goal! It's just ahead! We've finally made it!"
He wiped the sweat from his forehead. Though breathing heavily, he increased his pace. <**avatar**>**Anana**</**avatar**> was a few steps <**relation type="BEHIND" subject="Anana" object="him"**>**behind**</**relation**> and below him on the steep <**setting**>**mountain**</**setting**> trail. She spoke to herself in a low voice. He never paid any attention to her discouraging-that is, realistic-words, anyway.
"I'll believe it when I see it."

The above extract specifies a "following" motion, where the model representing Anana must be *behind* the moving Kickaha model over a time interval.

Snapshots from the final animated scene are presented in Figure 6.33 (the film is available for viewing in Appendix F). When the sentence "Anana was a few steps behind..." is encountered, the behaviour quantified by the *behind* Relation is visualized. The Anana avatar moves to a position "behind" the Kickaha model. This spatial relation between the two moving entities is maintained for the entire time interval, illustrated by the last four snapshots in Figure 6.33.

These examples demonstrate that the behaviour specified by Transition and Relation annotation categories is visualized appropriately. Quantified behaviour is visualized correctly in a virtual environment.

6.5.4.4 Complex behaviour

This section investigates the visualization of complex behaviour in a scene. We define complex behaviour as trajectories that include a number of way-points, specified as a chain of low degree curves (described in Section 6.3.3). We investigate the following question:

- *Is the complex behaviour described by Transition and Relation annotations visualized in a virtual environment?*

We use two extracts in the investigation of the above question.

Rabbit scene The Rabbit scene from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942) demonstrates complex motion with respect to three different entities in the scene, namely a rabbit, the avatar Timothy, and the avatar George (we use ellipsis to indicate where text

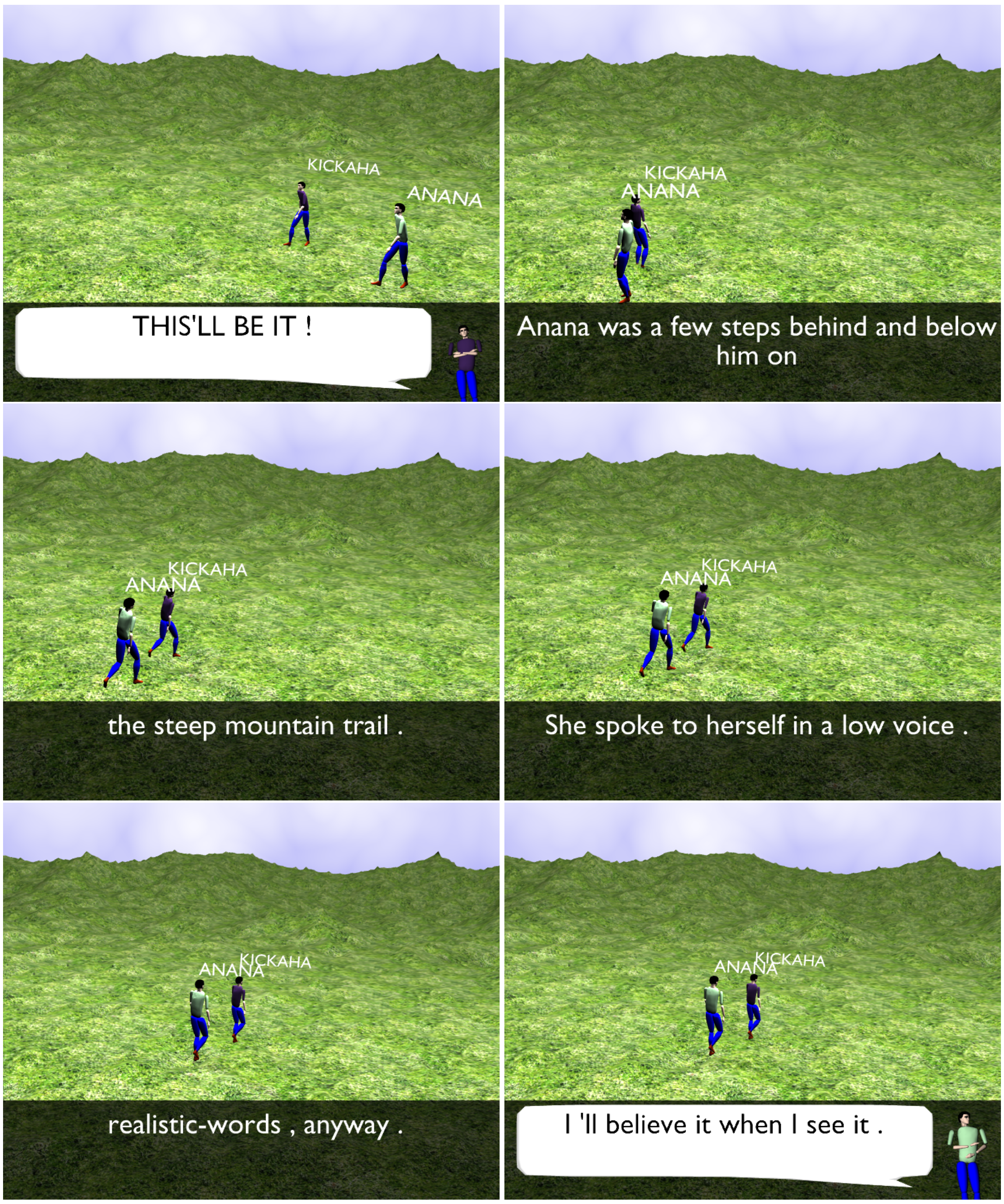

KICKAHA
ANANA
THIS'LL BE IT !
KICKAHA
ANANA
Anana was a few steps behind and below him on
KICKAHA
ANANA
the steep mountain trail .
KICKAHA
ANANA
She spoke to herself in a low voice .
KICKAHA
ANANA
realistic-words , anyway .
KICKAHA
ANANA
I 'll believe it when I see it .

Figure 6.33: Follow scene from the *World of Tiers 7: More than Fire* by Philip Jose Farmer (1993).

is omitted):

"Look! There's a rabbit!" cried **<avatar>Dick</avatar>**, as a big sandy **<object>rabbit</object>** lolloped slowly across the **<setting>yard</setting>**. It **<transition type=“OUTSIDE” subject=“It”>disappeared</transition>** into a hole on the other side. Then another rabbit **<transition type=“INSIDE” subject=“rabbit”>appeared</transition>**, sat up and looked at the children, and then **<transition type=“OUTSIDE” subject=“rabbit”>vanished</transition>** too. A third rabbit **<transition type=“INSIDE” subject=“rabbit”>appeared</transition>**. It was a small one with absurdly big ears ...

... But this was too much for **<avatar>Timothy</avatar>**. ... . He gave an excited **<foley>yelp</foley>** and rushed full-tilt **<relation type=“NEAR” subject=“He” object=“rabbit”>at</relation>** the surprised rabbit. ...

Then it turned itself about and tore off at top speed, its white bobtail going up and down as it bounded away. It disappeared **<relation type=“UNDER” subject=“It” object=“bush”>under</relation>** a gorse **<object>bush</object>** near the children. **<avatar>Timothy</avatar>** went after it, vanishing **<relation type=“UNDER” subject=“Timothy” object=“bush”>under</relation>** the big **<object>bush</object>** too. ...

"Tim! Do you hear me! Come out of there!" shouted **<avatar>George</avatar>**. "You're not to chase the rabbits ...

... **<avatar>George</avatar>** went to fetch him. Just as she got up **<relation type=“near” subject=“she” object=“bush”>to</relation>** the gorse **<object>bush</object>** the scraping suddenly stopped.

This extract contains a number of Transition annotations describing the behaviour of the Rabbit entity. A number of different behaviours are also specified by Relation annotations: Timothy rushes towards the Rabbit; the Rabbit moves under a bush; and Timothy follows the rabbit to the bush. The combination of these different behaviour types results in complex motion in the virtual environment.

Snap-shots of the first portion of the Rabbit scene are presented in Figure 6.34 containing an outdoor scene with rabbits, some avatars, a bush, and the dog Timothy (the animated film is available for viewing in Appendix F). The rabbit model moves in and out of the scene appropriately. The sequence of snapshots is continued in Figure 6.35, illustrating the motion of Timothy towards the rabbit and the subsequent motion of the rabbit towards the bush. The images visualize Timothy following the rabbit into the bush, as well as the subsequent movement of George to the bush.

Study scene The study scene from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942) is an extract describing complex behaviour, as well as describing static scene layout:

He stole **<transition type=“INSIDE” subject=“He”>in</transition>**. His **<avatar>uncle</avatar>** still snored. He tiptoed by him **<relation type=“NEAR” subject=“he” object=“table”>to</relation>** the **<object>table</object>** **<relation type=“BEHIND” subject=“table” object=“chair”>behind</relation>** his uncle's **<object>chair</object>**. He took hold of the **<object>box</object>**. And then a bit of the broken wood of the box fell to the floor with a **<foley>thud</foley>**! His uncle stirred **<relation type=“INSIDE” subject=“uncle” object=“chair”>in</relation>** his **<object>chair</object>** and opened his eyes. Quick as lightning the boy crouched down **<relation type=“BEHIND” subject=“boy” object=“chair”>behind</relation>** his uncle's **<object>chair</object>**, hardly breathing.

"What's that?" he heard his uncle say. **<avatar>Julian</avatar>** didn't move. Then his uncle settled down again and shut his eyes. Soon there was the sound of his rhythmic **<foley>snoring</foley>**!

"Hurrah!" thought **<avatar>Julian</avatar>**. "He's off again!" Quietly he stood up, holding the box. On tiptoe he crept to the French window. He slipped **<transition type=“OUTSIDE” subject=“He”>out</transition>** and ran softly down the garden path. He didn't think of hiding the box. All he wanted to do was to get to the other **<avatar>children</avatar>** and show them what he had done!

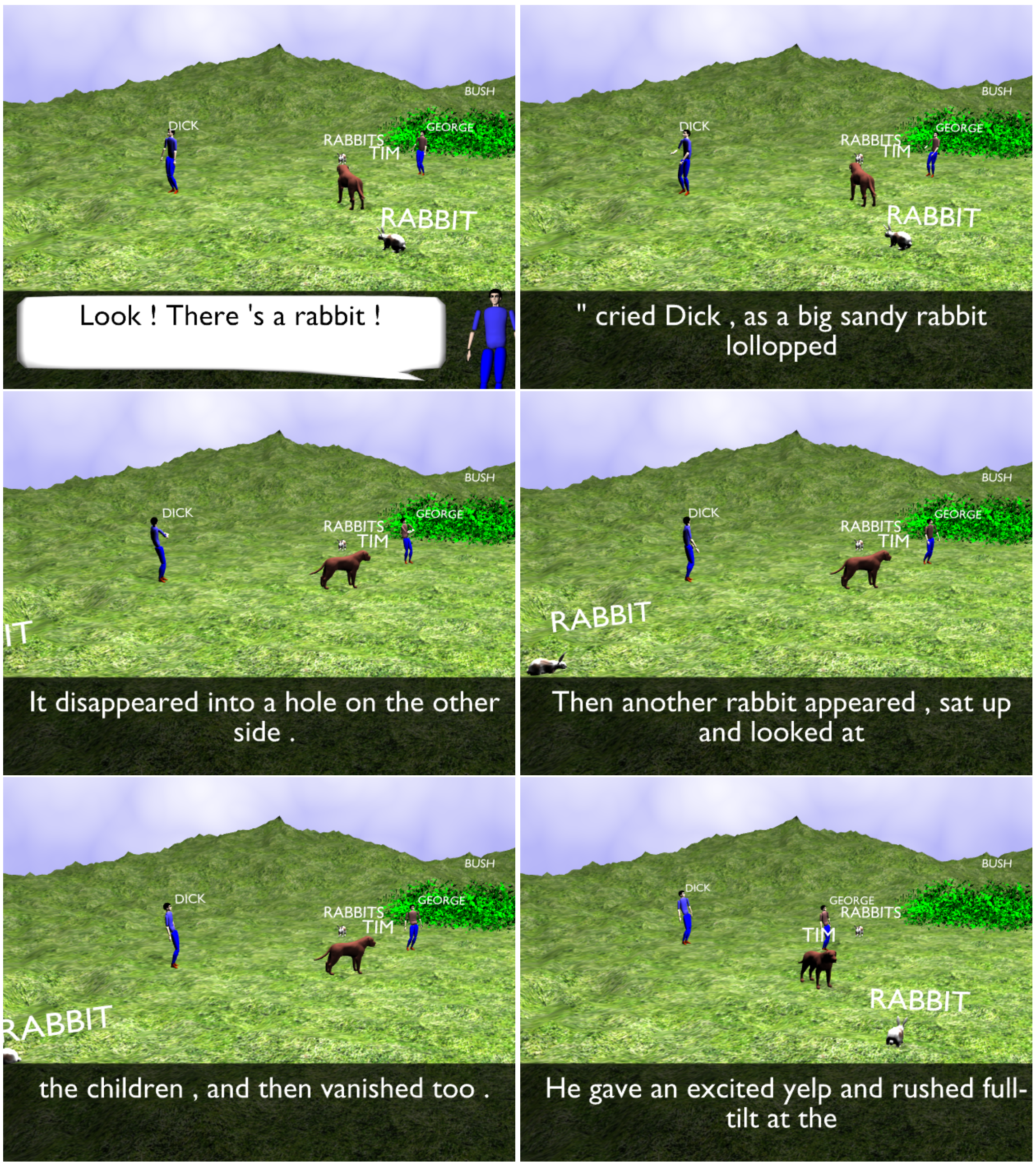

BUSH
DICK
GEORGE
RABBITS
TIM
RABBIT
Look ! There 's a rabbit !
" cried Dick , as a big sandy rabbit lollopped
It disappeared into a hole on the other side .
Then another rabbit appeared , sat up and looked at
the children , and then vanished too .
He gave an excited yelp and rushed full-tilt at the

Figure 6.34: Rabbit scene (1) from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

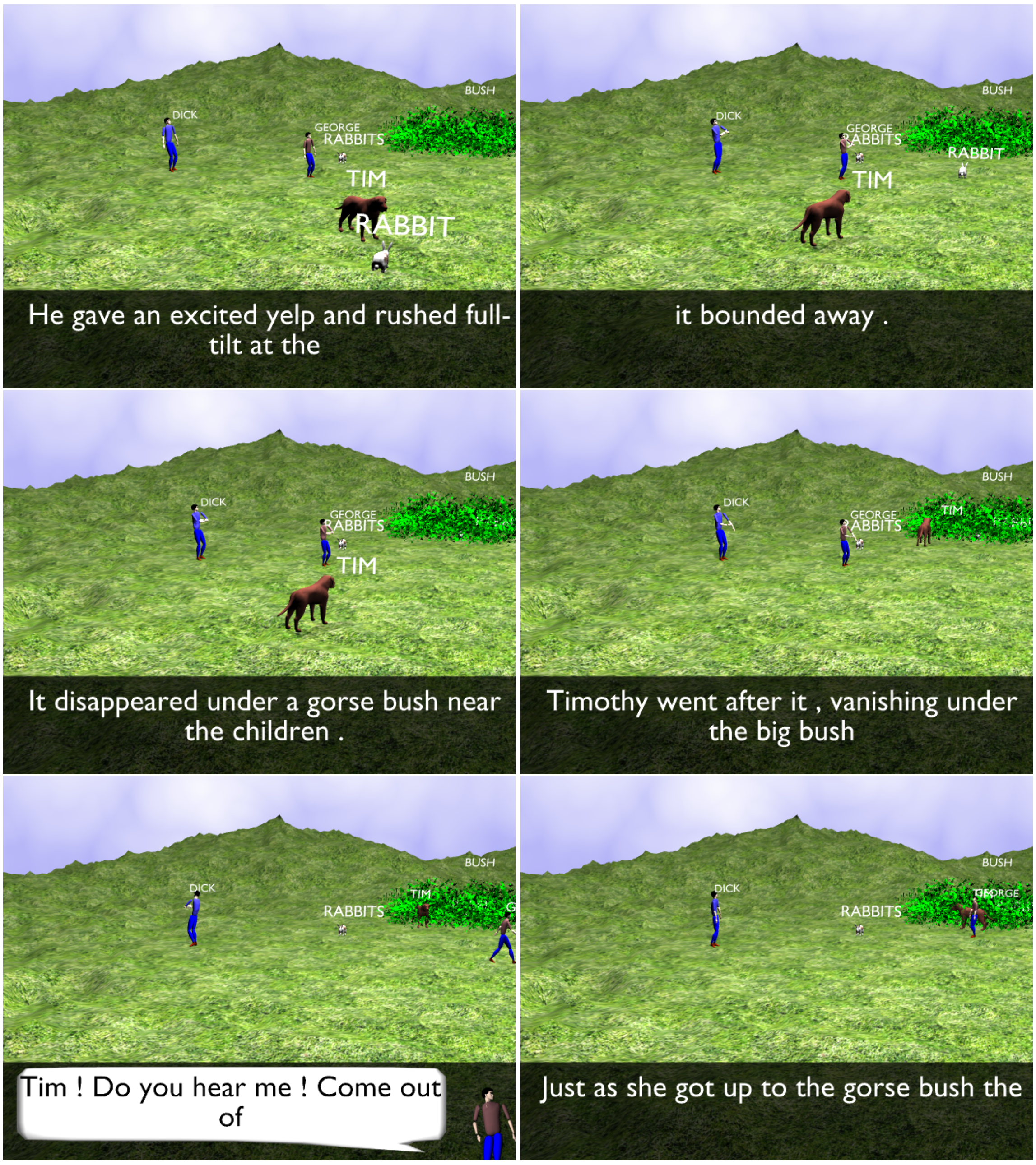

BUSH
DICK
GEORGE
RABBITS
TIM
RABBIT
He gave an excited yelp and rushed full-tilt at the
it bounded away .
It disappeared under a gorse bush near the children .
Timothy went after it , vanishing under the big bush
Tim ! Do you hear me ! Come out of
Just as she got up to the gorse bush the

Figure 6.35: Rabbit scene (2) from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

The above extract describes the motion of the avatar Julian as he sneaks into his uncle's study, to the table (which is described as being *behind* his uncle's chair). The boy then exits the scene.

Snap-shots of the final animated film are presented in Figure 6.36 and Figure 6.37 (available for viewing in Appendix F). The model representing Julian enters the scene, moves towards the table, moves behind the chair, and exits the room in a manner that corresponds to the concurrent subtitles. Quentin moves to his chair at the appropriate moment. This example demonstrates that complex behaviour is visualized in the form of correct movement through multiple way-points.

We identify problems with Figure 6.36, such as the fact that the box is not placed *on* the table. This fact is never explicitly stated in the text however, and the created scene is correct according to the annotations. One flaw is the initial location of Quentin, who should be in the chair throughout the scene. This constraint is only implemented from the point at which the annotation is encountered in the text, resulting in the movement of Quentin to his chair only midway through the scene. The visualized scene is also missing articulation described in the text, but we do not evaluate the scene on behaviour that includes crouching, picking up the box, dropping it, or taking the box out of the scene. These kinds of descriptions are not identified using the current set of annotation categories.

The results presented in this section demonstrate that complex trajectories are visualized appropriately in a virtual environment.

6.5.4.5 Consecutive scenes

This section investigates whether multiple scenes are visualized in one continuous animated presentation:

- *Is a presentation created that switches correctly from one scene to the next?*

We use two extracts in the investigation of the above question.

House sequence The following extract from the *Narnia* series, *The Lion, the Witch and the Wardrobe* by C.S. Lewis (1950) describes two different scenes:

They were sent to the <**object**>**house**</**object**> of an old <**avatar**>**Professor**</**avatar**> who lived in the heart of the <**setting**>**country**</**setting**>, ten miles from the nearest railway station and two miles from the nearest post office. He had no wife and he lived in a very large house with a housekeeper called <**avatar**>**Mrs Macready**</**avatar**> and three servants. (Their names were Ivy, Margaret and Betty, but they do not come into the story much.) He himself was a very old man with shaggy white hair which grew over most of his face as well as on his head, and they liked him almost at once; but on the first evening when he <**transition type="INSIDE" subject="he"**>**came**</**transition**> out to meet them at the front door he was so odd-looking that <**avatar**>**Lucy**</**avatar**> (who was the youngest) was a little afraid of him, and <**avatar**>**Edmund**</**avatar**> (who was the next youngest) wanted to laugh and had to keep on pretending he was blowing his nose to hide it.

As soon as they had said good night to the <**avatar**>**Professor**</**avatar**> and gone upstairs on the first night, the boys <**transition type="INSIDE" subject="boys"**>**came**</**transition**> into the girls' <**setting**>**room**</**setting**> and they all talked it over.

"We've fallen on our feet and no mistake," said <**avatar**>**Peter**</**avatar**>. "This is going to be perfectly splendid. That old chap will let us do anything we like."

"I think he's an old dear," said <**avatar**>**Susan**</**avatar**>.

"Oh, come off it!" said <**avatar**>**Edmund**</**avatar**>, who was tired and pretending not to be tired, which always made him bad-tempered.

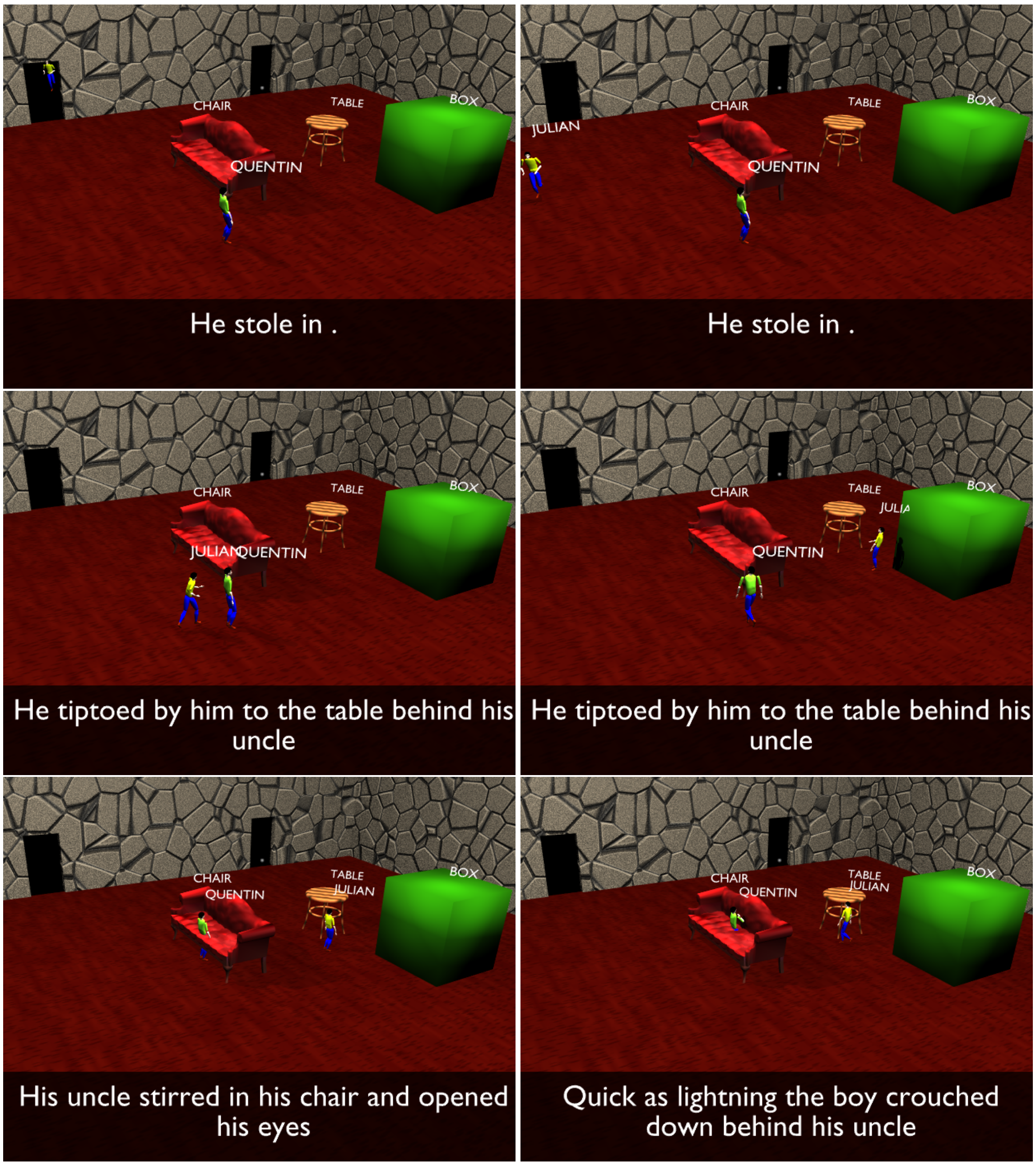

CHAIR
TABLE
BOX
QUENTIN
He stole in .
JULIAN
CHAIR
TABLE
BOX
QUENTIN
He stole in .
CHAIR
TABLE
BOX
JULIANQUENTIN
He tiptoed by him to the table behind his uncle
CHAIR
TABLE
BOX
JULIA
QUENTIN
He tiptoed by him to the table behind his uncle
CHAIR
QUENTIN
TABLE
JULIAN
BOX
His uncle stirred in his chair and opened his eyes
CHAIR
QUENTIN
TABLE
JULIAN
BOX
Quick as lightning the boy crouched down behind his uncle

Figure 6.36: Study scene (1) from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

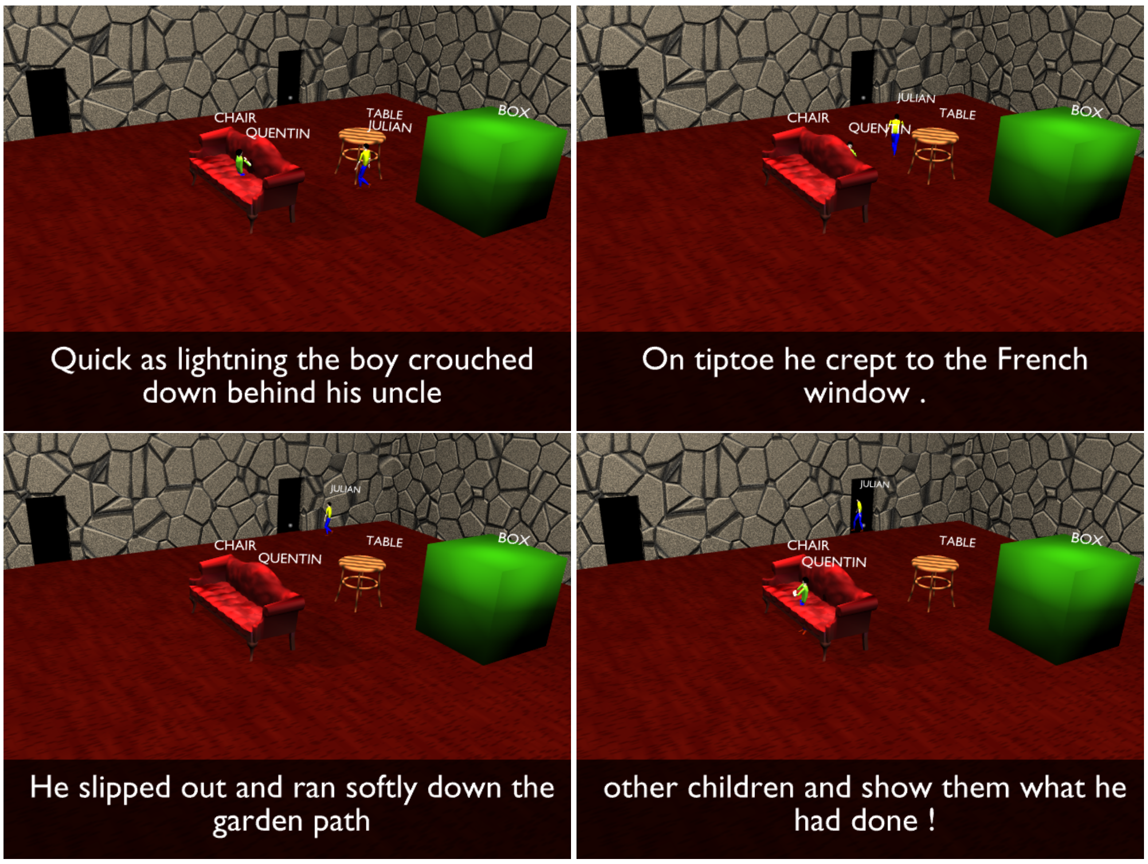
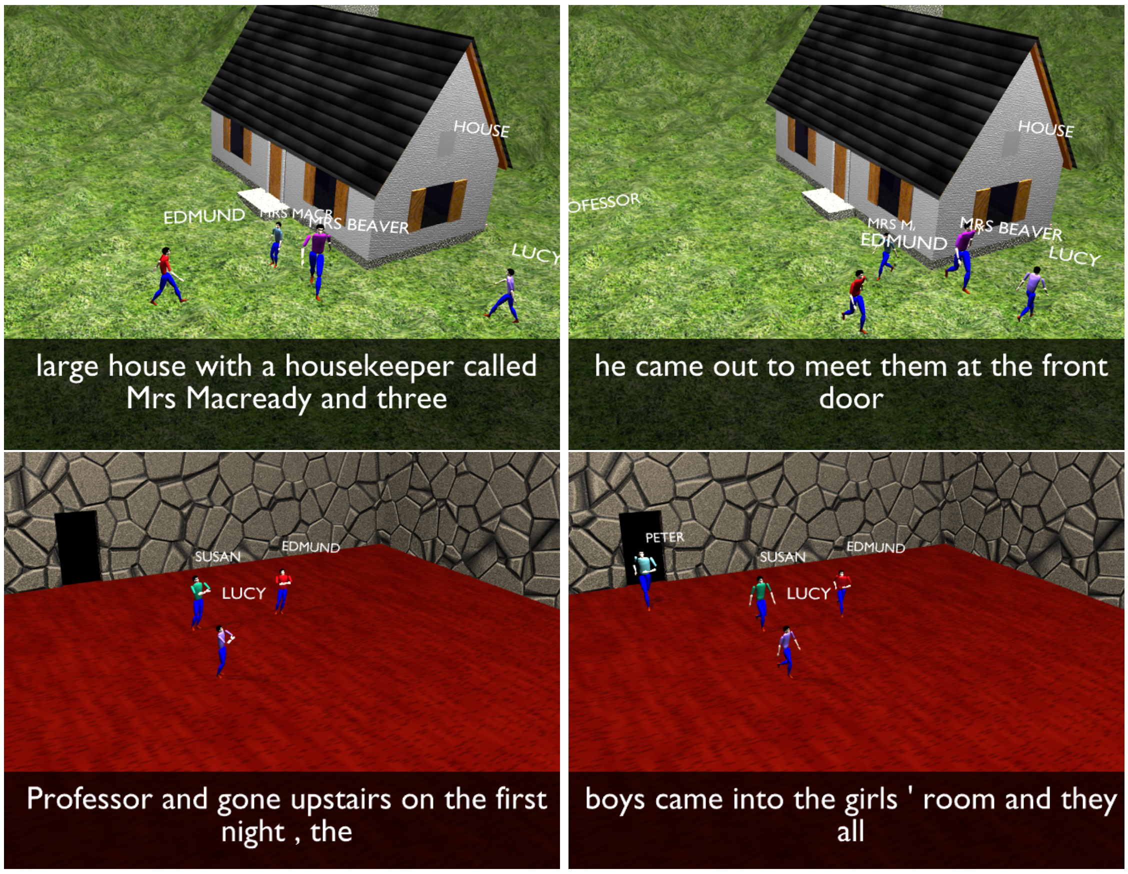

CHAIR
QUENTIN
TABLE
JULIAN
BOX
Quick as lightning the boy crouched down behind his uncle
JULIAN
CHAIR
TABLE
QUENTIN
BOX
On tiptoe he crept to the French window .
JULIAN
CHAIR
QUENTIN
TABLE
BOX
He slipped out and ran softly down the garden path
JULIAN
CHAIR
QUENTIN
TABLE
BOX
other children and show them what he had done !

Figure 6.37: Study scene (2) from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).



large house with a housekeeper called Mrs Macready and three

he came out to meet them at the front door

Professor and gone upstairs on the first night , the

boys came into the girls ' room and they all

Figure 6.38: House sequence from *Narnia 2: The Lion, the Witch and the Wardrobe* by C.S. Lewis (1950).

The above extract begins by describing the location of the Professor's house and the initial meeting of the characters outside it. The setting then changes to inside the house, in one of the rooms. A unique scene should be created for each setting, and each scene should be displayed at the correct moment in the animated film.

Snapshots from the final animated film are presented in Figure 6.38 (available for viewing in Appendix F). The first scene occurs in an outdoor setting, with a model representing a house. The change in scene is shown from the third snapshot onwards, where the setting is automatically changed to the room in which the avatars are having a conversation.

Travel sequence The following extract from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942) describes a number of successive scene changes:

<avatar>Dick</avatar> and **<avatar>Julian</avatar>**, who shared a **<setting>room</setting>**, woke up at about the same moment, and stared out of the nearby window.
"It's a lovely day, hurrah!" cried **<avatar>Julian</avatar>**, leaping out of **<object>bed</object>**. "I don't know why, but it always seems very important that it should be sunny on the first day of a holiday. Let's wake Anne."
<avatar>Anne</avatar> slept in the next **<setting>room</setting>**. **<avatar>Julian</avatar>** ran **<transition type="INSIDE" subject="Julian">in</transition>** and shook her.
"Wake up! It's Tuesday! And the sun's shining." Anne woke up with a jump and stared at **<avatar>Julian</avatar>** joyfully. "It's come at last!" she said. "I thought it never would. Oh, isn't it an exciting feeling to go away for a holiday!"
They started soon after breakfast. Their **<object>car</object>** was a big one, so it held them all very comfortably. **<avatar>Mother</avatar>** sat in front with **<avatar>Daddy</avatar>**, and the three **<avatar>children</avatar>** sat behind, their feet on two **<object>suitcases</object>**. In the luggage-place at the back of the car were all kinds of odds and ends, and one small **<object>trunk</object>**. **<avatar>Mother</avatar>** really thought they had remembered everything.
Along the crowded **<setting>London</setting>** roads they went, slowly at first, and then, as they left the **<setting>town</setting>** behind, more quickly. Soon they were right into the open **<setting>country</setting>**, and the **<object>car</object>** sped along fast. The **<avatar>children</avatar>** sang songs to themselves, as they always did when they were happy.
"Are we picnicking soon?" asked **<avatar>Anne</avatar>**, feeling hungry all of a sudden.
"Yes," said **<avatar>Mother</avatar>**.

Snap-shots of the automatically produced animated film are presented in Figure 6.39 (available for viewing in Appendix F). Two visually distinct room settings are produced automatically: a city setting representing London; and an outdoor setting representing "country". Each scene appears correctly according to the concurrent subtitles, successfully demonstrating the sequencing ability of the conversion process.

This results in this section demonstrate that a sequence of scene descriptions in fiction text is successfully converted into a corresponding presentation that contains appropriate visual changes.

6.5.4.6 Appropriate media

We investigate whether the automated conversion process creates appropriate media for presenting the story:

- *Are geometric models selected that adequately visualize the entities and background scenery in an environment?*
- *Does content in different modalities (graphics, audio) represent the fiction text and is it aligned correctly in the final animated presentation?*

We do not perform an individual experiment for this investigation, but refer to results from previous experiments as evidence for answering the above questions.

All the experiments detailed previously in this section produce visual results that depict appropriate visualizations of the entities described in the extracts. This includes the appropriate visualization of avatars and objects. Each virtual environment created automatically in these experiments also contains background scenery appropriate to the described setting. These observations are made with one reservation in mind, namely that the appropriate selection of models is dependent on the variety of models available in the model library.

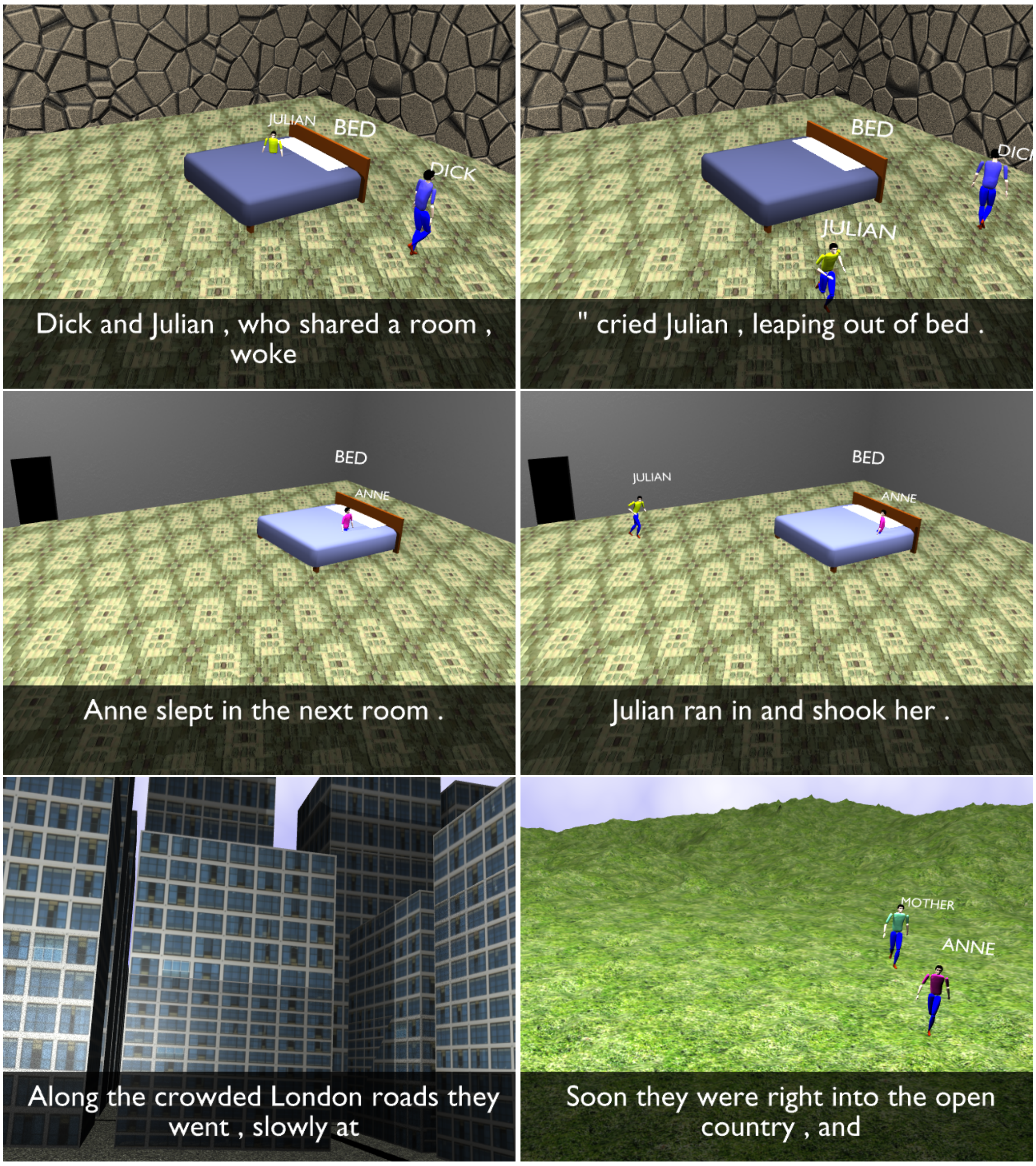

JULIAN
BED
DICK
Dick and Julian , who shared a room , woke
BED
DICK
JULIAN
" cried Julian , leaping out of bed .
BED
ANNE
Anne slept in the next room .
JULIAN
BED
ANNE
Julian ran in and shook her .
Along the crowded London roads they went , slowly at
MOTHER
ANNE
Soon they were right into the open country , and

Figure 6.39: Travel sequence from the *Famous Five 1: Five on a Treasure Island* by Enid Blyton (1942).

All the aforementioned experiments result in animated films that provide evidence in support of the use of audio narrations, the use of different voices for different avatars, as well as the use of visually enhanced subtitles. The digital animations available in Appendix F include the audio modality for each film, demonstrating the successful synchronization between the audio (narrations and foleys), the subtitles, and the behaviour in the graphical scene.

These examples demonstrate that the automated process appropriately selects and creates geometry for visualizing the described scenes, and supports the creation of presentations that contain multiple correctly aligned modalities.

6.5.4.7 Applicability to different types of books

Evidence presented in Chapter 4 suggests that books of different type (different author and readability index) impact the success of the automated conversion process. We investigate the following question:

- *Is the automated process for creating multi-modal animated 3D films from annotated fiction text applicable over books of different type?*

We do not perform an individual experiment for this investigation, but refer to results from previous experiments.

The suite of extracts presented in this section are sourced from 3 different fiction books of different type. Consistent virtual environments and animated films are produced from all three sources. This demonstrates that the automated process is applicable over books of different type, for the categories of annotation used in these experiments.

6.5.5 Summary of findings

We conclude that corresponding multi-modal animated 3D virtual environments and films are created using the processes described in this chapter. This conclusion is supported by the following observations from the experiments conducted in this section:

1. The creation of high-level scene descriptions (including the identification of scenes, their content, and behaviour) is performed consistently, requiring minimal human modification.
2. Virtual environments are populated in a manner that requires little significant modification.
3. Automatically generated virtual environments are representative of the input text, specifically with regards to the following aspects:
   (a) Behaviour specified by annotation categories such as Transition and Relation is visualized correctly in a virtual environment.
   (b) Complex behaviour is visualized correctly in a virtual environment.
   (c) Visual sequences containing multiple scenes are created that correctly represent descriptions in the input text.
   (d) Appropriate media is chosen for different modalities of representation, including the selection and creation of visual geometry, and the insertion of audio narrations and sound effects.

(e) The automated process supports the creation of multi-modal animated 3D virtual environments and films across different types of books.

6.6 Conclusion

This chapter presents a process that successfully interprets annotated fiction text to create corresponding multi-modal animated 3D virtual environments and films. We summarize this process in terms of three stages, namely the automatic creation of high level scene descriptions from annotated fiction text, the quantification of behaviour in virtual environments (using constraint formulation and optimization), and the instantiation and population of the virtual environments using automated techniques.

We draw the following conclusions with regards to the original problems stated in Section 6.1.1:

1. The interpretation of annotations is performed successfully in the creation of high-level structured scene descriptions using knowledge-poor approaches.

    (a) The set of annotation categories defined in Chapter 4 provide for the automatic specification of scene detail. Setting annotations correctly identify the scenes to portray visually, while Object and Avatar annotations successfully identify the entities contained in each scene. Transition and Relation annotations specify the behaviour of entities in each scene. This shows that the limited set of annotation categories we define provides sufficient detail to specify scenes in a structured and complete manner.

    (b) The definition of a scene as a finite space that exists over a contiguous interval of time (provided in Chapter 5, Definition 5.1 on page 107) is fundamental to the interpretation of a book as a collection of independent virtual environments. This justifies the segmentation of text according to physical location.

    (c) Visual consistency regarding the representation of entities in different environments is achieved using entity descriptors, which are scene-independent instantiations of each entity. These descriptors provide for the assignment of unique visual attributes to each entity, including a representative visual icon, colouring, and motion type.

    (d) The link between instantiated entities and the annotations is maintained through the use of co-reference. A knowledge-poor method successfully creates these links (using the gender specificity of personal pronouns), which are used for attributing behaviour to entities in a scene.

    (e) Abstract constraints are fundamental for summarizing time-quantified behaviour in a structured manner, while still permitting human review and correction. They also provide a means through which implicit physical restrictions (such as gravity and interpenetration) can be defined.

    (f) Time is derived in a knowledge-poor fashion using synthesized audio equivalents of the fiction text. This method quantifies behaviour in a manner that translates to visualizations corresponding to the descriptions in the text.

   (g) Abstract constraints are directly translated into analytical constraints due to their structured nature. In this manner, a human need not be concerned with complex low level behaviour reasoning. However, manual control over behaviour is still provided through the human readable abstract constraints.
   (h) The phrasing of constraints in terms of distance provides a valuable ability to measure the quality of the quantified behaviour during the optimization process (using the optimizer described in Chapter 5). This is particularly useful where the described behaviour is inconsistent, or the time permitted for constraint optimization is limited. This provides the designer with an indication of the quality of the current specified behaviour, allowing optimization to be terminated prematurely if quality is sufficiently high for the designer's needs.
   (i) The disadvantage of the use of analytical constraints for quantifying scene behaviour is the lack of solving techniques capable of finding solutions to constraints containing large numbers of variables. This problem is overcome by expressing trajectories as sequences of low degree curves (resulting in reduced numbers of variables in each constraint system), and incremental solving. Complex behaviour is produced efficiently, while maintaining the benefits of environment-independent spatial reasoning (described in Section 5.2.3 on page 112).

2. Virtual environments are successfully instantiated and populated from scene descriptions using automated techniques. There is no need for repetitive 3D modeling and animation, and rich virtual environments are created rapidly.

   (a) Entities in virtual environments exhibit visual consistency in relation to size and orientation by using 3D models that are standardized in these respects. Entity visualization is enhanced through the use of motion-capture for realistic articulation.
   (b) Detailed background scenery of unlimited size is successfully produced using procedural methods.
   (c) The derivation of time from audio representations of the text provides an accurate time-line against which multiple modalities (visual, audio, and text) are aligned.
   (d) The automated methods support the creation of multiple virtual environments as well as a coherently sequenced multi-modal animated film presentation.

The virtual environments created from automatically interpreted text visually convey the environments and behaviour described. This is supported by the range of experiments conducted in Section 6.5. In summary, multi-modal animated 3D virtual environments and films are created successfully using the techniques described in this chapter.

The techniques presented in this chapter contribute novel innovations regarding the text-to-graphics task:

- This work presents the first use of text sourced from popular fiction books as input for the creation of animated 3D graphics.

- We use a knowledge-poor approach that allows human intervention in the conversion process. This is different to existing text-to-graphics research which focuses on providing fully automatic knowledge-centric processes.
- The use of corresponding audio files for deriving temporal information for quantifying behaviour is innovative in the domain of text-to-graphics.
- We present the first use of interval-based constraint optimization for determining quantified scene behaviour. This method is innovative in its ability to measure the quality of derived behaviour, and no evidence exists in related research regarding similar metrics.
- The methods described in this chapter contribute to the field of virtual reality by providing a mechanism for automatically instantiating and populating virtual environments. Our method creates virtual environments without the need for extensive manual effort in 3D modeling, environment design and motion quantification.

A number of aspects of the automatic conversion process stand to benefit from future improvement. Our primary focus is on expanding support for different categories of annotation, including descriptive phrases that specify visual features such as colour and emotion, as well as articulated behaviour that includes posing avatar models.

Chapter 7

Conclusion

This chapter summarizes our approach for automating the fiction-to-animation task (Section 7.1). Conclusions drawn from this research are presented in Section 7.2. We describe significant contributions and innovations in Section 7.3.

7.1 Summary

We view the conversion of fiction text into multi-modal animated virtual environments as two problems: the analysis of the natural language text to create a structured intermediate representation; and the interpretation of this intermediate representation for creating a corresponding animated virtual environment.

Text analysis begins with the creation of surface annotations, which involves identifying the structural and syntactic properties of fiction text. We use a custom-built text tokenizer and sentence splitter for identifying structural properties. Publicly available tools provide syntactic information including parts-of-speech, syntactic function, and phrasing. Accuracies of greater than 95% can be expected for every category of surface annotation.

We use annotated fiction text as the structured intermediate representation of a fiction book. We create these annotations using a pattern-based machine learning approach. Rules are induced from manually provided examples (supplemented with automatically generated surface annotations). The result is a model that creates annotations specific to the style of the human who provides the examples. We believe that error potentially introduced by incorrect surface annotations is accommodated by the induction of special-case rules, and has minimal impact on the creation of semantic annotations. Models are induced that create correct annotations in different categories, and success varies according to the category (we report accuracy levels ranging between 51.4% and 90.4%). We observe that the greater the number of examples provided, the more accurate the automatically produced annotations become.

The interpretation of the annotated fiction text involves formulating structured scene descriptions, quantifying entity behaviour in a virtual environment, and populating corresponding virtual environments.

We use knowledge-poor techniques for formulating scene descriptions from annotations. Scene descriptions include: a list of the different scenes to visualize (using the Setting annotation);

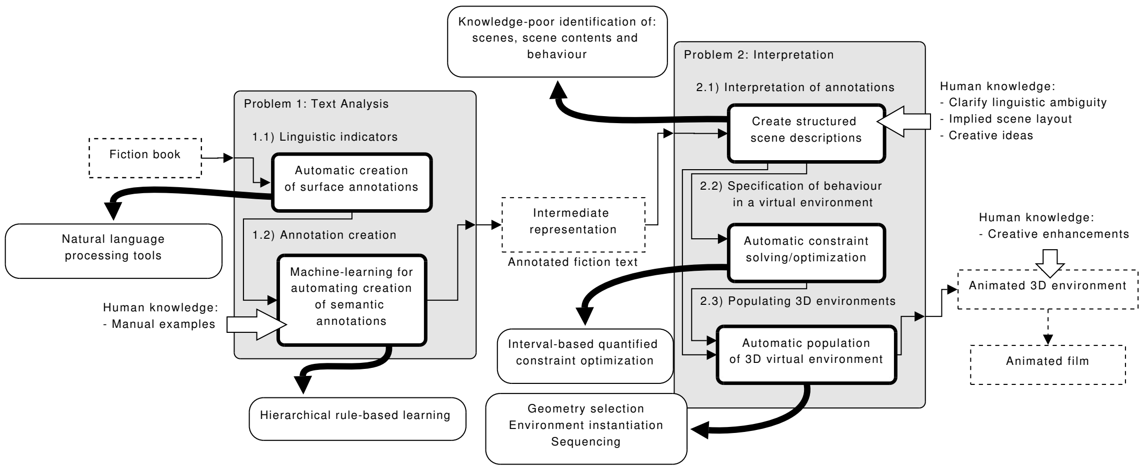

Knowledge-poor identification of: scenes, scene contents and behaviour
Problem 2: Interpretation
2.1) Interpretation of annotations
Human knowledge:
- Clarify linguistic ambiguity
- Implied scene layout
- Creative ideas
Problem 1: Text Analysis
1.1) Linguistic indicators
Create structured scene descriptions
Fiction book
Automatic creation of surface annotations
2.2) Specification of behaviour in a virtual environment
Intermediate representation
Human knowledge:
- Creative enhancements
Natural language processing tools
1.2) Annotation creation
Automatic constraint solving/optimization
Annotated fiction text
Machine-learning for automating creation of semantic annotations
Animated 3D environment
2.3) Populating 3D environments
Human knowledge:
- Manual examples
Interval-based quantified constraint optimization
Automatic population of 3D virtual environment
Animated film
Hierarchical rule-based learning
Geometry selection
Environment instantiation
Sequencing

Figure 7.1: Summary of the problems and automated processes that solve them for the complete fiction-to-animation task.

entities that populate each scene (using Avatar and Object annotations and a library of geometric models); and structured descriptions of entity behaviour in each scene (by translating Transition and Relation annotations into time-based constraints). Time values that quantify behaviour are acquired from audio representations of the text. We observe that between 83% and 96% of the automatically created scene descriptions (from a set of extracts from fiction text) require no human modification, validating our knowledge-poor techniques.

Behaviour is quantified in a virtual environment by formulating symbolic time-quantified constraints, and subsequently searching for solutions. Constraints are produced automatically, and are potentially inconsistent due to ambiguity in the text or erroneous annotations. Interval-based quantified constraint optimization automates the search for constraint solutions, and provides approximate solutions for inconsistent constraints. This method outperforms existing constraint solving approaches for the types of constraints produced by our fiction-to-animation system.

Structured scene descriptions and quantified behaviour are used to automate the population of virtual environments. One virtual environment is created for each scene in a book. We automatically select 3D geometric models from a library and procedurally generate geometry for the background scenery of each environment. Geometry is animated according to the quantified behaviour. Additional modalities are automatically synchronized, including an audio version of the narration and textual subtitles.

Error introduced by inconsistent semantic annotations has little impact on the interpretation mechanisms. We provide opportunities for human intervention where errors potentially occur (such as direct modifications to the virtual environment), but processes such as quantified optimization enable scene creation even where inconsistencies occur. We provide a suite of multi-modal animated examples to demonstrate that the automatically generated presentations correspond to the original fiction extracts.

The complete fiction-to-animation process is illustrated in Figure 7.1 (first presented as Figure 1.7 on page 8 in Chapter 1), and is augmented with the innovative methods we use for accomplishing each task.

7.2 Conclusions

The conversion of fiction text into corresponding virtual environments is automated using the techniques we have developed. Automated processes replace repetitive manual tasks in text analysis and interpretation, and as a result of this we maintain that:

$\boxed{\text{The process of converting a fiction book into an animated 3D film can be automated.}}$

With reference to the original problem statement in Section 1.2 on page 3, we conclude:

1. ***Text analysis is automated using hierarchical rule-based learning.***
   This method automatically creates semantic annotations that comprise the structured intermediate representation of the fiction book. This is significant because we now have an automated method for identifying different categories of visual description in fiction text, a source previously considered as too unstructured and complex to use in the text-to-graphics context. In this manner we replace the repetitive tasks of manually reading and comprehending the text.

2. ***Virtual environments are populated using techniques that interpret semantic annotations.***
   Interpretation methods automatically derive scene descriptions from semantic annotations, quantify behaviour in a virtual environment, and populate virtual environments with geometry. This is significant because it rapidly and deterministically constructs rich virtual environments that contain both visual detail and dynamic entity interactions. The need for repetitive tasks such as manual scene planning, 3D modeling and animation is eliminated.

The consequence of this work is a system that reduces the manual effort in performing the fiction-to-animation task, and which can be used as a labour saving device in existing animation work-flows. It creates virtual worlds and films quickly and cheaply, without requiring specialist expertise.

Annotated fiction text

The central feature of our approach to the fiction-to-animation task is the use of annotated text as the intermediate representation. Annotated fiction text is beneficial in a number of respects:

- The human readable format supports a boot-strapping process for training the hierarchical rule-based learning system. This facilitates the manual creation of examples for the induction process, and also supports review and correction of automatically produced annotations.
- Annotations are interpreted using automated processes, as a result of the structured format. This is significant because it provides the mechanism that directly enables portions of unstructured free text to be used as parameters for procedures that automatically produce corresponding visual geometry.
- Annotations maintain a direct link between the source text and the resulting interpretation. This is advantageous for deriving timing information (for behaviour) from annotated triggers, and providing a mechanism for synchronizing multiple modalities. This link is also

significant because it exposes the relationship between fragments of text and corresponding visualizations.

- Multiple categories of visual description are identified using annotations. This provides a rich array of structured visual descriptions, which is easily extensible when required.
- Annotations qualify visual descriptions with other fragments of text, or with semantic information. Qualifiers parametrize subsequent interpretation modules, resulting in unique visualizations that correspond to individual descriptions.

The significance of our work in text-analysis and interpretation is discussed in the following sections.

7.2.1 Text analysis

The processes we describe create an intermediate representation of the fiction text:

1. Surface annotations are created for natural language with high levels of accuracy. This limits the error that is introduced into the subsequent automatic creation of annotations.
2. Accurate models for creating annotations are induced from manual examples using hierarchical rule-based learning. The implication is that our automated mechanism can be refined to match to a human's annotation style, and produce similar annotations to the examples provided.
   (a) Tree-structures encapsulate the structural and syntactic properties of text, and generalize concepts to make them more applicable. These provide an effective mechanism for expressing patterns in the English language that identify annotations in fiction text.
   (b) A model consists of a set of hierarchical rules, and is capable of describing the wide range of scenarios peculiar to a category of annotation. This is significant in that both common and rare scenarios can be accommodated in a single model. The set is also extensible, which provides for future refinement of the model by adding rules to deal with special cases if needed.
   (c) A generalized rule-set creates correct annotations in unseen text. This demonstrates that the induced patterns model the underlying principles used by humans to represent a particular annotation category.
   (d) Rule-structures can be tailored for different annotation categories, without modifying the core learning algorithms. Accurate models are induced for multiple categories of annotation, resulting in a more descriptive intermediate representation. This limits optimizations to affect only the rule-structure if there is a need to improve the accuracy of an induced model in a particular category.
   (e) Accurate rule-sets are induced for qualifying annotations with text-references and semantic concepts. This strategy provides semantic information without the need for an external knowledge base.

We believe that hierarchical rule-based learning induces models that express fundamental rules in a human's cognition of English. The provision of enough examples will result in a model that is applicable to general fiction. However, it is impossible to determine what quantity would be sufficient for achieving this state given the abundance of available fiction titles.

7.2.2 Interpretation

The processes we describe create animated virtual environments and films from annotated fiction text:

1. Accurate scene descriptions are derived from annotated fiction text. This validates our knowledge poor approaches for interpreting semantic annotations, and shows that the limited set of annotation categories we define provides sufficient detail to specify scenes. The implication is that visualizations can be enhanced with the addition of further annotation categories.

   (a) A book can be interpreted as a collection of independent virtual environments (scenes). The concept of a scene provides an elegant method for presenting a consistent ordering of events, because time is one contiguous interval in each scene. This eliminates the need to "unravel" the ordering of events (as would be the case if the entire book is interpreted as one virtual universe).

   (b) Visual consistency between different scenes is maintained using entity descriptors. This is important for the multi-scene property of fiction writing, and ensures that entities have the same appearance in each virtual environment. This is also significant from a virtual reality perspective, in that one unique entity is instantiated for representing that entity in any potential environment.

   (c) Annotations are linked correctly to instantiated entities (descriptors) using a knowledge-poor co-reference technique. This enables behaviour to be attributed to the correct entities in each scene, regardless of the type of book.

   (d) Abstract constraints summarize time-quantified behaviour in a structured manner, while still maintaining a human readable format. This provides opportunity for human review and correction, while enabling subsequent use of automated techniques. Abstract constraints also provide a means through which implicit physical restrictions (such as gravity and interpenetration) can be expressed.

   (e) Behaviour that corresponds to the text is quantified (in terms of time) using synthesized audio. This is significant in that temporal information is provided without the need for knowledge-based reasoning.

   (f) Abstract constraints are directly translated into analytical constraints, due to their structured nature. In this manner, a human need not be concerned with complex low level behaviour reasoning. However, manual control over behaviour is still provided through the human readable abstract constraints.

   (g) The phrasing of constraints in terms of distance provides a valuable ability to measure solution quality during the optimization process. This provides the designer with an indication of the quality of the current specified behaviour, allowing optimization to be terminated prematurely if quality is sufficiently high for the designer's needs.

(h) A disadvantage of using analytical constraints is the limited scalability of solution finding methods with respect to the number of variables. Expressing trajectories as sequences of low degree curves (reducing the number of variables) and incremental solving overcomes this limitation. Complex behaviour is produced efficiently, while maintaining the benefits of environment-independent spatial reasoning (described in Section 5.2.3 on page 112).

2. Interval-based quantified constraint optimization locates solutions that specify time-based behaviour in a virtual environment. This means that coherent visual behaviour can be specified between dynamic bodies in our virtual environments.

   (a) Constraints formulated over contiguous intervals of time and space are represented effectively using interval arithmetic. This enables direct solving of symbolically phrased constraints, and removes the need to transform systems into discrete representations more suitable for conventional computer-based solving techniques.

   (b) Solutions are guaranteed to be consistent over the contiguous time interval. This ensures that no erratic behaviour is produced in a scene that could potentially result from aliasing when using discrete solving methods.

   (c) Relaxed constraints and iterative tightening narrows the search space in a manner that tends towards actual solutions. This is significant because it reduces search time (particularly for fiction-to-animation problems), but also establishes a correlation between search time and the proximity of the process to an actual solution. The consequence is that if a designer has patience to wait then a higher quality solution is likely (as opposed to *solving* methods where this correlation does not apply).

   (d) There is a trade-off between locating sound solutions and the amount of available search time. If pressed for time a designer is able to use an approximate solution for subsequent scene population processes, but is able to incorporate better quality solutions at a later stage if the optimization is left to continue concurrently.

   (e) The interval-based quantified constraint optimizer ensures a behaviour specification (even for inconsistent constraint systems) through the provision of approximate solutions. The significance is that it enables the automatic creation of constraints from text, without the need for semantic consistency checking. This provides for the population of subsequent virtual worlds, regardless of inconsistencies in the text or annotations.

3. Multi-modal virtual environments are populated automatically. There is no need for repetitive 3D modeling and animation, and results in the rapid creation of rich virtual environments containing both visual and behavioural visualizations.

   (a) Entities in virtual environments are represented visually in a coherent and convincing fashion. The automatically produced environments contain enough basic content to visually portray concepts in the text.

   (b) Multiple modalities (visual, audio, and text) are aligned correctly. We can generate alternative combinations of modalities for producing varied presentations of the text, including coherently sequenced multi-modal animated film presentations.

Virtual environments produced from annotated fiction text are significant in that they not only *contain* described entities, but they also *portray* plot-line through the animated behaviour. This ability is further enhanced by the inclusion of other modalities. We believe that this ability benefits other applications besides the creation of films, including populating environments for virtual reality applications such as interactive storytelling, education or massive multi-player games.

7.3 Contributions

We make a number of novel contributions in the text-to-graphics and associated domains.

The use of a knowledge-poor paradigm for solving the text-to-graphics problem

Existing text-to-graphics systems rely on the existence of a pre-constructed knowledge base. Our research shows that knowledge-poor techniques are effective in producing coherent animated graphics, using only minor human intervention to provide world-knowledge. In all existing research human intervention is avoided at all costs, usually at the expense of sacrificing certain scene creation capabilities due to limitations in the quantity of encoded world knowledge (Coyne and Sproat, 2001; Lu and Zhang, 2002; Ma, 2006). Our research contributes to the text-to-graphics domain by demonstrating that the bulk of repetitive manual effort is eliminated. The restrictions on knowledge-centric approaches are removed.

The use of text sourced from popular fiction books as a source of input

Most text-to-graphics research uses restricted or contrived language as input (Lu and Zhang, 2002; Zeng et al., 2003; Johansson et al., 2005; Ma, 2006). We source language directly from popular fiction books, without simplifying or paraphrasing the text. This represents a significant contribution to the text-to-graphics domain.

Annotated text as an intermediate representation

Our approach of using annotated fiction text as an intermediate representation is novel. Other existing intermediate representations (such as semantic frames (Coyne and Sproat, 2001) or lexical visual semantic representations (Ma, 2006)) seldom support automatic creation and interpretation while simultaneously exhibiting human-readability.

Annotated text is significant in its ability to be extended to different visual categories. Our definition of annotation categories including Avatar, Object, Setting, Transition and Relation provides a starting point for future fiction-to-animation development.

Pattern-based machine learning for performing the text analysis task

Existing text-to-graphics systems use widely varying methods for text analysis. The most popular trend is the combination of syntactic analysis with encoded world knowledge. We use an alternative technique based on information extraction. CarSim (Johansson et al., 2005) is the only text-to-graphics system that uses information extraction, employing a statistical machine learning approach. Our approach is innovative because it is the only pattern-based learning mechanism for performing text analysis.

The hierarchical rule paradigm is novel in the field of pattern-based information extraction. Our use of tree-structures for representing patterns, and our techniques for generalizing these patterns by pair-wise comparison and insertion of wild-cards is a novel approach to inducing rules regarding the English language. The ability to induce patterns for different annotation tasks (such as identifying text fragments in a category, identifying relationships between text fragments, and associating semantic information) is a significant contribution.

Interval arithmetic for quantifying time-based behaviour

The majority of related text-to-graphics approaches specify behaviour using encoded world-knowledge (Coyne and Sproat, 2001; Lu and Zhang, 2002; Ma, 2006). Few systems use constraint-based techniques that do not require detailed knowledge bases (Johansson et al., 2005). We develop a previously unexplored technique that uses interval-based constraint optimization.

Our method for optimizing universally quantified constraint systems extends the state of the art in quantified constraint solving. This method is innovative in its ability to provide approximate solutions to inconsistent quantified constraint systems, as well as provide a quantitative measure of the quality of these approximations.

We develop a novel approach that overcomes the limitations of solution finding mechanisms (in terms of scalability with respect to the number of variables). This approach phrases behaviour as sequences of low degree trajectories and solves constraint systems in an incremental fashion, thereby ensuring that the number of variables remains small enough for efficient solving.

A knowledge-poor approach to deriving time from fiction text

Existing text-to-graphics systems that specify time-based behaviour use knowledge-rich analysis of the input text to derive temporal information (Lu and Zhang, 2002; Ma and McKevitt, 2003, 2004c; Johansson et al., 2005). Our use of audio representations is a contribution in this respect because it avoids complex reasoning and analysis, and also creates behaviour that corresponds visually to the concurrent presentation of the original text.

More efficient development processes for virtual reality and film production

The fiction-to-animation system presented in this research demonstrates that the transformation of a fiction book into an animated film is automated using computer technology. Our automated system removes the requirement for repetitive manual effort in converting the fiction text into corresponding virtual environments. This is significant because it allows human effort to be exerted elsewhere in the creative process, resulting in environments and films that contain rich visual detail for less cost or effort.

7.4 Future work

A weakness in our current implementation of the fiction-to-animation process is the limited range of annotation categories. Fiction text contains types of descriptions beyond the categories used in

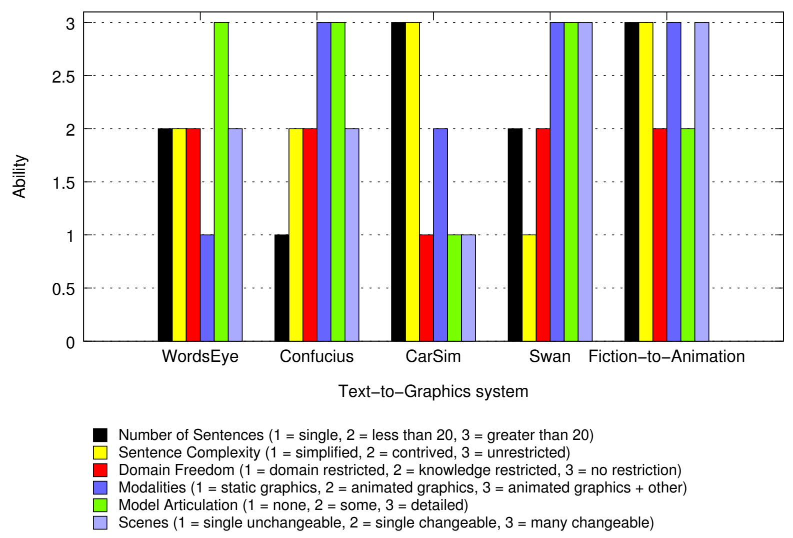


Figure 7.2: Comparative ratings of capabilities for related text-to-graphics systems.

this exposition, including descriptions of entity features and emotion, the identification of poses and actions of entities in the scene, as well as setting-related detail. Hierarchical rule-based learning is effective for creating different categories of annotation, and will assist in the creation of large corpora of fiction text containing additional annotated visual descriptions.

The current methods for creating virtual environments are limited in their ability to add visual detail. This limitation is deliberate in this exposition so as not to obfuscate the results produced with irrelevant special effects. However, techniques stand to be included that add additional visual benefit with minimum human effort. We place emphasis on techniques that produce visual geometry procedurally, and we believe that such methods contribute towards solving the problem of limited libraries of visual media. More advanced forms of model articulation, special effects, texturing, lighting and rendering are candidates for adding detail to virtual environments.

Future investigations also include applications that benefit from the technology presented in this research. We believe that these techniques are of use in teaching language concepts, as well as in automatically creating content for educational and entertainment purposes.

7.5 State of the art in text-to-graphics research

The fiction-to-animation system surpasses existing text-to-graphics research in its combined capability to handle large quantities of input with little restriction on sentence complexity, and produce multi-modal presentations consisting of a number of unique scenes. Our approach is superior to any single related approach in terms of the combination of its input and output capabilities, illustrated using comparative ratings in Figure 7.2. In this respect our fiction-to-animation system represents a significant contribution to the text-to-graphics domain.

Bibliography

Adorni, G., Manzo, M. D., and Giunchiglia, F. 1984. Natural language driven image generation. In *COLING '84: Proceedings of the 10th International Conference on Computational Linguistics* (Stanford, United States of America). Association for Computational Linguistics, Morristown, United States of America, 495–500.

Agichtein, E. and Gravano, L. 2000. Snowball: extracting relations from large plain-text collections. In *DL '00: Proceedings of the 5th ACM conference on Digital libraries* (San Antonio, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 85–94.

Apt, K. R. 1999. The essence of constraint propagation. *Theoretical Computer Science 221,* 1-2, 179–210.

Aseltine, J. H. 1999. WAVE: An incremental algorithm for information extraction. In *Proceedings of the AAAI Workshop on Machine Learning for Information Extraction* (Orlando, United States of America). Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America.

Atkinson, K. 2003. 12dicts. Website: http://wordlist.sourceforge.net/ [Accessed on 22 November 2005].

Atwell, E., Demetriou, G., Hughes, J., Schriffin, A., Souter, C., and Wilcock, S. 2000. A comparative evaluation of modern English corpus grammatical annotation schemes. *ICAME (International Computer Archive of Modern and Medieval English) Journal 24*, 7–23.

Atwell, E., Hughes, J., and Souter, C. 1994. AMALGAM: Automatic mapping among lexico-grammatical annotation models. In *The Balancing Act: Combining Symbolic and Statistical Approaches to Language - Proceedings of the ACL Workshop* (Las Cruces, New Mexico), J. Klavans, Ed. Association for Computational Linguistics, Morristown, United States of America, 21–28.

Back, M., Gold, R., and Kirsch, D. 1999. The SIT book: audio as affective imagery for interactive storybooks. In *CHI '99: Extended abstracts on Human factors in computing systems* (Pittsburgh, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 202–203.

Badler, N. I., Bindiganavale, R., Allbeck, J., Schuler, W., Zhao, L., and Palmer, M. 2000. Parameterized action representation for virtual human agents. *Embodied conversational agents*, 256–284.

Baldwin, B. 1997. CogNIAC: high precision coreference with limited knowledge and linguistic resources. In *Proceedings of the ACL'97/EACL'97 workshop on operational factors in practical, robust anaphora resolution* (Madrid, Spain). Association for Computational Linguistics, Morristown, United States of America, 38–45.

Basili, R., Pazienza, M., and Vindigni, M. 2000. Corpus-driven learning of event recognition rules. In *Proceedings of Machine Learning for Information Extraction workshop, held jointly with the ECAI2000* (Berlin, Germany).

Batnini, H., Michel, C., and Rueher, M. 2005. Mind the gaps: A new splitting strategy for consistency techniques. In *CP '05: Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming* (Barcelona, Spain). Lecture Notes in Computer Science, vol. 3709. Springer-Verlag, Berlin, Germany, 77–91.

Baykan, C. and Fox, M. S. 1991. Constraint satisfaction techniques for spatial planning. In *Intelligent CAD Systems III: Practical Experience and Evaluation*, P. ten Hagen and P. Veerkamp, Eds. Springer-Verlag, Berlin, Germany, 187–204.

Belhadj, F. 2007. Terrain modeling: a constrained fractal model. In *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (Grahamstown, South Africa). Association for Computing Machinery, ACM Press, New York, United States of America, 197–204.

Benhamou, F. 1995. Interval constraint logic programming. In *Constraint Programming: Basics and Trends*, A. Podelski, Ed. Springer-Verlag, Berlin, Germany, 1–21.

Benhamou, F. and Goualard, F. 2000. Universally quantified interval constraints. In *CP '00: Proceedings of the 6th International Conference on Principles and Practice of Constraint Programming* (Singapore). Lecture Notes In Computer Science, vol. 1894. Springer-Verlag, Berlin, Germany, 67–82.

Benhamou, F., Goualard, F., Granvilliers, L., and Puget, J. 1999. Revising hull and box consistency. In *ICLP '99: Proceedings of the International Conference on Logic Programming* (Las Cruces, New Mexico). MIT Press, Cambridge, United States of America, 230–244.

Benhamou, F., Goualard, F., Languénou, É., and Christie, M. 2004. Interval constraint solving for camera control and motion planning. *ACM Transactions on Computational Logic (TOCL) 5,* 4, 732–767.

Benhamou, F., McAllester, D., and van Hentenryck, P. 1994. CLP(intervals) revisited. In *ILPS '94: Proceedings of the 1994 International Symposium on Logic programming* (Santa Marherita Ligure, Italy). MIT Press, Cambridge, United States of America, 124–138.

Benhamou, F. and Older, W. J. 1997. Applying interval arithmetic to real, integer, and boolean constraints. *The Journal of Logic Programming 32,* 1, 1–24.

BENNETT, S. W., AONE, C., AND LOVELL, C. 1997. Learning to tag multilingual texts through observation. In *Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing* (Providence, United States of America), C. Cardie and R. Weischedel, Eds. Association for Computational Linguistics, Morristown, United States of America, 109–116.

BERSOT, O., EL GUEDJ, P., GODEREAUX, C., AND NUGUES, P. 1998. A conversational agent to help navigation and collaboration in virtual worlds. *Virtual Reality 3,* 1, 71–82.

BILLINGHURST, M., KATO, H., AND POUPYREV, I. 2001. The magicbook - moving seamlessly between reality and virtuality. *Computer Graphics and Applications 21,* 3, 2–4.

BINDIGANAVALE, R., SCHULER, W., ALLBECK, J. M., BADLER, N. I., JOSHI, A. K., AND PALMER, M. 2000. Dynamically altering agent behaviors using natural language instructions. In *AGENTS '00: Proceedings of the 4th international conference on Autonomous agents* (Catalonia, Spain). Association for Computing Machinery, ACM Press, New York, United States of America, 293–300.

BLENDER FOUNDATION. 2008. Blender manual. Website: http://wiki.blender.org/index.php/Manual [Accessed on 30 April 2008].

BOBERG, R. W. 1972. Generating line drawings from abstract scene descriptions. M.S. thesis, Massachusetts Institute of Technology, Cambridge, United States of America.

BONNEFOI, P. AND PLEMENOS, D. 1999. Object oriented constraint satisfaction for hierarchical declarative scene modeling. In *WSCG'99: Proceedings of the 7th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media* (Plzen, Czech Republic). University of West Bohemia Press.

BORNING, A., ANDERSON, R., AND FREEMAN-BENSON, B. N. 1996. Indigo: A local propagation algorithm for inequality constraints. In *Proceedings of the ACM Symposium on User Interface Software and Technology* (Seattle, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 129–136.

BORTHWICK, A., STERLING, J., AGICHTEIN, E., AND GRISHMAN, R. 1998. NYU: Description of the MENE named entity system as used in MUC-7. In *Proceedings of the 7th Message Understanding Conference* (Fairfax, United States of America).

BRILL, E. 1994. Some advances in transformation-based part of speech tagging. In *AAAI '94: Proceedings of the 12th national conference on Artificial intelligence* (Seattle,United States of America). Vol. 1. Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 722–727.

BRILL, E. AND WU, J. 1998. Classifier combination for improved lexical disambiguation. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics* (Montréal, Canada), C. Boitet and P. Whitelock, Eds. Morgan Kaufmann Publishers, San Francisco, United States of America, 191–195.

Bunescu, R. and Mooney, R. J. 2004. Collective information extraction with relational markov networks. In *ACL '04: Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics* (Barcelona, Spain). Association for Computational Linguistics, Morristown, United States of America, 438.

Buss, S. R. 2003. *3-D Computer Graphics: A Mathematical Introduction with OpenGL*. Cambridge University Press, Cambridge, United Kingdom.

Califf, M. E. and Mooney, R. J. 2003. Bottom-up relational learning of pattern matching rules for information extraction. *Journal of Machine Learning Research 4*, 177–210.

Calomeni, A. and Celes, W. 2006. Assisted and automatic navigation in black oil reservoir models based on probabilistic roadmaps. In *I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games* (Redwood City, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 175–182.

Cantor, J. and Valencia, P. 2004. *Inspired 3D short film production.* Thompson Course Technology PTR, Boston, United States of America.

Castaño, J., Zhang, J., and Pustejovsky, J. 2002. Anaphora resolution in biomedical literature. In *Proceedings of the International Symposium on Reference Resolution* (Alicante, Spain).

Català, N., Castell, N., and Martín, M. 2003. A portable method for acquiring information extraction patterns without annotated corpora. *Natural Language Engineering 9,* 2, 151–179.

Chabert, G., Trombettoni, G., and Neveu, B. 2005. Box-set consistency for interval-based constraint problems. In *SAC '05: Proceedings of the 2005 ACM Symposium on Applied Computing* (Santa Fe, New Mexico). Association for Computing Machinery, ACM Press, New York, United States of America, 1439–1443.

Chai, J. Y., Biermann, A. W., and Guinn, C. I. 1999. Two dimensional generalization in information extraction. In *AAAI '99/IAAI '99: Proceedings of the 16th national conference on Artificial intelligence and the 11th Innovative applications of artificial intelligence conference innovative applications of artificial intelligence* (Orlando, United States of America). Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 431–438.

Chieu, H. L. and Ng, H. T. 2002. A maximum entropy approach to information extraction from semi-structured and free text. In *AAAI '02: Proceedings of the 18th national conference on Artificial intelligence* (Edmonton, Canada). Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 786–791.

Chieu, H. L., Ng, H. T., and Lee, Y. K. 2003. Closing the gap: learning-based information extraction rivaling knowledge-engineering methods. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics* (Sapporo, Japan). Association for Computational Linguistics, Morristown, United States of America, 216–223.

Chinchor, N. and Marsh, E. 1998. MUC-7 information extraction task definition v.5.1. In *Proceedings of 7th Message Understanding Conference* (Fairfax, United States of America). Website: http://www-nlpir.nist.gov/related_projects/muc/proceedings/ie_task.html [Accessed on 19 May 2008].

Christie, M., Languénou, E., and Granvilliers, L. 2002. Modeling camera control with constrained hypertubes. In *CP '02: Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming* (Ithaca, United States of America). Springer-Verlag, Berlin, Germany, 618–632.

Chu, Y., Witten, I. H., Lobb, R., and Bainbridge, D. 2003. How to turn the page. In *Proceedings of the 3rd ACM/IEEE-CS joint conference on Digital Libraries* (Houston, United States of America). IEEE Computer Society, Washington DC, United States of America, 186–188.

Ciravegna, F. 2001. Adaptive information extraction from text by rule induction and generalisation. In *IJCAI: Proceedings of the 17th International Joint Conference on Artificial Intelligence* (Seattle, United States of America). Vol. 2. 1251–1256.

Clark, A. 2003. Pre-processing very noisy text. In *Proceedings of Workshop on Shallow Processing of Large Corpora* (Lancaster, United Kingdom). Corpus Linguistics.

Clay, S. R. and Wilhelms, J. 1996. Put: Language-based interactive manipulation of objects. *IEEE Computer Graphics and Applications 16,* 2, 31–39.

Cleary, J. G. 1987. Logical arithmetic. *Future Computing Systems 2*, 125–149.

Cohen, W. W. and Sarawagi, S. 2004. Exploiting dictionaries in named entity extraction: combining semi-Markov extraction processes and data integration methods. In *KDD '04: Proceedings of the 10th ACM SIGKDD international conference on Knowledge discovery and data mining* (Seattle, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 89–98.

Coyne, B. and Sproat, R. 2001. Wordseye: an automatic text-to-scene conversion system. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (Los Angeles, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 487–496.

de Vries, B. and Jessurun, A. 2000. Using 3D geometric constraints in architectural design support systems. In *Proceedings of the 8th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media* (Plzen, Czech Republic).

Déjean, H. 2000. How to evaluate and compare tagsets? A proposal. In *LREC 2000: Proceedings of the 2nd International Conference on Language Resources and Evaluation* (Athens, Greece).

Déjean, H. 2002. Learning rules and their exceptions. *Journal of Machine Learning Research 2*, 669–693.

DIMITROV, M. 2002. A light-weight approach to coreference resolution for named entites in text. M.S. thesis, University of Sofia, Sofia, Republic of Bulgaria.

DOLGOV, Y. G. 2005. Developing interval global optimization algorithms on the basis of branch-and-bound and constraint propagation methods. *Reliable Computing 11,* 5, 343–358.

DRUCKER, S. M. AND ZELTZER, D. 1994. Intelligent camera control in a virtual environment. In *Proceedings of the Graphics Interface Conference* (Banff, Canada), W. A. Davis and B. Joe, Eds. Canadian Human-Computer Communications Society, 190–199.

EGGES, A., NIJHOLT, A., AND NUGUES, P. 2001. Generating a 3D simulation of a car accident from a formal description: the CarSim system. In *ICAV3D: Proceedings of the International Conference on Augmented, Virtual Environments and Three-Dimensional Imaging* (Ornos, Mykonos). Mykonos, Greece.

EYSENCK, M. W. AND KEANE, M. T. 2000. *Cognitive Psychology: A student's handbook.* Psychology Press Ltd, East Sussex, United Kingdom.

FELLBAUM, C., Ed. 1998. *WordNet: An Electronic Lexical Database.* MIT Press, Cambridge, United States of America.

FLESCH, R. 1949. *The Art of Readble Writing.* Harper and Row, New York, United States of America.

FOSKEY, M., GARBER, M., LIN, M., AND MANOCHA, D. 2001. A Voronoi-based hybrid motion planner for rigid bodies. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE Computer Society, Washington DC, United States of America, 55–60.

FRANCIS, W. N. AND KUCERA, H. 1979. *Manual of information to accompany a standard corpus of present-day edited American English, for use with digital computers*, 2nd ed. Brown University, Providence, United States of America. Website: http://icame.uib.no/brown/bcm.html [accessed on 10 September 2007].

FREITAG, D. 1998. Toward general-purpose learning for information extraction. In *COLING '98: Proceedings of the 17th international conference on Computational linguistics* (Montréal, Canada). Association for Computational Linguistics, Morristown, United States of America, 404–408.

FREITAG, D. 2000. Machine learning for information extraction in informal domains. *Machine Learning 39,* 2-3, 169–202.

FREITAG, D. AND MCCALLUM, A. 2000. Information extraction with HMM structures learned by stochastic optimization. In *Proceedings of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence* (Austin, Texas). Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 584–589.

GARBER, M. AND LIN, M. C. 2002. Constraint-based motion planning for virtual prototyping. In *SMA '02: Proceedings of the 7th ACM symposium on Solid modeling and applications*

(Saarbr ucken, Germany). Association for Computing Machinery, ACM Press, New York, United States of America, 257–264.

Garber, M. and Lin, M. C. 2003. *Constraint-based motion planning using Voronoi diagrams.* Algorithmic Foundations of Robotics V, vol. 7. Springer-Verlag, Berlin, Germany, 541–558.

GATE. 2005. ANNIE: A nearly-new information extraction system. Website: http://gate.ac.uk/ie/annie.html [Accessed on 22 November 2005].

Gayle, R., Lin, M. C., and Manocha, D. 2005. Constraint-based motion planning of deformable robots. In *ICRA '05: Proceedings of the 2005 IEEE International Conference on Robotics and Automation* (Barcelona, Spain). IEEE Computer Society, Washington DC, United States of America, 1046–1053.

Ge, S. S. and Cui, Y. J. 2002. Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots 13,* 3, 207–222.

Giménez, J. and Marquez, L. 2003. Fast and accurate part-of-speech tagging: The SVM approach revisited. In *RANLP '03: Proceedings of the workshop on Recent Advances in Natural Language Processing* (Borovets, Bulgaria). 153–163.

Glass, K. and Bangay, S. 2005. Evaluating parts-of-speech taggers for use in a text-to-scene conversion system. In *SAICSIT '05: Proceedings of the 2005 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries* (White River, South Africa). Association for Computing Machinery, ACM Press, New York, United States of America, 20–28.

Glass, K. and Bangay, S. 2006. Hierarchical rule generalisation for speaker identification in fiction books. In *SAICSIT '06: Proceedings of the 2006 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries* (Cape Town, South Africa). Association for Computing Machinery, ACM Press, New York, United States of America, 31–40.

Glass, K. and Bangay, S. 2007a. Constraint-based conversion of fiction text to a time-based graphical representation. In *SAICSIT '07: Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries* (Fish River, South Africa). Association for Computing Machinery, ACM Press, New York, United States of America, 19–28.

Glass, K. and Bangay, S. 2007b. Evaluating and improving morpho-syntactic classification over multiple corpora using pre-trained, "off-the-shelf", parts-of-speech tagging tools. In *PRASA '07: Proceedings of the 18th Annual Symposium of the Pattern Recognition Association of South Africa* (Pietermaritzburg, South Africa). 19–24. To appear in the South African Computer Journal.

Glass, K. and Bangay, S. 2007c. A naïve salience-based method for speaker identification in fiction books. In *PRASA '07: Proceedings of the 18th Annual Symposium of the Pattern Recognition Association of South Africa* (Pietermaritzburg, South Africa). 1–6.

Glass, K. and Bangay, S. 2008. Automating the creation of 3D animation from annotated fiction text. In *IADIS-CGV '08: Proceedings of the IADIS International conference on Computer Graphics and Visualization* (Amsterdam, The Netherlands). International Association for Development of the Information Society, 3–10.

Glass, K., Bangay, S., and Alcock, B. 2007. Mechanisms for multimodality: taking fiction to another dimension. In *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (Grahamstown, South Africa). Association for Computing Machinery, ACM Press, New York, United States of America, 135–144.

Granvilliers, L. and Hains, G. 2000. A conservative scheme for parallel interval narrowing. *Information Processing Letters 74,* 3-4, 141–146.

Grefenstette, G. and Tapanainen, P. 1994. What is a word, what is a sentence? Problems of tokenization. In *Proceedings of the 3rd International Conference on Computational Lexicography* (Budapest, Hungary). 79–87.

Grover, C., Matheson, C., Mikheev, A., and Moens, M. 2000. LT TTT - a flexible tokenisation tool. In *LREC 2000: Proceedings of the 2nd International Conference on Language Resources and Evaluation* (Athens, Greece).

Gunning, R. 1952. *The Technique of Clear Writing.* McGraw-Hill Book Company, New York, United States of America.

Harabagiu, S. M. and Maiorano, S. J. 2000. Acquisition of linguistic patterns for knowledge-based information extraction. In *LREC '00: Proceedings of the 2nd International Conference on Language Resources and Evaluation* (Athens, Greece).

Hauge, M. 1988. *Writing screen-plays that sell.* McGraw-Hill Book Company, New York, United States of America.

He, L., Cohen, M. F., and Salesin, D. H. 1996. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. *Computer Graphics 30*, 217–224.

Hickey, T. J., Ju, Q., and van Emden, M. H. 2001. Interval arithmetic: From principles to implementation. *Journal of the ACM 48,* 5, 1038–1068.

Hickey, T. J., van Emden, M. H., and Wu, H. 1998. A unified framework for interval constraints and interval arithmetic. In *CP '98: Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming* (Pisa, Italy). Springer-Verlag, Berlin, Germany, 250–264.

Hong, L., Muraki, S., Kaufman, A., Bartz, D., and He, T. 1997. Virtual voyage: interactive navigation in the human colon. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques* (Los Angeles, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 27–34.

Hood, M. 2004. Creating a voice for festival speech synthesis system. Tech. rep., Virtual Reality Special Interest Group, Computer Science Department, Rhodes University, Grahamstown, South Africa. November.

Hudson, D. 2005. Dependency grammar, a five session course. In *ESSLLI 2005: Proceedings of the European Summer School in Logic, Language and Information* (Edinburgh, Scotland).

Huffman, S. B. 1996. Learning information extraction patterns from examples. In *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing*. Vol. 1040. Springer-Verlag, Berlin, Germany, 246–260.

Huyer, W. and Neumaier, A. 1999. Global optimization by multilevel coordinate search. *Journal of Global Optimization 14,* 4, 331–355.

Jardillier, F. and Languénou, E. 1998. Screen-space constraints for camera movements: the virtual cameraman. *Computer Graphics Forum 17,* 3, 175–186.

Järvinen, T. and Tapanainen, P. 1998. Towards an implementable dependency grammar. In *COLING-ACL '98: Proceedings of the Workshop on Processing of Dependency-based Grammars* (Montréal, Canada), S. Kahane and A. Polguère, Eds.

Jaulin, L. and Walter, É. 1993. Set inversion via interval analysis for nonlinear bounded-error estimation. *Automatica 29,* 4, 1053–1064.

Jaulin, L. and Walter, É. 1996. Guaranteed tuning, with application to robust control and motion planning. *Automatica 32,* 9, 1217–1221.

Johansson, R., Berglund, A., Danielsson, M., and Nugues, P. 2005. Automatic text-to-scene conversion in the traffic accident domain. In *IJCAI '05: Proceedings of the 19th International Joint Conference on Artificial Intelligence* (Edinburgh, Scotland). 1073–1078.

Johansson, S., Atwell, E., Garside, R., and Leech, G. 1986. *The Tagged LOB Corpus: Users' manual.* ICAME, The Norwegian Computing Centre for the Humanities, Bergen, Norway. Website: http://www.comp.lancs.ac.uk/ucrel/local/lob/ [accessed on 10 September 2007].

Johansson, S., Leech, G. N., and Goodluck, H. 1978. *Manual of information to accompany the Lancaster-Oslo/Bergen corpus of British English, for use with digital computers.* Department of English, University of Oslo, Oslo, Norway. Website: http://khnt.hit.uib.no/icame/manuals/lob/ [accessed on 10 September 2007].

Joshi, D., Wang, J. Z., and Li, J. 2004. The story picturing engine: finding elite images to illustrate a story using mutual reinforcement. In *MIR 04: Proceedings of the 6th ACM SIGMM international workshop on Multimedia information retrieval* (New York, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 119–126.

Kahn, K. 1979. Creation of computer animation from story descriptions. Ph.D. thesis, Massachusetts Institute of Technology, Cambridge, United States of America.

Kavraki, L. and Latombe, J. 1994. Randomized preprocessing of configuration for fast path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation.* Vol. 3. IEEE Computer Society, Washington DC, United States of America, 2138–2145.

Kennedy, C. and Boguraev, B. 1996. Anaphora for everyone: Pronominal anaphora resolution without a parser. In *COLING '96: Proceedings of the 16th International Conference on Computational Linguistics* (Copenhagen, Denmark). Association for Computational Linguistics, Morristown, United States of America, 113–118.

Kim, J. and Moldovan, D. I. 1995. Acquisition of linguistic patterns for knowledge-based information extraction. *IEEE Transactions on Knowledge and Data Engineering 7,* 5, 713–724.

Koga, Y., Kondo, K., Kuffner, J., and Latombe, J. 1994. Planning motions with intentions. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques* (Orlando, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 395–408.

Kudo, T. and Matsumoto, Y. 2001. Chunking with support vector machines. In *NAACL '01: Proceedings of the 2nd meeting of the North American Chapter of the Association for Computational Linguistics on Language technologies* (Pittsburgh, United States of America).

Kuffner Jr., J. J. 1998. Goal-directed navigation for animated characters using real-time path planning and control. In *CAPTECH '98: Proceedings of the International Workshop on Modelling and Motion Capture Techniques for Virtual Environments* (Geneva, Switzerland), N. Magnenat-Thalmann and D. Thalmann, Eds. Lecture Notes In Computer Science, vol. 1537. Springer-Verlag, Berlin, Germany, 171–186.

Kwaiter, G., Gaildrat, V., and Caubet, R. 1998. Modelling with constraints: A bibliographical survey. In *IV '98: Proceedings of the International Conference on Information Visualisation* (London, United Kindom). IEEE Computer Society, Washington DC, United States of America, 211–220.

Lappin, S. and Leass, H. J. 1994. An algorithm for pronominal anaphora resolution. *Computational Linguistics 20,* 4, 535–561.

Latombe, J. 1999. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research 18,* 11, 1119–1128.

Le Roux, O. and Gaildrat, V. 2003. Constraint-based 3D isothetic object layout for declarative scene modeling. In *CISST '03: Proceedings of the International Conference on Imaging Science, Systems, and Technology* (Las Vegas, United States of America). CSREA press, Las Vegas, United States of America, 320–326.

Lhomme, O., Gotlieb, A., and Rueher, M. 1998. Dynamic optimization of interval narrowing algorithms. *Journal of Logic Programming 37,* 1-3, 165–183.

Li, Y. and Bontcheva, K. 2007. Hierarchical, perceptron-like learning for ontology-based information extraction. In *WWW '07: Proceedings of the 16th international conference on World*

Wide Web (Banff, Canada). Association for Computing Machinery, ACM Press, New York, United States of America, 777–786.

Lu, R. and Zhang, S. 2002. *Automatic Generation of Computer Animation: using AI for movie animation.* Lecture Notes in Computer Science, vol. 2160. Springer-Verlag, Berlin, Germany.

Ma, M. 2006. Automatic conversion of natural language to 3D animation. Ph.D. thesis, University of Ulster, Derry, Ireland.

Ma, M. and McKevitt, P. 2003. Semantic representation of events in 3D animation. In *IWCS-5: Proceedings of The 5th International Workshop on Computational Semantics* (Tilburg, The Netherlands), I. v. d. S. H. Bunt and R. Morante, Eds. 253–281.

Ma, M. and McKevitt, P. 2004a. Interval relations in visual semantics of verbs. *Artificial Intelligence Review 21,* 3-4, 293–316.

Ma, M. and McKevitt, P. 2004b. Using lexical knowledge of verbs in language-to-vision applications. In *AICS '04: Proceedings of the 15th Artificial Intelligence and Cognitive Science Conference* (Castlebar, Ireland), L. McGinty and B. Crean, Eds. Galway-Mayo Institute of Technology, 255–264.

Ma, M. and McKevitt, P. 2004c. Visual semantics and ontology of eventive verbs. In *IJCNLP-04: Proceedings of the 1st International Joint Conference on Natural Language Processing* (Sanya, China), K.-Y. Su and J.-I. Tsujii, Eds. 278–285.

Mandler, J. M. and Johnson, N. S. 1977. Remembrance of things parsed: Story structure and recall. *Cognitive Psychology 9*, 111–151.

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. 1994. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics 19,* 2, 313–330.

Màrquez, L., Rodríguez, H., Carmona, J., and Montolio, J. 1999. Improving POS tagging using machine-learning techniques. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora* (Maryland, United States of America). Association for Computational Linguistics, Morristown, United States of America, 53–62.

McDonald, D. D. 1996. Internal and external evidence in the identification and semantic categorization of proper names. In *Corpus processing for lexical acquisition*, B. Boguraev and J. Pustejovsky, Eds. MIT Press, Cambridge, United States of America, 21–39.

Mikheev, A. 2002. Periods, capitalized words, etc. *Computational Linguistics 28,* 3, 289–318.

Minsky, M. 1975. *A framework for representing knowledge.* The Psychology of Computer Vision. McGraw-Hill Book Company, New York, United States of America, Chapter 6, 211–277.

Mitchell, D. P. 1991. Three applications of interval analysis in computer graphics. In *SIGGRAPH '91 Course Notes: Frontiers of Rendering* (Las Vegas, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 1–13.

Mitkov, R. 1998. Robust pronoun resolution with limited knowledge. In *COLING-ACL '98: Proceedings of the 18th International Conference on Computational Linguistics* (Montréal, Canada). Association for Computational Linguistics, Morristown, United States of America, 869–875.

Mitkov, R., Evans, R., and Oruasan, C. 2002. A new, fully automatic version of Mitkov's knowledge-poor pronoun resolution method. In *CICLing-2002: Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics* (Mexico City, Mexico). Lecture Notes in Computer Science, vol. 2276. Springer, Verlag, Berlin, Germany, 168–186.

Moore, R. E. 1966. *Interval Analysis.* Prentice-Hall, Inc., Englewood Cliffs, United States of America.

Mukerjee, A., Gupta, K., Nautiyal, S., M.P, M. S., and Mishra, N. 2000. Conceptual description of visual scenes from linguistic models. *Image and Vision Computing 18,* 2, 173–187.

Muller, P., Wonka, P., Haegler, S., Ulmer, A., and Gool, L. V. 2006. Procedural modeling of buildings. In *SIGGRAPH '06: Proceedings of the International Conference on Computer Graphics and Interactive Techniques* (Boston, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 614–623.

Musgrave, F. K., Kolb, C. E., and Mace, R. S. 1989. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques* (Boston, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 41–50.

Muslea, I. 1999. Extraction patterns for information extraction tasks: A survey. In *Proceedings of the AAAI Workshop on Machine Learning for Information Extraction* (Orlando, United States of America). Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 1–6.

Narayanan, A., Manuel, D., Ford, L., Tallis, D., and Yazdani, M. 1995. Language visualisation: Applications and theoretical foundations of a primitive-based approach. *Artificial Intelligence Review 9,* 2-3, 215–235.

Nasukawa, T. 1994. Robust method of pronoun resolution using full-text information. In *COLING '94: Proceedings of the 15th conference on Computational linguistics* (Kyoto, Japan). Association for Computational Linguistics, Morristown, United States of America, 1157–1163.

Neumaier, A. 1990. *Interval Methods for Systems of Equations.* Encyclopedia of Mathematics and its Applications, vol. 37. Cambridge Univeristy Press, Cambridge, United Kingdom.

Nieuwenhuisen, D. and Overmars, M. H. 2003. Motion planning for camera movements in virtual environments. Tech. rep., Utrecht University, Utrecht, The Netherlands. January.

Nugues, P., Dupuy, S., and Egges, A. 2003. Information extraction to generate visual simulations of car accidents from written descriptions. In *ICCSA '03: Proceedings of the 2003 International Conference on Computational Science and its Applications* (San Diego, United States of America). Lecture Notes in Computer Science, vol. 2667. Springer-Verlag, Berlin, Germany, 31–40.

Older, W. and Vellino, A. 1990. Extending prolog with constraint arithmetic on real intervals. In *Proceedings of the IEEE Canadian Conference on Computer and Electrical Engineering* (Ottawa, Canada). IEEE Computer Society, Washington DC, United States of America.

Oliveira, J. B. and De Figueiredo, L. H. 2003. Robust approximation of offsets, bisectors, and medial axes of plane curves. *Reliable Computing 9,* 2, 161–175.

Overmars, M. H. 1992. A random approach to motion planning. Tech. rep., Department of Computer Science, Utrecht University, Utrecht, The Netherlands. October.

Overmars, M. H. and vSvestka, P. 1995. A probabilistic learning approach to motion planning. In *WAFR '95: Proceedings of the workshop on Algorithmic foundations of robotics* (San Francisco, United States of America). A. K. Peters, Ltd., Natick, United States of America, 19–37.

Palmer, D. D. and Hearst, M. A. 1994. Adaptive sentence boundary disambiguation. In *Proceedings of the 4th ACL Conference on Applied Natural Language Processing* (Stuttgart, Germany). Morgan Kaufmann Publishers, San Francisco, United States of America, 78–83.

Parish, Y. I. H. and Muller, P. 2001. Procedural modeling of cities. In *SIGGRAPH '01: Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (Los Angeles, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 301–308.

Patwardhan, S. and Riloff, E. 2007. Effective information extraction with semantic affinity patterns and relevant regions. In *EMNLP-CoNLL: Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning* (Prague, Czech Republic). Association for Computational Linguistics, Morristown, United States of America, 717–727.

Pettré, J., Laumond, J., and Siméon, T. 2003. A 2-stages locomotion planner for digital actors. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Aire-la-Ville, Switzerland). Eurographics Association, 258–264.

Pettré, J., Laumond, J., and Siméon, T. 2003. 3D collision avoidance for digital actors locomotion. In *IROS '03: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems.* Vol. 1. IEEE Computer Society, Washington DC, United States of America, 400–405.

Phillips, W. and Riloff, E. 2007. Exploiting role-identifying nouns and expressions for information extraction. In *EMNLP '07:Proceedings of the 2007 Conference on Empirical Methods in Natural Language Processing)* (Borovets, Bulgaria).

Piesk, J. and Trogemann, G. 1997. Animated interactive fiction: Storytelling by a conversational virtual actor. In *VSMM '97: Proceedings of the 1997 International Conference on*

Virtual Systems and MultiMedia (Geneva, Switzerland). IEEE Computer Society, Washington DC, United States of America, 100–108.

PUGET, J. 1994. A C++ implementation of CLP. In *Proceedings of the 2nd Singapore International Conference on Intelligent Systems.* Singapore.

QUINLAN, J. R. 1986. Induction of decision trees. *Machine Learning 1,* 1, 81–106.

QUINLAN, J. R. 1990. Learning logical definitions from relations. *Machine Learning 5,* 3, 239–266.

RAMSHAW, L. AND MARCUS, M. 1995. Text chunking using transformation-based learning. In *Proceedings of the Third Workshop on Very Large Corpora* (Cambridge, United States of America), D. Yarovsky and K. Church, Eds. Association for Computational Linguistics, Morristown, United States of America, 82–94.

RATSCHAN, S. 2006. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic 7,* 4, 723–748.

RATSCHAN, S. 2008. Applications of quantified constraint solving over the reals bibliography. Website: http://uivt1.cs.cas.cz/ ratschan/appqcs.html [accessed on 21 March 2008].

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Computer Graphics Forum 21,* 3, 279.

RILOFF, E. 1993. Automatically constructing a dictionary for information extraction tasks. In *AAAI '93: Proceedings of the 11th National Conference on Artificial Intelligence* (Washington, D.C., United States of America). Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 811–816.

RILOFF, E. 1996. Automatically generating extraction patterns from untagged text. In *AAAI '96: Proceedings of the 13th National Conference on Artificial Intelligence* (Portland, United States of America). Vol. 2. Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 1044–1049.

SALOMON, B., GARBER, M., LIN, M. C., AND MANOCHA, D. 2003. Interactive navigation in complex environments using path planning. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics* (Monterey, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 41–50.

SAMPSON, G. 2007. The SUSANNE analytic scheme. Website: http://www.grsampson.net/RSue.html [accessed on 10 September 2007].

SANCHEZ, S., LE ROUX, O., LUGA, H., AND GAILDRAT, V. 2003. Constraint-based 3D-object layout using a genetic algorithm. In *3IA '03: The 6th International Conference on Computer Graphics and Artificial Intelligence* (Limoges, France). Laboratoire XLIM - Université de Limoges.

SCHANK, R. C. 1972. Conceptual dependency: A theory of natural language understanding. *Cognitive Psychology 3*, 552–631.

SCHANK, R. C. 1973. The fourteen primitive actions and their inferences. Tech. rep., Stanford University, Stanford, United States of America.

SCHMID, H. 1994. Probabilistic part-of-speech tagging using decision trees. Tech. rep., University of Stuttgart, Stuttgart, Germany.

SCHWARTZ, L. S. 1963. *Principles of coding, filtering, and information theory.* Spartan Books, Inc., Baltimore, United States of America.

SEVERSKY, L. M. AND YIN, L. 2006. Real-time automatic 3D scene generation from natural language voice and text descriptions. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia* (Santa Barbara, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 61–64.

SHEN, L., SATTA, G., AND JOSHI, A. 2007. Guided learning for bidirectional sequence classification. In *ACL 2007: Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics* (Prague, Czech Republic). Association of Computational Linguistics, Association of Computational Linguistics, Prague, Czech Republic, 760–767.

SHINYAMA, Y. AND SEKINE, S. 2006. Preemptive information extraction using unrestricted relation discovery. In *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics* (New York, United States of America). Association for Computational Linguistics, Morristown, United States of America, 304–311.

SHINYAMA, Y., TOKUNAGA, T., AND TANAKA, H. 2000. "Kairai" - software robots understanding natural language. In *Third International Workshop on Human-Computer Conversation* (Bellagio, Italy). 158–163.

SIMMONS, R. F. 1975. The clowns microworld. In *TINLAP '75: Proceedings of the 1975 workshop on Theoretical issues in natural language processing* (Cambridge, Massachusetts). Association for Computational Linguistics, Morristown, United States of America, 17–19.

SNYDER, J. M. 1992. Interval analysis for computer graphics. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques* (Chicago, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 121–130.

SNYDER, J. M., WOODBURY, A. R., FLEISCHER, K., CURRIN, B., AND BARR, A. H. 1993. Interval methods for multi-point collisions between time-dependent curved surfaces. In *SIGGRAPH '93: Proceedings of the 20th annual conference on Computer graphics and interactive techniques* (Anaheim, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 321–334.

SODERLAND, S. 1999. Learning information extraction rules for semi-structured and free text. *Machine Learning 34,* 1-3, 233–272.

SODERLAND, S. G. 1997. Learning text analysis rules for domain-specific natural language processing. Ph.D. thesis, University of Massachusetts, Amherst, United States of America.

SPROAT, R. 2001. Inferring the environment in a text-to-scene conversion system. In *K-CAP '01: Proceedings of the international conference on Knowledge capture* (Victoria, Canada). Association for Computing Machinery, ACM Press, New York, United States of America, 147–154.

STEVENSON, M. AND GREENWOOD, M. A. 2005. A semantic approach to ie pattern induction. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics* (Ann Arbor, United States of America). Association for Computational Linguistics, Morristown, United States of America, 379–386.

SUDO, K., SEKINE, S., AND GRISHMAN, R. 2003. An improved extraction pattern representation model for automatic ie pattern acquisition. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics* (Sapporo, Japan). Association for Computational Linguistics, Morristown, United States of America, 224–231.

TAKASHIMA, Y., SHIMAZU, H., AND TOMONO, M. 1987. Story driven animation. In *CHI '87: Proceedings of the SIGCHI/GI conference on Human factors in computing systems and graphics interface* (Toronto, Canada). Association for Computing Machinery, ACM Press, New York, United States of America, 149–153.

TAPANAINEN, P. 1999. Parsing in two frameworks: finite-state and functional dependency grammar. Ph.D. thesis, University of Helsinki, Helsinki, Finland.

TAPANAINEN, P. AND JÄRVINEN, T. 1997. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing* (Washington D.C., United States of America). Association for Computational Linguistics, Morristown, United States of America, 64–71.

TEUFEL, S. 1995. A support tool for tagset mapping. In *EACL95: Proceedings of the Workshop SIGDAT* (Dublin, Ireland). European Chapter of the Association for Computational Linguistics.

TEUFEL, S., SCHMID, H., ILEID, U., AND SCHILLER, A. 1996. Study of the relation between tagsets and taggers. Tech. rep., Expert Advisory Group on Language Engineering Standards, University of Stuttgart, Stuttgart, Germany. May. EAGLES Document EAG-CLWG-TAGS/V.

TOTH, D. L. 1985. On ray tracing parametric surfaces. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (San Francisco, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 171–179.

TOUTANOVA, K. AND MANNING, C. 2000. Enriching the knowledge sources used in a maximum entropy part-of-speech tagger. In *Proceedings of the 2000 Joint SIGDAT conference on Empirical methods in natural language processing and very large corpora: held in conjunction with the 38th Annual Meeting of the Association for Computational Linguistics* (Hong Kong, China). Vol. 13. Association for Computational Linguistics, Morristown, United States of America, 63–70.

TOUTANOVA, K., MANNING, C., KLEIN, D., AND SINGER, Y. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL 2003: Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics* *on Human Language Technology* (Edmonton, Canada). Vol. 1. Association for Computational Linguistics, Morristown, United States of America, 252–259.

Tufis, D. and Mason, O. 1998. Tagging Romanian texts: a case study for QTAG, a language independent probabilistic tagger. In *LREC 1998: Proceedings of the 1st International Conference on Language Resources and Evaluation* (Granada, Spain).

Turmo, J., Ageno, A., and Català, N. 2006. Adaptive information extraction. *ACM Computing Surveys 38,* 2, 4.

Turmo, J. and Rodriguez, H. 2002. Learning rules for information extraction. *Natural Language Engineering 8,* 3, 167–191.

van den Berg, J. and Overmars, M. 2006. Planning the shortest safe path amidst unpredictably moving obstacles. In *WAFR '06: Proceedings of the Workshop on Algorithmic Foundations of Robotics* (New York, United States of America).

van Halteren, H., Daelemans, W., and Zavrel, J. 2001. Improving accuracy in word class tagging through the combination of machine learning systems. *Computational Linguistics 27,* 2, 199–229.

van Halteren, H., Zavrel, J., and Daelemans, W. 1998. Improving data driven wordclass tagging by system combination. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics* (Montréal, Canada), C. Boitet and P. Whitelock, Eds. Morgan Kaufmann Publishers, San Francisco, California, 491–497.

van Hentenryck, P., McAllester, D., and Kapur, D. 1997. Solving polynomial systems using a branch and prune approach. *SIAM Journal on Numerical Analysis 34,* 2, 797–827.

Webber, B., Badler, N., Di Eugenio, B., Geib, C., Levison, L., and Moore, M. 1995. Instructions, intentions and expectations. *Artificial Intelligence 73,* 1-2, 253–269.

Winograd, T. 1972. *Understanding Natural Language.* Edinburgh University Press, Edinburgh, United Kingdom.

Witkin, A. and Kass, M. 1988. Spacetime constraints. In *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (Atlanta, United States of America). Association for Computing Machinery, ACM Press, New York, United States of America, 159–168.

Xu, K., Stewart, J., and Fiume, E. 2002. Constraint-based automatic placement for scene composition. In *Proceedings of Graphics Interface* (Calgary, Canada). 25–34.

Xun, E., Zhou, M., and Huang, C. 2000. A unified statistical model for the identification of English BaseNP. In *ACL 2000: Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics* (Hong Kong, China). Association for Computational Linguistics, Morristown, United States of America.

YAMADA, A., YAMAMOTO, T., IKEDA, H., NISHIDA, T., AND DOSHITA, S. 1992. Reconstructing spatial image from natural language texts. In *COLING '92: Proceedings of the 14th International Conference on Computational Linguistics* (Nantes, France). Vol. 4. Association of Computational Linguistics, 1279–1283.

YANGARBER, R. 2003. Counter-training in discovery of semantic patterns. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics* (Sapporo, Japan). Association for Computational Linguistics, Morristown, United States of America, 343–350.

ZENG, X., MEHDI, Q. H., AND GOUGH, N. E. 2003. Shape of the story: Story visualization techniques. In *IV '03: Proceedings of the 7th International Conference on Information Visualization* (London, United Kingdom).

ZENG, X., MEHDI, Q. H., AND GOUGH, N. E. 2005. From visual semantic parameterization to graphic visualization. In *IV '05: Proceedings of the 9th International Conference on Information Visualisation* (London, United Kingdom). IEEE Computer Society, Washington DC, United States of America, 488–493.

ZHANG, J., BLACK, A., AND SPROAT, R. 2003. Identifying speakers in children's stories for speech synthesis. In *Proceedings of EUROSPEECH '03* (Geneva, Switzerland). Geneva, Switzerland, 2041–2044.

ZHU, X., GOLDBERG, A., ELDAWY, M., DYER, C., AND STROCK, B. 2007. A text-to-picture synthesis system for augmenting communication. In *AAAI-07: The Integrated Intelligence Track of the 22nd AAAI Conference on Artificial Intelligence* (Vancouver, Canada). Association for the Advancement of Artificial Intelligence, AAAI Press, Menlo Park, United States of America, 1590–1596.

Appendix A

Properties of raw text

We impose the following requirements regarding the input text:

- Characters that occur in the input are sourced only from the standard ASCII domain;
- Paragraphs in the text are separated by a blank line;
- Punctuation that is not part of a recognized set (listed in Table A.1) does not occur in the text;
- Direct-speech is indicated by inverted commas (“...”) only;
- Direct-speech which spans multiple paragraphs only uses a single set of inverted commas, at the start of the quote, and after the last token inside the quote.

It is acknowledged that the raw text needs pre-processing to ensure conformance to the above requirements. Pre-processing is achieved using existing regular expression and command-line tools.

Punctuation	--- -- ... ; : , @ # $ % ( ) “ ” ‘ ’ ~ . ! ? { } [ ]
Apostrophe	n’t ’s ’re ’m ’ve ’d ’ll ’
Sentence Terminators	... . ! ?

Table A.1: Recognised punctuation, apostrophe and sentence termination symbols.

Appendix B

Coarse tag-set and mappings

B.1 Coarse tag-set

The coarse tag-set used by the fiction-to-animation system is listed in Table B.1. It is a derivative of the Penn tag-set (Marcus et al., 1994).

B.2 Mappings to the coarse tag-set

Mappings between the coarse tag-set and the Penn, LOB and SUSANNE tagsets are listed in Table B.2. The * symbol denotes a wild-card.

	Tag	Description
1.	CC	Co-ordinating Conjunction
2.	CD	Cardinal Number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	MD	Modal
9.	NN	Noun
10.	NNP	Proper Noun
11.	PRP	Pronoun
12.	RB	Adverb
13.	RP	Particle
14.	TO	To
15.	UH	Interjection
16.	VB	Verb
17.	VBD	Verb, past tense
18.	VBG	Verb, gerund or present participle
19.	VBN	Verb, past participle
20.	VBZ	Verb, third person singular present
21.	WDT	Wh-determiner
22.	WP	Wh-pronoun
23.	WRB	Wh-adverb
24.	@COPY@	*Punctuation*

Table B.1: Coarse tag-set, adapted from the Penn tag-set.

Coarse Tag	Penn Treebank	LOB	SUSANNE
CC	CC	CC*	CC*
CD	CD	CD*; OD*	MC
DT	DT; PDT	AT*; DT*	AT*
EX	EX	EX	EX
FW	FW	&FO	FW*
IN	IN	CS*; IN*	I*
JJ	JJ; JJR; JJS	AP*; JJ*	J*
MD	MD	MD	MD*; VM*
NN	NN; NNS	NN*; NR*; PN*	NN*
NNP	NNP; NNPS	JNP; NNP*; NP*	NP*
PRP	PRP; PRP$	PP*	PN*; PP*; APP*
RB	RB; RBR; RBS	ABL; Q*; R*	R*
RP	RP	RP	FB; RP*
TO	TO	TO*	TO; IIt
UH	UH	UH	UH*
VB	VB	BE; DO; HV; VB*	VB0; VD0; VV0*
VBD	VBD	BED; BEDZ; DOD; HVD; VBD	VBDZ; VBDR; VDD; VHD; VVD*
VBG	VBG; VBP	BEG; BEM; BEP; HVG; VBG	VBG; VDG; VHG; VVG*; VBR; VBM; VH0
VBN	VBN	BEN; HVN; VBN;	VBN; VDN; VHN; VVN*
VBZ	VBZ	BEZ; DOZ; HVZ; VBZ	VBZ; VDZ; VHZ; VVZ*
WDT	WDT	WDT*	DDQ*
WP	WP; WP$	WP*	PNQ*
WRB	WRB	WRB	RRQ*
@COPY@	*Punctuation*	*Punctuation*	Y*; G*

Table B.2: Mappings between the coarse tag-set and the Penn, LOB and SUSANNE tag-sets.

Appendix C

Algorithms in quantified constraint solving

C.1 Methods for constraint propagation

Constraint propagation is achieved over constraints using interval arithmetic and narrowing algorithms. Each is based on the *fixpoint* algorithm.

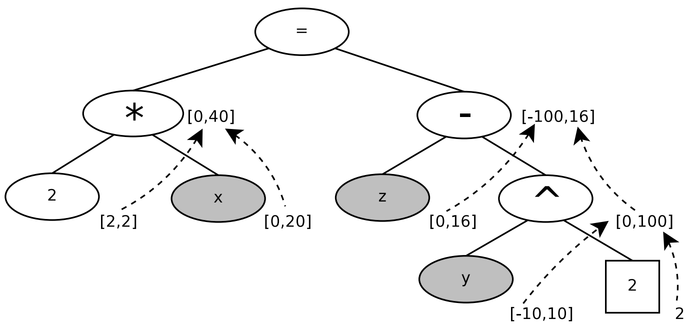
C.1.1 Fixpoint algorithm

The *fixpoint* algorithm shown in Algorithm C.1 (Benhamou et al., 1994; Benhamou, 1995) implements the process of chaotic iteration. This process continues to apply narrowing operators to a set of constraints until no further narrowing occurs over the variable domains.

The problem with Algorithm C.1 is that it works only with primitive constraints. That is, the constraint $c: x+y*z=t$ needs to be decomposed into primitive constraints such as $c_{dec} = \{y*z = \alpha, x+\alpha = t\}$. The introduction of the new variable α leads to poor domain tightening, especially when the same variable occurs multiple times in the same constraint.

Algorithm C.1 *Fixpoint* algorithm.

```
fixpoint(in: $\{C_1, ..., C_n\}$; inout: $\mathbf{B}$)
begin
  $Q \leftarrow \{C_1, ..., C_n\}$
  while $size(Q) \neq 0$ do
    $C \leftarrow removeFirst(Q)$          %%pop from stack
    $\mathbf{B}' \leftarrow narrow(C, \mathbf{B})$          %%narrow $\mathbf{B}$ with respect to $C$
    if $\mathbf{B}' \neq \mathbf{B}$ then
      $\mathbf{B} \leftarrow \mathbf{B}'$
      Add all constraints (except $C$) containing variables whose
          domains are narrowed in $\mathbf{B}$ to $Q$
  return $\mathbf{B}$
end
```



Constraint: $2x = z - y^2$

Figure C.1: Forward evaluation of a constraint (as illustrated by Benhamou et al. (1999)).

Algorithm C.2 Forward evaluation algorithm (adapted from Benhamou et al. (1999)).

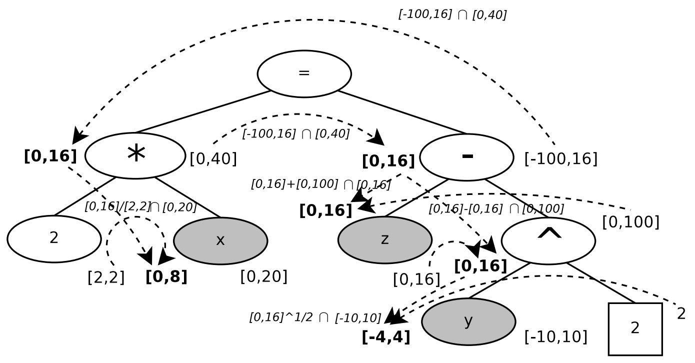
```
$forwardEvaluation$(in: node; inout: $\mathbf{B}$)
begin
  $t \leftarrow type$(node)
  case ($t$) of:
    $\Diamond$:  %An operation
      foreach child $c$ of node do
        $forwardEvaluation(c, \mathbf{B})$
      node.r$\leftarrow$Interval extension of $\Diamond$ operator, using
              interval r of child-nodes as operands
    $\alpha$:   %A constant
      node.r$\leftarrow [\alpha, \alpha]$
    $V_k$:  %A variable
      node.r$\leftarrow domain_k(\mathbf{B})$
end
```

C.1.2 Hull consistency

The algorithm for achieving hull consistency is named $HC4$ and is defined by Benhamou et al. (1999). The process is divided into two sections, namely forward evaluation and backward propagation of the evaluation tree of the constraint expressions.

Forward Evaluation The forward evaluation process is illustrated in Figure C.1, where the tree is traversed from the leaves to the root, evaluating the interval extension of each sub-term. The algorithm recursively traverses down the tree until the leaves are met, where the initial intervals of the variables and constants are loaded into a temporary variable r in each node. On the upward run the values for the inner nodes are calculated using the values from the children with respect to the interval extension of the operator at the current inner node. Algorithm C.2 implements forward evaluation.

Backward Propagation The process of backward propagation is illustrated in Figure C.2. The tree is traversed from the root node downwards. At each inner node the children are calculated according to the projection operator of the current node. The narrowing operation uses the current values of the children, as well as the value of the current node (which was modified as a result of



Constraint: $2x = z - y^2$

Figure C.2: Backward propagation of a constraint (as illustrated by Benhamou et al. (1999)).

Algorithm C.3 Backward propagation algorithm (adapted from Benhamou et al. (1999)).

```
$backwardPropogation$(in: node; inout: $\mathbf{B}$)
begin
  $t \leftarrow type$(node)
  case $t$ of:
    $\Diamond$:  %An operation
      $\mathbf{D}' \leftarrow$box constructed from r interval of each child
      foreach child of node do
        child.r$\leftarrow \pi_i(\rho_c \cap \mathbf{D}')$  %% Projection operator for operand of $\Diamond$
        $backwardPropogation$(child, $\mathbf{B}$)
    $V_j$:  %Variable
      $B_j \leftarrow B_j \cap$node.r  %% intersect domain of variable $j$
end
```

the propagation of its parent). Eventually, the propagation reaches the leaf nodes, where the final intervals of the variables are assigned. If the variable occurs more than once, then all intervals returned for that variable are intersected. Backward propagation is presented in Algorithm C.3. Further details regarding forward and backward propagation are described by Benhamou et al. (1999).

Algorithm *HC*4*Revise* The *forwardEvaluation* and *backwardPropogation* algorithms fit together into an algorithm called *HC*4*Revise*, which is presented in Algorithm C.4. This implements the constraint narrowing operator for the *HC*4 algorithm.

Algorithm C.4 HC4Revise (adapted from Benhamou et al. (1999)).

```
$HC4Revise$(in: constraint $c$; inout: box $\mathbf{B}$)
begin
  $forwardEvaluation(root(c), \mathbf{B})$
  $backwardPropagation(root(c), \mathbf{B})$
  return $\mathbf{B}$
end
```

Algorithm C.5 *HC*4 (as described by Benhamou et al. (1999)).

```
$HC4$(in: $\{c_1, ..., c_m\}$; inout: box $\mathbf{B}$)
begin
  $S \leftarrow \{c_1, ..., c_m\}$
  while $size(S) \neq \emptyset$ and $\mathbf{B} \neq \emptyset$ do
    $c \leftarrow removeFirst(S)$        %%pop constraint from stack
    $\mathbf{B}' \leftarrow HC4Revise(c, \mathbf{B})$
    if ($\mathbf{B}' \neq \mathbf{B}$) then        %%box is narrowed
       Add all constraints (except $c$) containing variables whose
          domains are narrowed in $\mathbf{B}'$ to $S$
       $\mathbf{B} \leftarrow \mathbf{B}'$
    else
      Remove $c$ from $S$
  return $\mathbf{B}$
end
```

Algorithm *HC*4 The *HC*4 algorithm presented in Algorithm C.5 is designed by Benhamou et al. (1999) for achieving hull consistency over a set of constraints. In a set of constraints, a variable x may occur in more than a single constraint at a time (for example, x may occur in constraint c_1 and c_2). If x's domain is narrowed over constraint c_1 then this may lead to further narrowing of other variables in c_2. Algorithm C.5 uses *chaotic iteration* to achieve a *fixpoint*, that is, a box for which hull consistency is achieved over all constraints. A function *HC*4*Revise* is executed that performs the forward and backward propagation steps. A list of constraints is maintained which indicates which constraints to use for further narrowing of the input box. Constraints are removed from the list when *idempotence* is achieved after using the *HC*4*Revise* operator (that is, no change results from narrowing the constraint). If however, the box is modified by *HC*4*Revise*, then all constraints that contain variables whose corresponding intervals in the box were modified are added to the end of the list.

C.1.3 Box consistency

Box consistency was created to overcome the problem of a constraint containing more than one instance of a variable. This is done by forming a set of univariate constraints for an input constraint, where each variable in a univariate constraint is replaced with its domain in the input box, except for one variable. For box consistency, the left-most and right-most roots define the global bounds of all possible roots for the function (Benhamou et al., 1994).

We use the *shrinkLeft* and *shrinkRight* algorithms to locate the left-most and right-most roots of a univariate function. The former is defined as Algorithm C.6. This algorithm evaluates the function over the initial interval, discarding it if it contains no roots. If roots are found then the interval is split, and each sub-division is pushed onto a stack for future root checking. The order in which the subdivisions are pushed onto the stack determine whether the left-most or right-most root is to be found. If a canonical interval is found that still contains zero, then this is assumed to be the left-most (right-most) root, and the algorithm returns successfully. This method is different from the Newton method for locating roots.

Algorithm C.6 *shrinkLeft* algorithm.

```
$shrinkLeft$(in: univariate function $F$; inout: interval $I$)
begin
  $S \leftarrow \{I\}$
  while $size(S) > 0$ do
    $I_n \leftarrow removeFirst(S)$        %%pop off stack
    $X \leftarrow F(I_n)$  %%Evaluate univariate function over $I_n$
    if $[0,0] \subset X$ then
       $V \leftarrow split(I_n)$
       if $size(V) > 0$ then
         Push intervals in $V$ onto stack $S$
       else
         return $I_n$ %%precision reached, root found
  return $FAIL$ %% No roots are found
end
```

Determining box consistency for a constraint is a matter of finding the left-most and right-most roots of each univariate function. If either one of the methods returns $FAIL$ then no root exists.

Algorithm *BC*3*Revise* The algorithm that creates the set of univariate constraints and executes *shrinkLeft* and *shrinkRight* is called *BC*3*Revise*. If this algorithms returns $FAIL$ then box consistency over the constraint is not possible (Benhamou et al., 1994).

Algorithm *BC*3 Box consistency over a set of constraints is achieved in a similar manner to hull consistency, that is, box-narrowing is repeated until no change occurs in the box of domains (Benhamou et al., 1994, 1999). Algorithm *BC*3 is identical to *HC*4, except that *BC*3*Revise* is used instead of *HC*4*Revise*.

Algorithm *BC*4 Algorithm C.7 is a more efficient method for box consistency over a set of constraints. Initially hull consistency is achieved for all the constraints. Once hull consistency is achieved over the set, box consistency is applied (Benhamou et al., 1999).

C.1.4 Inner contracting operator

Algorithm C.8 implements the inner contracting operator presented by Benhamou et al. (2004). *ICO*2 uses an outer contracting operator implemented as the *BC*3*Revise* algorithm. The initial box is narrowed using *BC*3*Revise*, and if a universally quantified variable is narrowed, then no solutions exist. If none are narrowed, then the set of inverted constraints are narrowed using *BC*3*Revise* over $\mathbf{B}'$. The box set difference between $\mathbf{B}'$ and $\mathbf{B}''$ forms the solution to the constraint set. If further solutions are required, then $\mathbf{B}''$ is split and processed recursively. If $\mathbf{B}''$ becomes canonical the algorithm stops (in this case, if the average width of the box is less than a specified threshold α).

Algorithm C.7 $BC4$ (adapted from Benhamou et al. (1994)).

```
$BC4$(in: set of constraints $C$; inout: box $\mathbf{B}$)
begin
  repeat
    $\mathbf{B}' \leftarrow \mathbf{B}$
    do
      $notFinished \leftarrow false$
      foreach $c \in C$ do
        $\mathbf{B}'' \leftarrow \mathbf{B}$
        $HC4Revise(c, \mathbf{B})$
        $notFinished \leftarrow$true if any variables in $\mathbf{B}''$ that occur once are narrowed,
                      false if $\mathbf{B}'' = FAIL$ or $notFinished$ already false
    while $notFinished$
    if ($\mathbf{B} \neq \emptyset$) then
      $BC3(C, \mathbf{B})$
  until $\mathbf{B}' = \mathbf{B}$ or $\mathbf{B} = \emptyset$
  return $\mathbf{B}$
end
```

Algorithm C.8 Inner contracting operator adapted from Benhamou et al. (2004).

```
$ICO2$(in: constraint $C$,
          box $\mathbf{B}$;
      out: $S$ containing solution boxes)
begin
  $\mathbf{B}' \leftarrow BC3Revise(C, \mathbf{B})$
  if universally quantified domains not shrunk then
    $\mathbf{B}'' \leftarrow BC3Revise(\overline{C}, \mathbf{B}')$
    $S \leftarrow$box set difference where no shrinking in
         universally quantified variables
    if $width(\mathbf{B}'') < \alpha$ then
      return $S$
    else
      $(\mathbf{B_1}, ..., \mathbf{B_k}) \leftarrow split_k(\mathbf{B}'')$
      apply $ICO2(C, B_j)$ for each split box, add solutions to $S$
      return $S$
  else
   return $FAIL$
end
```

Appendix D

Benchmark constraint systems

D.1 Example constraint file

Constraint systems are described in a constraint file that symbolically represents the constraints:

```
########## Object 1 in front of Object 0 ###############
CONSTRAINT{
  VARS{
    xo0cp0
    xo1cp0
    yo0cp0
    yo1cp0
    t1*
  }
  EXPR{
    (0.707*(xo1cp0-xo0cp0))+(-0.707*(yo1cp0-*yo0cp0)))^2 -
     0.8*(((xo1cp0-xo0cp0)^2 + (yo1cp0-yo0cp0))^2) * (0.707^2+(-0.707)^2)
  }
  OPERATOR{
    >
  }
  BOUNDARY{
    0
  }
}
########## Object 1 near Object 0 ###############
CONSTRAINT{
  ...
}
BOX{
  xo0cp0: -10, 10
  xo1cp0: -10, 10
  yo0cp0: -10, 10
  yo1cp0: -10, 10
  ...
  t1: 0.5, 0.6
}
```

D.2 Non-quantified benchmarks

The three benchmarks used in Chapter 5 for verifying the underlying algorithms are defined as follows:

D.2.1 CLPRevisited

This benchmark is used as a toy example by Benhamou et al. (1994), and is concerned with finding the roots of each of the following functions:

$$\begin{aligned} f_1(x) &= x^4 - 12x^3 + 47x^2 - 60x \\ f_2(x) &= x^4 - 12x^3 + 47x^2 - 60x + 24 \\ f_3(x) &= x^4 - 12x^3 + 47x^2 - 60x + 24.1 \text{ (inconsistent)} \end{aligned}$$

In all three cases x is assigned the initial domain $[-10, 20]$.

D.2.2 Broyden Banded function

This example is used as a benchmark by Benhamou et al. (1994) and is concerned with finding the roots of the following functions:

$$f_i(x_1, ..., x_n) = x_i(2 + 5x_i^2) + 1 - \sum_{j \in J_i} x_j(1 + x_j) \ \ (1 \leq i \leq m)$$

where $J_i = \{j | j \neq i \ \& \ max(1, i-5) \leq j \leq min(m, i+1)\}$.

All domains of x_i are initially the interval $[-1, 1]$. Benchmark systems are created for $n = \{5, 10, 20, 40, 80\}$.

D.2.3 More-Cosnard example

This example is used as a benchmark by Benhamou et al. (1994) and is concerned with finding the roots of the following functions $(1 \leq k \leq m)$:

$$f_k(x_i, ..., x_m) = x_k + \frac{1}{2}\left[(1 - t_k)\sum_{j=1}^{k} t_j(x_j + t_j + 1)^3 + t_k \sum_{j=k+1}^{m}(1 - t_j)(x_j + t_j + 1)^2\right]$$

where $t_j = jh$ and $h = 1/(m+1)$. Every x_i begins with the initial domain $[-4, 5]$. Benchmark systems are created for $m = \{10, 20, 40, 80\}$.

D.3 Quantified benchmarks

The following benchmarks are used for testing the ability to solve constraint systems containing universally quantified variables.

D.3.1 Parabola Fitting

Used by Jardillier and Languénou (1998) and Benhamou et al. (2004), this benchmark is concerned with finding parabolas above a line:

$$\forall t \in [0, 2] : at^2 + bt + c \geq 2t - 1$$

where the initial domains for a, b and c are $[0, 1]$.

D.3.2 Circle

A collision avoidance problem defined by Benhamou et al. (2004):

$$\forall t \in [-\pi, \pi] : \sqrt{(r_1 sint - x)^2 + (r_1 cost - y)^2} \geq d_1$$

where the initial domains of x and y are $[-5, 5]$, $d_1 = 0.5$ and $r_1 = 2.5$.

D.3.3 Satellite

A collision avoidance problem defined by Benhamou et al. (2004). If

$$\mathbf{f}_i(t) = \begin{pmatrix} x_i(t) \\ y_i(t) \\ z_i(t) \end{pmatrix} = \begin{pmatrix} d_i cos\theta_i sin\omega_i t + \phi_i \\ d_i \left(sin\psi_i sin\theta_i sin(\omega_i t + \phi_i) + cos\psi_i cos(\omega_i t + \phi_i) \right) \\ d_i \left(-cos\psi_i sin\theta_i sin(\omega_i t + \phi_i) + sin\psi_i cos(\omega_i t + \phi_i) \right) \end{pmatrix}$$

then the constraint system is defined as:

$$\forall t \in [-\pi, \pi] : \begin{cases} distance\left(\mathbf{f}_1(t), \mathbf{f}_j(t)\right) \geq s \\ distance\left(\mathbf{f}_2(t), \mathbf{f}_j(t)\right) \geq s \\ \vdots \\ distance\left(\mathbf{f}_n(t), \mathbf{f}_j(t)\right) \geq s \end{cases}$$

where s is the minimum distance between satellites (we use $s = 1$). Three satellites are used in the benchmark, so $n = 3$. Each satellite is parametrized as follows:

Parameter	Satellite 1	Satellite 2	Satellite 3
d_i	5.0	5.0	5.0
ω_i	1.0	1.0	1.0
ϕ_i	0.0	1.0	2.0
θ_i	0.0	1.0	1.5
ψ_i	0.0	1.0	1.5

The unknowns to be computed are for a fourth satellite $j = n+1$, where parameters θ_j, ϕ_j,ψ_j,ω_j all begin with domain $[0, 2\pi]$.

D.3.4 Robot

A collision problem defined by Benhamou et al. (2004):

$$\forall t \in [0,2] : \sqrt{(x - P_x(t))^2 + (y - P_y(t))^2} \geq d$$

where

$$\begin{aligned}
P_x(t) &= d_1 sin\alpha_1(t) + d_2 sin\left(\alpha_1(t) + \alpha_2(t) - \pi\right) + d_3 sin\left(\alpha_1(t) + \alpha_2(t) + \alpha_3(t)\right)\\
P_y(t) &= d_1 cos\alpha_1(t) + d_2 cos\left(\alpha_1(t) + \alpha_2(t) - \pi\right) + d_3 cos\left(\alpha_1(t) + \alpha_2(t) + \alpha_3(t)\right)\\
\alpha_1(t) &= t + \pi/4\\
\alpha_2(t) &= 2t - 1\\
\alpha_3(t) &= 0.2t + 0.1
\end{aligned}$$

The initial domains for x and y are $[0,5]$, $d = 0.5$, $d_1 = 1.0$, $d_2 = 2.0$ and $d_3 = 1.0$.

D.3.5 PointPath

A motion planning problem used by Jaulin and Walter (1996) and Benhamou et al. (2004):

$$\forall t \in [0,1] : \begin{cases} (x(t) - 4.8)^2 + (y(t) - 1)^2 \geq 1 \\ y(t) \geq sin\left(x(t)\right) \end{cases}$$

where $\mathbf{M}(t) = \begin{pmatrix} x(t) \\ y(t) \end{pmatrix}$ and

$$\mathbf{M}(t) = M_0 B_0^3(t) + P_1 B_1^3(t) + P_2 B_2^3(t) + M_1 B_3^3(t)$$

where Bernstein polynomials are:

$$B_0^3(t) = (1-t)^3, B_1^3(t) = 3t(1-t)^2, B_2^3(t) = 3t^2(1-t), B_3^3(t) = t^3$$

Initial domains for P_1 and P_2 are $[-10,10]$, $M_0 = (-1,-0.6)^T$ and $M_1 = (6,0)^T$.

D.3.6 Robust 1

Defined by Ratschan (2006, 2008):

$$\forall p \in [0,1] : \begin{cases} 9 + 48p + 48q + 32pq > 0 \\ 1 + p + q > 0 \\ -16p - 16q + 16p^2 + 16q^2 + 7 > 0 \end{cases}$$

where q has the initial domain $[-2,2]$.

D.4 Fiction-to-animation benchmarks

The following benchmarks are defined for evaluation in terms of fiction-to-animation constraints:

- NEAR: Four objects, each constrained to appear $inFrontOf$ and $near$ one of the others. $noCollide$ constraints over all objects.
- SCENE: Six objects arranged with using $toRightOf$, $toLeftOf$, $inFrontOf$, $behind$, $noCollide$ and $near$ constraints.
- LAYOUT3: Three objects arranged with the $noCollide$ constraint.
- WAYPOINTS: One object constrained to pass through 3 fixed way-points, using the $near$ constraint over 3 different time-intervals.
- DYNAMIC1STATIC1: One object static, the other dynamic having trajectories of increasing degree in each dimension. $near$ and $inFrontOf$ applied over sub-interval of time, $noCollide$ applied over entire interval of time.
- DYNAMIC2: Both objects dynamic, having trajectory of increasing degree in each dimension. $near$ and $inFrontOf$ applied over sub-interval of time, $noCollide$ applied over entire interval of time.
- COLLISION: n objects, each constrained to be $near$ and $noCollide$ with every other object. Increases in complexity with addition of each object, and for $n > 3$ no solution exists.

The exact constraints that comprise each benchmark are defined in the following sections. Actual formulations for each type of constraint (such as $inFrontOf$ and $near$) are presented in Chapter 6.

D.4.1 Front

Objects	Dimensions	Degree	Quantified variables	Initial Domains
4	4	0	0	$[-10, 10]$

- Trajectory A: $\mathbf{r}^A(t) = \mathbf{p}_0^A = \begin{cases} x0cp0 & (x - dimension) \\ z0cp0 & (z - dimension) \end{cases}$
- Trajectory B: $\mathbf{r}^B(t) = \mathbf{p}_0^B = \begin{cases} x1cp0 & (x - dimension) \\ z1cp0 & (z - dimension) \end{cases}$
- Trajectory C: $\mathbf{r}^C(t) = \mathbf{p}_0^C = \begin{cases} x2cp0 & (x - dimension) \\ z2cp0 & (z - dimension) \end{cases}$
- Trajectory D: $\mathbf{r}^D(t) = \mathbf{p}_0^D = \begin{cases} x3cp0 & (x - dimension) \\ z3cp0 & (z - dimension) \end{cases}$

Constraints:

`B InFrontOf A`	$\mathbf{u}_A = (0.707, 0.707)$
`B Near A`	
`C InFrontOf B`	$\mathbf{u}_B = (0.707, -0.707)$
`C Near B`	
`D InFrontOf C`	$\mathbf{u}_C = (0.707, 0.707)$
`D Near C`	
`B NoCollide A`	
`C NoCollide A`	
`D NoCollide A`	
`C NoCollide B`	
`D NoCollide B`	
`D NoCollide C`	

D.4.2 Scene

Objects	Dimensions	Degree	Quantified variables	Initial Domains
6	2	0	0	$[-20, 20]$

- Trajectory A: $\mathbf{r}^A(t) = \mathbf{p}_0^A = \begin{cases} x0cp0 & (x - dimension) \\ z0cp0 & (z - dimension) \end{cases}$
- Trajectory B: $\mathbf{r}^B(t) = \mathbf{p}_0^B = \begin{cases} x1cp0 & (x - dimension) \\ z1cp0 & (z - dimension) \end{cases}$
- Trajectory C: $\mathbf{r}^C(t) = \mathbf{p}_0^C = \begin{cases} x2cp0 & (x - dimension) \\ z2cp0 & (z - dimension) \end{cases}$
- Trajectory D: $\mathbf{r}^D(t) = \mathbf{p}_0^D = \begin{cases} x3cp0 & (x - dimension) \\ z3cp0 & (z - dimension) \end{cases}$
- Trajectory E: $\mathbf{r}^E(t) = \mathbf{p}_0^E = \begin{cases} x4cp0 & (x - dimension) \\ z4cp0 & (z - dimension) \end{cases}$
- Trajectory F: $\mathbf{r}^F(t) = \mathbf{p}_0^F = \begin{cases} x5cp0 & (x - dimension) \\ z5cp0 & (z - dimension) \end{cases}$

Constraints:

`A InFrontOf B`	$\mathbf{u}_B = (1,0)$	`C NoCollide B`
`A Near B`		`D NoCollide B`
`C Behind B`	$\mathbf{u}_B = (-1,0)$	`E NoCollide B`
`C Near B`		`F NoCollide B`
`E ToLeftOf B`	$\mathbf{u}_B = (0,1)$	`D NoCollide C`
`E Near B`		`E NoCollide C`
`F ToRightOf D`	$\mathbf{u}_D = (0,-1)$	`F NoCollide C`
`F Near D`		`E NoCollide D`
`B NoCollide A`		`F NoCollide D`
`C NoCollide A`		`F NoCollide E`
`D NoCollide A`		
`E NoCollide A`		
`F NoCollide A`		

D.4.3 Layout3

Objects	Dimensions	Degree	Quantified variables	Initial Domains
3	2	0	0	$[-20, 20]$

- Trajectory A: $\mathbf{r}^A(t) = \mathbf{p}_0^A = \begin{cases} x0cp0 & (x - dimension) \\ z0cp0 & (z - dimension) \end{cases}$
- Trajectory B: $\mathbf{r}^B(t) = \mathbf{p}_0^B = \begin{cases} x1cp0 & (x - dimension) \\ z1cp0 & (z - dimension) \end{cases}$
- Trajectory C: $\mathbf{r}^C(t) = \mathbf{p}_0^C = \begin{cases} x2cp0 & (x - dimension) \\ z2cp0 & (z - dimension) \end{cases}$

Constraints:

```
A NoCollide B
A NoCollide C
B NoCollide C
```

D.4.4 WayPoints

Objects	Dimensions	Degree	Quantified variables	Initial Domains
1	2	2	3	$[-100, 100]$

Trajectory A (degree 2), for the time interval $t = [0, 1]$:

$$\mathbf{r}^A(t) = (1-t)^2\mathbf{p}_0^A + t(1-t)\mathbf{p}_1^A + t^2\mathbf{p}_2^A = \begin{cases} \big((1-t)^2 * x0cp0\big) + (t * (1-t) * x0cp1) + \big(t^2 * x0cp2\big) \\ \big((1-t)^2 * z0cp0\big) + (t * (1-t) * z0cp1) + \big(t^2 * z0cp2\big) \end{cases}$$

Constraints:

A Near B $\forall t_1 \in [0, 0.01]$	B at $(-7.29, -18.98)$
A Near C $\forall t_2 \in [0.5, 0.51]$	C at $(-17.28, 0.0)$
A Near D $\forall t_3 \in [0.99, 1.0]$	D at $(10.0, -10.0)$

D.4.5 Dynamic1Static1

Objects	Dimensions	Degree	Quantified variables	Initial Domains
2	1 to 2	0 to 2	2	$[-10, 10]$

The following provides the example in 2 dimensions, degree 1 curves, for the time interval $t = [0, 1]$:

- Trajectory A: $\mathbf{r}^A(t) = (1-t)\mathbf{p}_0^A + t\mathbf{p}_1^A = \begin{cases} ((1-t) * x0cp0) + (t * x0cp1) & (x - dimension) \\ ((1-t) * z0cp0) + (t * z0cp1) & (z - dimension) \end{cases}$
- Trajectory B: $\mathbf{r}^B(t) = (1-t)\mathbf{p}_0^B + t\mathbf{p}_1^B = \begin{cases} x1cp0 & (x - dimension) \\ z1cp0 & (z - dimension) \end{cases}$

Constraints:

`A InFrontOf B` $\forall t_1 \in [0.5, 0.6]$ $\quad \mathbf{u}_B = \dfrac{d}{dt}\mathbf{r}^B(t_1)$

`A Near B` $\forall t_1 \in [0.5, 0.6]$

`A NoCollide B` $\forall t_2 \in [0, 1]$

D.4.6 Dynamic2

Objects	Dimensions	Degree	Quantified variables	Initial Domains
2	1 to 2	0 to 2	2	$[-10, 10]$

The following provides the example in 2 dimensions, degree 1 curves, for the time interval $t = [0, 1]$:

- Trajectory A: $\mathbf{r}^A(t) = (1-t)\mathbf{p}_0^A + t\mathbf{p}_1^A = \begin{cases} ((1-t) * x0cp0) + (t * x0cp1) & (x - dimension) \\ ((1-t) * z0cp0) + (t * z0cp1) & (z - dimension) \end{cases}$
- Trajectory B: $\mathbf{r}^B(t) = (1-t)\mathbf{p}_0^B + t\mathbf{p}_1^B = \begin{cases} ((1-t) * x1cp0) + (t * x1cp1) & (x - dimension) \\ ((1-t) * z1cp0) + (t * z1cp1) & (z - dimension) \end{cases}$

Constraints:

`A InFrontOf B` $\forall t_1 \in [0.5, 0.6]$ $\quad \mathbf{u}_B = \dfrac{d}{dt}\mathbf{r}^B(t_1)$

`A Near B` $\forall t_1 \in [0.5, 0.6]$

`A NoCollide B` $\forall t_2 \in [0, 1]$

D.4.7 Collision

Objects	Dimensions	Degree	Quantified variables	Initial Domains
2 to 8	2	0 to 1	0	$[-10, 10]$

The following provides an example of 4 static objects, although the experiment ranges from 2 to 8 objects, with static or dynamic trajectories.

- Trajectory A: $\mathbf{r}^A(t) = \mathbf{p}_0^A = \begin{cases} x0cp0 & (x - dimension) \\ z0cp0 & (z - dimension) \end{cases}$
- Trajectory B: $\mathbf{r}^B(t) = \mathbf{p}_0^B = \begin{cases} x1cp0 & (x - dimension) \\ z1cp0 & (z - dimension) \end{cases}$
- Trajectory C: $\mathbf{r}^C(t) = \mathbf{p}_0^C = \begin{cases} x2cp0 & (x - dimension) \\ z2cp0 & (z - dimension) \end{cases}$
- Trajectory D: $\mathbf{r}^D(t) = \mathbf{p}_0^D = \begin{cases} x3cp0 & (x - dimension) \\ z3cp0 & (z - dimension) \end{cases}$

Constraints:

```
B Near A            B NoCollide A
C Near A            C NoCollide A
D Near A            D NoCollide A
C Near B            C NoCollide B
D Near B            D NoCollide B
D Near C            D NoCollide C
```

Appendix E

Details of manual modifications

The modifications made to the automatically produced scene descriptions and virtual environments are listed here.

E.1 Cow scene

- Coreference corrections (indicated in bold):

valley. Anne/ANNE did n't very much like a big brown cow/COW who came up close and stared at her/ANNE, but it/**COW** went away when Daddy/DADDY told it/**COW** to. The children/CHILDREN ate enormously, and Mother/MOTHER said that instead of having a tea-picnic at half-past four they would have to go to a tea-house somewhere, because they had eaten all the tea sandwiches/SANDWICHES as well as the lunch ones! "What time shall we be at Aunt Fanny's?" asked Julian/JULIAN, finishing up the very last sandwich/SANDWICH and wishing there were more. "About six o'clock with luck," said Daddy/DADDY. "Now who wants to stretch their legs a bit? We've another long spell in the car, you know." The car/CAR seemed to eat up the miles as it purred along.

- Abstract constraint creation process:

Scene detail:	Modifications:
Scene segmentation:	- None
Model descriptors:	- Cow trajectory increased to degree 1 (from 0)
Coreference:	- As shown above.
Abstract constraints:	- None

- Modifications in 3D modeling environment:

Modification Type:	Modifications:
Insertion	- None
Modification	- Camera positioning
Deletion	- Removal of “sandwiches” placeholder cube; - Removal of “sandwich” placeholder cube; - Removal of “children” model

E.2 Rabbit scene

- Coreference corrections (indicated in bold):

" Look ! There 's a rabbit ! " cried Dick/DICK , as a big sandy rabbit/RABBIT lollopped slowly across the yard . It/**RABBIT** disappeared into a hole on the other side . Then another rabbit/RABBIT appeared , sat up and looked at the children/CHILDREN , and then vanished too . The children/CHILDREN were thrilled . They had never seen such tame rabbits/RABBITS before . A third rabbit/RABBIT appeared . It/**RABBIT** was a small one with absurdly big ears , and the tiniest white bob of a tail . It/**RABBIT** did n't even look at the children/CHILDREN . It/**RABBIT** bounded about in a playful way , and then , to the children/CHILDREN 's enormous delight , it/**RABBIT** sat up on its/**RABBIT** hind legs , and began to wash its/**RABBIT** big ears , pulling down first one and then another . But this was too much for Timothy/TIM . He/TIM had watched the other two bound across the yard and then disappear without so much as barking at them . But to see this youngster actually sitting there washing its/**RABBIT** ears under his/TIM very nose was really too much for any dog/TIM . He/TIM gave an excited yelp and rushed full-tilt at the surprised rabbit/RABBIT . For a moment the little thing did n't move . It/**RABBIT** had never been frightened or chased before , and it/**RABBIT** stared with big eyes at the rushing dog/TIM . Then it/**RABBIT** turned itself about and tore off at top speed , its/**RABBIT** white bobtail going up and down as it/**RABBIT** bounded away . It/**RABBIT** disappeared under a gorse bush/BUSH near the children/CHILDREN . Timothy/TIM went after it/**RABBIT** , vanishing under the big bush too . Then a shower of sand and earth was thrown up as Tim/TIM tried to go down the hole after the rabbit and scraped and scrabbled with his/TIM strong front paws as fast as he/TIM could . He/TIM yelped and whined in excitement , not seeming to hear George/GEORGE 's voice calling to him/TIM . He/TIM meant to get that rabbit ! He/TIM went almost mad as he/TIM scraped at the hole , making it bigger and bigger . " Tim/TIM ! Do you hear me ! Come out of there ! " shouted George/GEORGE . " You 're not to chase the rabbits here . You know you must n't . You 're very naughty . Come out ! " But Tim/TIM did n't come out . He/TIM just went on and on scraping away madly . George/GEORGE went to fetch him/TIM . Just as she/GEORGE got up to the gorse bush/BUSH the scraping suddenly stopped .

- Abstract constraint creation process:

Scene detail:	Modifications:
Scene segmentation:	- None
Model descriptors:	- Rabbit trajectory increased to 1 (from 0); - Timothy assigned "dog" model (from humanoid model)
Coreference:	- As shown above.
Abstract constraint corrections:	- 1 corrected constraint - TIM NEAR RABBIT, end time reduced until start of RABBIT UNDER BUSH constraint.

- Modifications in 3D modeling environment:

Modification Type:	Modifications:
Insertion	- None
Modification	- Camera positioning
Deletion	- None

E.3 Study scene

- Coreference corrections (indicated in bold):

He/**JULIAN** stole in . His/**JULIAN** uncle/QUENTIN still snored . He/**JULIAN** tiptoed by him/QUENTIN to the table/TABLE behind his/**JULIAN** uncle/QUENTIN 's chair/CHAIR . He/JULIAN took hold of the box/BOX . And then a bit of the broken wood**[Remove]** of the box/BOX fell to the floor with a thud ! His/JULIAN uncle/QUENTIN stirred in his/QUENTIN chair/CHAIR and opened his/QUENTIN eyes . Quick as lightning the boy/**JULIAN** crouched down behind his/**JULIAN** uncle/QUENTIN 's chair/CHAIR , hardly breathing . " What 's that ? " he/**JULIAN** heard his/JULIAN uncle/QUENTIN say . Julian/JULIAN did n't move . Then his/JULIAN uncle/QUENTIN settled down again and shut his/QUENTIN eyes . Soon there was the sound of his/QUENTIN rhythmic snoring ! " Hurrah ! " thought Julian/JULIAN . " He/JULIAN 's off again ! " Quietly he/JULIAN stood up , holding the box/BOX . On tiptoe he/JULIAN crept to the French window . He/JULIAN slipped out and ran softly down the garden path . He/JULIAN did n't think of hiding the box/BOX . All he/JULIAN wanted to do was to get to the other children/CHILDREN and show them what he/JULIAN had done !

- Abstract constraint creation process:

Scene detail:	Modifications:
Scene segmentation:	- Manual delimitation of this scene, since no explicit indicator exists
Model descriptors:	- None
Coreference:	- As shown above.
Abstract constraint corrections:	- None

- Modifications in 3D modeling environment:

Modification Type:	Modifications:
Insertion	- None
Modification	- Camera positioning
Deletion	- Deletion of “Children” model

E.4 Travel sequence

- Coreference corrections (indicated in bold):

Dick/DICK and Julian/JULIAN , who shared a room , woke up at about the same moment , and stared out of the nearby window . " It 's a lovely day , hurrah ! " cried Julian/JULIAN , leaping out of bed/BED . " I do n't know why , but it always seems very important that it should be sunny on the first day of a holiday . Let 's wake Anne/ANNE . " Anne/ANNE slept in the next room . Julian/JULIAN ran in and shook her/ANNE . " Wake up ! It 's Tuesday ! And the sun 's shining . " Anne/ANNE woke up with a jump and stared at Julian/JULIAN joyfully . " It 's come at last ! " she/ANNE said . " I thought it never would . Oh , is n't it an exciting feeling to go away for a holiday ! " They started soon after breakfast . Their car/CAR was a big one , so it held them all very comfortably . Mother/MOTHER sat in front with Daddy/DADDY , and the three children/CHILDREN sat behind , their feet on two suitcases/SUITCASES . In the luggage-place at the back of the car/CAR were all kinds of odds and ends , and one small trunk/TRUNK . Mother/MOTHER really thought they had remembered everything . Along the crowded London roads they went , slowly at first , and then , as they left the town behind , more quickly . Soon they were right into the open country , and the car/CAR sped along fast . The children/CHILDREN sang songs to themselves , as they always did when they were happy . " Are we picnicking soon ? " asked Anne/ANNE , feeling hungry all of a sudden . " Yes , " said Mother/MOTHER .

- Abstract constraint creation process:

Scene detail:	Modifications:
Scene segmentation:	- Manual delimitation for setting ANNES_ROOM and OUTSIDE (scene directly after)
Model descriptors:	- None
Coreference:	- As shown above.
Abstract constraint corrections:	- Insertion of JULIAN INSIDE BED; JULIAN NO_COLLIDE BED to achieve motion of Julian getting out of bed. - Insertion of BED INSIDE ANNES_ROOM; ANNE_INSIDE BED to achieve the idea of Anne being in bed (not explicitly state) - Insertion of JULIAN NO_COLLIDE BED; JULIAN NEAR BED to cater for implied constraints required for the addition of the new BED object to the scene. - Change MOTHER NO_COLLIDE_CAR; DADDY NO_COLLIDE CAR; CHILDREN NO_COLLIDE CAR to MOTHER INSIDE CAR; DADDY INSIDE CAR; CHILDREN INSIDE CAR to ensure avatars inside car, since it is not explicitly stated in a simple manner in the text.

- Modifications in 3D modeling environment:

Modification Type:	Modifications:
Insertion	- None
Modification	- Camera positioning in 6 scenes (modification) - Removal of "BOYS" object (deletion) - Material adjustments in 2 scenes (modifiction)
Deletion	- Deletion of "Children" model

E.5 Follow scene

- Coreference corrections (indicated in bold):

" THIS'LL BE IT ! " KICKAHA/KICKAHA SAID . " I KNOW IT , KNOW IT ! I CAN feel the forces shaping themselves into a big funnel pouring us onto the goal ! It 's just ahead ! We 've finally made it ! " He/KICKAHA wiped the sweat from his/KICKAHA forehead . Though breathing heavily , he/KICKAHA increased his/KICKAHA pace . Anana/ANANA was a few steps behind and below him/KICKAHA on the steep mountain trail . She/ANANA spoke to herself/ANANA in a low voice . He/KICKAHA never paid any attention to her/ANANA discouraging-that is , realistic-words , anyway . " I 'll believe it when I see it . "

- Abstract constraint creation process:

Scene detail:	Modifications:
Scene segmentation:	- None
Model descriptors:	- None
Coreference:	- As shown above.
Abstract constraint corrections:	- None

- Modifications in 3D modeling environment:

Modification Type:	Modifications:
Insertion	- None
Modification	- Camera positioning
Deletion	- None

E.6 House sequence

- Coreference corrections (indicated in bold):

They were sent to the house/HOUSE of an old Professor/PROFESSOR who lived in the heart of the country , ten miles from the nearest railway station and two miles from the nearest post office . He/PROFESSOR had no wife/MRS BEAVER and he/PROFESSOR lived in a very large house/HOUSE with a housekeeper called Mrs/MRS MACREADY Macready/MRS MACREADY and three servants . (Their names were Ivy , Margaret and Betty , but they do not come into the story much .) He/PROFESSOR himself/PROFESSOR was a very old man with shaggy white hair which grew over most of his/PROFESSOR face as well as on his/PROFESSOR head , and they liked him/PROFESSOR almost at once ; but on the first evening when he/PROFESSOR came out to meet them at the front door he/PROFESSOR was so odd-looking that Lucy/LUCY (who was the youngest) was a little afraid of him/PROFESSOR , and Edmund/EDMUND (who was the next youngest) wanted to laugh and had to keep on pretending he/EDMUND was blowing his/EDMUND nose to hide it . As soon as they had said good night to the Professor/PROFESSOR and gone upstairs on the first night , the boys/**PETER** came into the girls/GIRLS ' room and they all talked it over . " We 've fallen on our feet and no mistake , " said Peter/PETER . " This is going to be perfectly splendid . That old chap will let us do anything we like . " " I think he/PETER 's an old dear , " said Susan/SUSAN . " Oh , come off it ! " said Edmund/EDMUND , who was tired and pretending not to be tired , which always made him/EDMUND bad-tempered .

- Abstract constraint creation process:

Scene detail:	Modifications:
Scene segmentation:	- None
Model descriptors:	- None
Coreference:	- As shown above.
Abstract constraint corrections:	- None

- Modifications in 3D modeling environment:

Modification Type:	Modifications:
Insertion	- None
Modification	- Camera positioning
Deletion	- Removal of "Girls" model and "Professor" model

Appendix F

Multi-modal animated films

The accompanying DVD contains rendered films.

Behaviour Quality

The following videos are provided as described in Chapter 6, Section 6.3.2.2:

- Dynamic1Static1, degree 1
- Dynamic1Static1, degree 2
- Dynamic2, degree 2 (Quality = 25.08)
- Dynamic2, degree 2 (Quality = 161.32)

Animated films

The following videos are provided as described in Chapter 6, Section 6.5.4:

1. Cow scene
2. Rabbit scene
3. Study scene
4. Travel sequence
5. Follow scene
6. House sequence

Videos and snapshots are also available at the project web-site:
`http://www.cs.ru.ac.za/research/g05g1909/`