Computer Science 3 - 2017

Programming Language Translation

Practical for Week 4, beginning 7 August 2017

Hand in your solutions to the second part of this practical *before* lunch time on your next practical day, correctly packaged in a transparent folder with your cover sheets. Please do NOT come to a practical and spend the first hour printing or completing solutions from the previous week's exercises. Since the practical will have been done on a group basis, please hand in one copy of the cover sheet for each member of the group. These will be returned to you in due course, signed by the marker. **Please make it clear whose folder you have used for the electronic submission, for example g15A1234.** Lastly, please resist the temptation to carve up the practical, with each group member only doing one task. The group experience is best when you discuss each task together.

Objectives:

In this practical you are to

- familiarize you with the rules and restrictions of LL(1) parsing, and
- help you understand the concept of ambiguity, and
- get more experience in writing simple grammars.

You will need this prac sheet and your text book. As usual, copies of the prac sheet are also available at `http://www.cs.ru.ac.za/courses/CSc301/Translators/trans.htm`.

Outcomes:

When you have completed this practical you should understand

- how to apply the LL(1) rules manually and automatically;
- how to use Coco/R more effectively;

To hand in:

This week you are required to hand in, besides the cover sheet:

- *The solutions to tasks 1, 2, 3 and 4 by 08h00 tomorrow morning on the attached sheet or an edited copy*.
- Listings of your solutions to the grammar problems, produced on the laser printer by using the LPRINT utility. (Don't put "tabs" into your files - use hard "spaces", but take care: lines can get quite wide!)
- Electronic copies of your grammar files (ATG files), stored in a folder under one of the group's student numbers.

I do NOT require listings of any C# code produced by Coco/R.

Keep the prac sheet and your solutions until the end of the semester. Check carefully that your mark has been entered into the Departmental Records.

You are referred to the rules for practical submission which are clearly stated in our Departmental Handbook. However, for this course pracs must be posted in the "hand-in" box outside the laboratory and not given to demonstrators.

A rule not stated there, but which should be obvious, is that you are not allowed to hand in another group's or student's work as your own. Attempts to do this will result in (at best) a mark of zero and (at worst) severe disciplinary action and the loss of your DP. You are allowed - even encouraged - to work and study with other students, but if you do this you are asked to acknowledge that you have done so. You are expected to be familiar with the University Policy on Plagiarism, which you can consult on the University web site.

The first few tasks do not need you to use a computer, nor should you. Do them by hand.

Task 1 - Self-describing EBNF

The following Cocol grammar describes (in EBNF) a set of productions written in EBNF. For example, it could be supplied to Coco/R to produce a system that could parse a set of productions similar to the ones discussed in class (and given below the Cocol description).

```
COMPILER EBNFO $CN
/* Parse a set of EBNF productions - grammar only
   P.D. Terry, Rhodes University, 2017 */

CHARACTERS
  letter   = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz" .
  lowline  = "_" .
  digit    = "0123456789" .
  noquote  = ANY - '"' .

TOKENS
  nonterminal = letter { letter | lowline | digit } .
  terminal    = '"' noquote { noquote } '"' .

COMMENTS FROM "(*" TO "*)"  NESTED

IGNORE  CHR(9) .. CHR(13)

PRODUCTIONS
   EBNFO      = { Production } EOF .
   Production = nonterminal "=" Expression "." .
   Expression = Term { "|" Term } .
   Term       = Factor { Factor } .
   Factor     =   nonterminal
                | terminal
                | "[" Expression "]"
                | "(" Expression ")"
                | "{" Expression "}" .
END EBNFO.
```

(a) Do the production in the EBNF grammar obey the LL(1) constraints? Justify your answer.

(b) The `nonterminal` definition allows a non-terminal to end with a lowline - as in `my_name__`. How should the definition be changed to allow lowlines *within* a non-terminal, but not at the end?

c) Now consider a set of productions specifying simple English sentences:

```
Sentence      = "the" QualifiedNoun Verb | Pronoun Verb .
QualifiedNoun = [ Adjective { Adjective } ] Noun .
Noun          = "man" | "girl" | "boy" | "lecturer" .
Pronoun       = "he" | "she" .
Verb          = "talks" | "listens" | "mystifies" .
Adjective     = "tall" | "thin" | "sleepy" .
```

The productions leading on from *Sentenc*e allow a *QualifiedNoun* to consist of a *Noun* preceded by zero or more *Adjectives*. But in place of the line reading

```
QualifiedNoun = [ Adjective { Adjective } ] Noun .
```

one might have been tempted to write

```
QualifiedNoun = [ { Adjective } Adjective ] Noun .
```

or even

```
QualifiedNoun = { Adjective } Noun .
```

Do any of these suggestions introduce either ambiguity or non-LL(1) compliance into the grammar for *Sentence*? If they do, does this mean that the EBNF parser that could be constructed by Coco/R from the Cocol code given earlier would be unable to recognise the productions defining a *Sentence*? Does it mean that if Coco/R was used to try to construct a parser for sentences using one or other of these suggestions for *QualifiedNoun* it would be unable to do so? Discuss.

Task 2 - Ambiguity

The following Cocol grammar attempts to describe a sequence of Roman numbers between 1 and 10 - as you probably know, these were represented by I, II, III, IV, V, VI, VII, VIII, IX and X.

```
COMPILER Roman
/* Parse a list of Roman numbers in the range 1 ... 10 - grammar only
   P.D. Terry, Rhodes University, 2017 */

IGNORE  CHR(9) .. CHR(13)

PRODUCTIONS
   Roman      = OneNumber { "," OneNumber } "." .
   OneNumber  = StartI | StartV | StartX .
   StartI     = "I" [ [ "I" ] [ "I" ] | "V" | "X" ] .
   StartV     = "V" [ "I" ] [ "I" ] [ "I" ] .
   StartX     = "X" .
END Roman.
```

Is this an LL(1) grammar? If not, why not - and if it is, make a convincing argument for it. Is it an ambiguous grammar? If so, demonstrate an ambiguity, and if not, convince a non-believer.

Task 3 - Palindromes

Palindromes are character strings that read the same from either end, like "Hannah" or my brother Peter's favourite line when he did the CTM advertisements: "Bob Bob". The following represent various attempts to find grammars that describe palindromes made only of the letters *a* and *b*:

(1) *Palindrome* = "*a*" *Palindrome* "*a*" | "*b*" *Palindrome* "*b*" .
(2) *Palindrome* = "*a*" *Palindrome* "*a*" | "*b*" *Palindrome* "*b*" | "*a*" | "*b*" .
(3) *Palindrome* = "*a*" [*Palindrome*] "*a*" | "*b*" [*Palindrome*] "*b*" .
(4) *Palindrome* = ["*a*" *Palindrome* "*a*" | "*b*" *Palindrome* "*b*" | "*a*" | "*b*"] .

Which grammars achieve their aim? If they do not, explain why not. Which of them are LL(1)? Can you find other (perhaps better) grammars that describe palindromes and which *are* LL(1)?

Task 4 - Pause for thought

Which of the following statements are true? Justify your answer.

(a) An LL(1) grammar cannot be ambiguous.
(b) A non-LL(1) grammar must be ambiguous.
(c) An ambiguous language cannot be described by an LL(1) grammar.
(d) It is possible to find an LL(1) grammar to describe any non-ambiguous language.

Hand in your solutions to tasks 1 through 4 before continuing.

Task 5 - Grab a mug of hot Coco and press on

There are several files that you need for the rest of the practical, zipped up this week in the file `PRAC4.ZIP`

Copy the prac kit into a newly created directory/folder in your file space in the usual way:

```
j:
md  prac4
cd  prac4
copy  i:\csc301\trans\prac4.zip
unzip  prac4.zip
```

You will find the executable version of Coco/R and batch files for running it, frame files, and various sample programs and grammars, including ones for the grammars given in tasks 1, 2 and 3.

After unpacking this kit attempt to make the parsers, as you did last week. Ask the demonstrators to show you how to get Coco/R to show you the *FIRST* and *FOLLOW* sets for the non-terminals of the grammar, and verify that the objections (if any) that Coco/R raises to these grammars are the same as you have determined by hand.

Task 6 - Stop watching videos and playing computer games - listen to steam radio

As an example of checking the LL(1) conditions for a grammar expressed in EBNF, let us consider how we might describe the daily activities of SAFM - "your news and information leader" (more realistically described as a non-stop party political broadcast). Hour by hour Sakina, Tsepiso, Nancy, Ashraf, Elvis, Bongi and their comrades present what are generously called shows. These include music and advertisements (as people pay no licence fees, the station has to raise revenue through advertising). What passes as "information" is provided by news bulletins, weather reports and talk shows. News bulletins consist of a homogeneous stream of stories, while talk shows consist of interchanges between the host and listeners who are keen enough to phone in - with the person hosting the exchange getting the first and last word on the subject, of course.

Items like "advert", "rain" and "song" are really in the category of the lexical terminals which a scanner (in the person of someone listening to the radio) would recognize as key symbols while parsing a broadcast. So one playful attempt to describe a day's activities might be on the lines of

```
Radio       =  { TalkShow | NewsBulletin | "song" | "advert" } EOF .
NewsBulletin =  "advert" NewsItem { NewsItem } [ Weather ] Filler .
NewsItem    =    "ANC" [ "SACP" ] | "protest" | [ "SACP" ] "ANC" | "zuma"
                | "Helen" "tweet" | Scandal | Accident .
TalkShow    =  "host" { listener "host" } [ Filler ] .
Accident    =  "collision" "claims" "another" number "lives" .
Scandal     =  { "GuptaLeak" } "corruption"  | "Hlaudi" .
Filler      =  "song" | "advert"  .
Weather     =  { "snow" | "rain" | "cloudy" | "windy" } .
```

Analyze this grammar in detail. Treat `listener` and `number` as terminals. One way is to begin by recasting it in a form that introduces further non-terminals that allow for the elimination of the [ ] and { } meta-brackets, but you can probably do the analysis along the lines of the methods suggested on pages 95 - 96. If it proves out to be non-LL(1), try to find an essentially equivalent description that *is* LL(1), or argue why this should be impossible.

The weather bulletins could be improved if one was informed of the town to which the prediction applied. Suggest how the grammar would need modification to do this. As with listener and number, regard a town name as a terminal.

Task 7 - How are things stacking up?

Develop a Cocol grammar `PVMASM,ATG` that describes programs written in the PVM code that you struggled with in Practical 2. That should be pretty easy but be careful to describe the language as tightly as possible. Then, just to make the language more attractive, suppose you had been able to use optional alphanumeric labels in your code as the destinations of branching instructions - as exemplified below. Modify the grammar to handle this.

```
        DSP   2
LOOP    LDA   1
        INPI
        LDL   1
        BZE   EXIT
        BRN   LOOP
EXIT    LDL   1
        PRNI
        BRN   2
        HALT
```

Of course your parser-only system won't actually be able to *assemble* the code or generate PVM code for the interpreter. In a few weeks time we might extend it further to do that - for the moment simply describe the language.

Wait! we are not finished yet. You might have noticed that the assembler you used a few weeks ago also generated a .COD file which, for the example above, would have looked like this

```
ASSEM
BEGIN
{ 0  }   DSP    2
{ 2  }   LDA    1
{ 4  }   INPI
{ 5  }   LDL    1
{ 7  }   BZE    11
{ 9  }   BRN    2
{ 11 }   LDL    1
{ 13 }   PRNI
{ 14 }   BRN    2
{ 16 }   HALT
END.
```

What you might not have noticed was that this .COD file could *also* be assembled by the assembler - the addresses in { braces } were treated as comments. Now that you have been told this, ensure that the language you describe can handle this feature as well.

Task 8 - All very logical!

Develop a Cocol grammar BOOL.ATG for describing a list of Boolean expressions like those you should remember from a Boolean Algebra course. In this context, allow your Boolean expressions to be terminated with an = operator, and to include parentheses, single-letter variables, the constants FALSE and TRUE (sometimes written as 0 and 1), and the operators *AND* (written either as AND or "&" or as a "dot", or simply omitted), *OR* (written either as OR or as "|" or as a "plus") and *NOT*, written either as a prefix NOT or as a suffix apostrophe. So some examples of Boolean expressions might be (these are in the file BOOL.TXT)

```
a AND B OR (C OR NOT D) =     a . b + (c + d') =     a b + (c + d') =
NOT (a OR b) AND TRUE =       (a + b)' . 1 =         (a + b)' AND 1 =
b AND NOT C AND D =           b . c' . d =           b c' d =
```

Note that there is a precedence ordering among Boolean operators - parentheses take precedence over *NOT*, which takes precedence over *AND*, which takes precedence over *OR*.

Test your parser out on some examples like those just shown.

And here is something to think about. Draw out the parse tree that would correspond to the expression *NOT (A OR B)*. Then draw out the parse tree that would correspond to *NOT A AND NOT B*. De Morgan will tell you that these expressions mean the same thing. Do you get the same tree in each case? Discuss the implications.

Hand in sheet - Available on the web in the ASCII file HANDIN.TXT for you to copy and edit, adding your answers.

Task 1 - Self-describing ENBF

Does the grammar as supplied obey the LL(1) constraints? Why/Why not?

How do you define a terminal to allow internal lowlines (underscores) but not allow one or more at the end?

Comment on whether the suggested changes to the productions for *QualifiedNoun* introduce non-LL(1) conformance or ambiguity.

Task 2 - Ambiguity

Is the grammar for Roman number sqquences an LL(1) grammar? Justify your answer.

Is the suggested grammar ambiguous? Justify your answer.

If the grammar is unsuitable according to the answers to the first questions, find a better grammar that does not suffer from the problems you have identified?

Task 3 - Palindromes

Does grammar 1 describe palindromes? If not, why not?

Is it an LL(1) grammar? If not, why not?

Does grammar 2 describe palindromes? If not, why not?

Is it an LL(1) grammar? If not, why not?

Does grammar 3 describe palindromes? If not, why not?

Is it an LL(1) grammar? If not, why not?

Does grammar 4 describe palindromes? If not, why not?

Is it an LL(1) grammar? If not, why not?

Can you find a better grammar to describe palindromes? If so, give it, if not, explain why not.

Task 4

Which of the following statements are true? Justify your answers.

(a) An LL(1) grammar cannot be ambiguous.

(b) A non-LL(1) grammar must be ambiguous.

(c) An ambiguous language cannot be described by an LL(1) grammar.

(d) It is possible to find an LL(1) grammar to describe any non-ambiguous language.