RHODES UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE

EXAMINATIONS: NOVEMBER 2016

Computer Science 301

PAPER 1 - Translators

Internal Examiner: Prof P.D. Terry

MARKS: 180
DURATION: 4 hours

External Examiner: Prof M. Kuttel

GENERAL INSTRUCTIONS TO CANDIDATES

1. This paper consists of 12 pages and 15 questions. Please ensure that you have a complete paper.
2. You will be provided with a separate document that indicates the hand-in procedure for the examination.

Full marks may be obtained by answering questions 1 - 13.

You may answer either or both of 14 and 15 to earn bonus marks if you have time.

Answers may be written in any medium except red ink.

A word of advice: The influential mathematician R.W. Hamming very aptly and succinctly professed that "the purpose of computing is insight, not numbers".

Several of the questions in this paper are designed to probe your insight - your depth of understanding of the important principles that you have studied in this course. If, as we hope, you have gained such insight, you should find that the answers to many questions take only a few lines of explanation. Please don't write long-winded answers - as Einstein put it "Keep it as simple as you can, but no simpler".

Good luck!

(For the benefit of future readers of this paper, various free information was made available to the students 24 hours before the formal examination. This included an augmented version of "Section C" - a request to develop set handling extensions for Parva. Some 16 hours before the examination a complete grammar and other support files for building such a system were supplied to students, along with an appeal to study this in depth (summarized in "Section D"). During the examination, candidates were given machine executable versions of the Coco/R compiler generator, the files needed to build the basic system, access to a computer, and machine readable copies of the questions.)

PLEASE DO NOT TURN OVER THIS PAGE UNTIL TOLD TO DO SO

Section A: Conventional questions [95 marks]

QUESTION A1 **[6 + 4 + 4 = 14 marks]**

A1 A compiler is often described as incorporating several "phases", as shown in the diagram below:

```
Character Handling
        |
        v
 Lexical Analysis
        |
        v
 Syntax Analysis
        |
        v
Semantic Analysis
        |
        v
 Code Generation
```

These phases are often developed in conjunction with an Error Handler and a Symbol Table Handler.

(a) Suggest - perhaps by example - the sort of errors that might be detected in each of the phases of the compilation process. [6 marks]

(b) The Parva compiler studied in the course is an example of what is often called a "one pass incremental compiler" - one in which the various phases summarized above are interleaved, so that the compiler makes a single pass over the source text of a program and generates code as it does so, without building an explicit AST. Discuss briefly whether the same technique could be used to compile one *class* of a C# program (consider the features of C# and Parva that support the use of such a technique, and identify features that seem to act against it). [4 marks]

(c) The Parva compiler supports the integer, character and Boolean types. Integer and character constants (literals) are represented in the usual way, for example 1234 (decimal) and 'X', respectively.

Suppose we wished to allow hexadecimal and binary representations of integer literals as well, for example 012FH and 01101% respectively. Which of the phases of compilation identified here would this affect, which would it not affect, and why? [4 marks]

QUESTION A2 **[6 + 6 = 12 marks]**

A2 Please combine the solutions to (a) and (b) in joint CHARACTERS and TOKENS definitions in Cocol.

(a) Write down the definition of a token that will match all the possible mother-tongue words that have each vowel appearing exactly once, and in the correct order (vowels are a, e, i, o and u; an example of such a word in English is "facetious"). [6 marks]

(b) Currency values are sometimes expressed in a format exemplified by

R12,345,101.99

where, for large amounts, the digits are grouped in threes to the left of the point. Exactly two digits appear to the right of the point. The leftmost group might only have one or two digits. Add the definition of a token that would match a valid currency value. [6 marks]

Your solution should not match an incorrect representation that has leading zeroes, like R001,123.00

```
CHARACTERS     // Feel free to add to these as you see fit
   digit =
   vowel =

TOKENS
   randAmount    =
   allVowelsOnce =
```

QUESTION A3 **[10 marks]**

Here is a Cocol description of simple mathematical expressions:

```
COMPILER Expression  $NC  /* expression.atg */
CHARACTERS
  digit      = "0123456789" .
TOKENS
  Number     = digit { digit } .
PRODUCTIONS
  Expression = Term { "+" Term  | "-" Term } .
  Term       = Factor { "*" Factor | "/" Factor } .
  Factor     = Number | "(" Expression ")" .
END Expression.
```

Two CSC 301 students were arguing very late at night / early in the morning about a prac question which read "extend this grammar to allow you to have leading unary minus signs in expressions, as exemplified by `-5 * (-4 + 6)`, and make sure you get the precedence of all the operators correct". One student suggested that the productions should be changed to read (call this GRAMMAR A)

```
Expression = [ "-" ] Term { "+" Term  | "-" Term } .     /* change here */
Term       = Factor { "*" Factor | "/" Factor } .
Factor     = Number | "(" Expression ")" .
```

while the other suggested that the correct answer would be (call this GRAMMAR B)

```
Expression = Term { "+" Term  | "-" Term } .
Term       = Factor { "*" Factor | "/" Factor } .
Factor     = Number | "-" Factor | "(" Expression ")" .  /* change here */
```

By drawing the parse trees for the string `- 12 * 4 + 5` try to decide which student was correct. Or were they both correct? If you conclude that the grammars can both accept that particular example expression, are the grammars really equivalent, or can you give an example of an expression that GRAMMAR A would accept but GRAMMAR B would reject (or vice-versa)? [10 marks]

QUESTION A4 **[8 + 2 + 16 + 2 + 4 = 32 marks]**

The contents page of a very useful textbook (if you can find a copy) begins as follows: (contents.txt)

```
All you need to know to be able to pass your compiler examination.

                              by

                           Pat Terry.

Chapter 1  Bribery is unlikely to succeed.

Chapter 2  Understand the phases of compilation.
    2.1    Lexical and syntactic analysis are easily confused
    2.2    Constraint analysis involves the concept of type
    2.3    Code generation for the PVM is a breeze

Chapter 3  Get clued up on grammars.
    3.1    Terminals
    3.2    Sentences and sentential forms
    3.3    Productions
    3.4    EBNF and Cocol
    3.5    Ambiguity is bad news
```

The following Cocol grammar attempts to describe this contents page (and others like it - there may be further chapters and further subsections, of course, and some components are optional):

```
COMPILER Contents                              // contents.atg
/* Describe the contents pages of a book
   P.D. Terry, Rhodes University, 2016 */

CHARACTERS
  uLetter = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" .
  lLetter = "abcdefghijklmnopqrstuvwxyz" .
  letter  = uLetter + lLetter .
  digit   = "0123456789" .

TOKENS
  word    = letter { letter } .
  number  = digit { digit } .
  section = digit { digit } "." digit { digit } .

IGNORE CHR(0) .. CHR(31)

PRODUCTIONS // Version 1
  Contents   = [ Title [ "by" Author [ Date ] ] ] { Chapter } .
  Title      = word { word } "." .
  Author     = word { word } "." .
  Date       = "(" number ")" .
  Chapter    = "Chapter" number Title { Subsection } .
  Subsection = section word { word } .
END Contents.
```

(a) Is this an LL(1) grammar? Justify your answer, don't just guess! [8 marks]

(b) The above grammar allows the *Contents* to be completely empty. Comment on the following attempt to change it so that (i) if the *Title* appears the list of chapters is optional but (ii) if the *Title* is absent there must be at least one *Chapter*. [2 marks]

```
PRODUCTIONS // Version 1 modified
  Contents   =   [ Title [ "by" Author [ Date ] ] ] Chapter { Chapter }
               | Title [ "by" Author [ Date ] ] { Chapter } .
```

(c) How would you add actions to the *original grammar* (Version 1) above so as to develop a system that could tell you the title of the chapter with the greatest number of subsections and also issue a warning if the contents turns out to be completely empty? [16 marks]

(A spaced version of the grammar appears in the machine readable copy of this exam paper, which you can complete and submit electronically.)

(d) Here are two other possibilities for the set of productions for this system:

```
PRODUCTIONS // Version 2
  Contents   = [ Sentence [ "by" Sentence [ "(" number ")" ] ] ] { Chapter } .
  Sentence   = Words "." .
  Words      = word { word } .
  Chapter    = "Chapter" number Sentence { Subsection } .
  Subsection = section Words .
END Contents.

PRODUCTIONS // Version 3
  Contents   = [ Words "." [ "by" Words "." Date ] ] { Chapter } EOF .
  Chapter    = "Chapter" number Words "." { section Words } .
  Words      = word { word } .
  Date       = [ "(" number ")" ] .
END Contents.
```

If I were to claim that these grammars are "equivalent" I would be using the word "equivalent" in a special way. What is meant by the statement "two grammars are equivalent"? [2 marks]

(e) Even though the sets of productions are equivalent, a developer might have reasons for preferring one set over the others. Which of these sets do you consider to be the "best", and why? [4 marks]

QUESTION A5 **[12 marks]**

Consider the description of the contents of a book in Question A4, using the productions in version 1.

Assume that you have `accept()` and `abort()` routines like those you used in this course, and a scanner `getSym()` that can recognise tokens that might be enumerated by

```
EOFSym, noSym, wordSym, numberSym, sectionSym, bySym, periodSym, chapterSym, lparenSym, rparenSym
```

How would you complete the *parser* routines below? There is no need to incorporate the actions required in (c) of Question A4 - simply show the syntax analysis. A spaced copy of this system appears in the machine readable version of the paper, which you are invited to complete and submit. [12 marks]

You are *not* required to give any of the code for the *scanner*.

```
static void Contents () {
// Contents = [ Title [ "by" Author [ Date ] ] ] { Chapter } .

} // Contents

static void Title () {
// Title = word { word } "." .

} // Title

static void Author () {
// Author = word { word } "." .

} // Author

static void Date () {
// Date = "(" number ")" .

} // Date

static void Chapter () {
// Chapter = "Chapter" number Title { Subsection } .

} // Chapter

static void Subsection () {
// Subsection = section word { word } .

} // Subsection
```

QUESTION A6 **[9 + 2 + 4 = 15 marks]**

A6 Brinch Hansen, an influential Danish computer scientist in the 1980s, incorporated an extended form of the *WhileStatement* in his experimental language Edison, instead of the usual one. Essentially this was defined as follows:

```
WhileStatement  =  "while" "(" Condition ")" Statement
                   { "or" "(" Condition ")" Statement } .
```

The dynamic semantics of this unfamiliar construct are that *Conditions* are evaluated one at a time in the order written until one is found to be true, when the corresponding *Statement* is executed, after which the process of evaluating conditions is repeated from the top. If no *Condition* is true, the loop terminates.

(a) Assuming that you want to incorporate this into Parva, and to target the PVM, how would you attribute the production to handle code generation? [9 marks]

(b) In his original proposal, the keyword *or* was, in fact, *else*. Why can we not follow Brinch Hansen and use *else* in Parva as well? [2 marks]

(c) Here is a simple example of a Brinch Hansen *WhileStatement*. Write a "regular" Parva statement (or statement sequence) that would achieve the same effect. [4 marks]

```
int a, b, c;       // assume that these will have had values defined by the time
                   // the WhileStatement is reached
while (a > b) {
  write(a, b);
  --b;
}
or (b < 10) write(b);
```

Section B: Parva with the addition of sets [85 marks]

Please note that there is no obligation to produce a fully working solution for this section. Coco/R and other files are provided so that you can enhance, refine, or test your solution if you desire. If you choose to produce a machine readable solution, you should create a working directory, unpack EXAM.ZIP, modify any files that you like, and then copy all the files back onto an exam folder on the network.

Yesterday you made history when you were invited to develop an extended Parva compiler which would provide some support for a *set* type. Later in the day you were provided with a sample solution to that challenge, and the files needed to build that system have been provided to you again today. Time to ask a few searching questions!

Your answers should, wherever possible, be expressed in "code terms", and not vague statements such as "add a Boolean parameter to the production". To illustrate, suppose you were asked the question "How would you extend the compiler to allow for statements like `x++`; or `list[i]--`;?" An excellent answer would be:

Such statements are a convenient form of assignment statement, so change the `AssignmentOrCall` *production as shown below. The Code Generator and PVM already support the* `INC` *opcode and need no further changes.*

```
   AssignmentOrCall                 (. int expType;
                                       DesType des;
                                       bool inc = true; .)
   = (    IF (IsCall(out des))         // use resolver to handle LL(1) conflict
          identifier                (. CodeGen.FrameHeader(); .)
          "("  Arguments<des>       (. CodeGen.Call(des.entry.entryPoint); .)
          ")"
        | Designator<out des>       (. if (des.entry.kind != Kinds.Var)
                                         SemError("cannot assign to " + Kinds.kindNames[des.entry.kind]); .)
***       (   AssignOp
              Expression<out expType> (. if (!Compatible(des.type, expType))
                                           SemError("incompatible types in assignment");
                                         CodeGen.Assign(des.type); .)
***         | ( "++" | "--"         (. inc = false; .)
***           )                     (. if (!IsArith(des.type))
***                                      SemError("arithmetic type needed");
***                                    CodeGen.IncOrDec(inc); .)
***       )
     ) WEAK ";" .
```

Continue now to answer the following unseen questions:

QUESTION B7 **[6 marks]**

B7 Oh dear! This code below should never compile and run, and yet it does. What two checks has the compiler writer forgotten to make, and how should the system be corrected? [6 marks]

```
void main() {                        // egB7.pav
  set a, b = set{5, 9, 7 > 4};
  write(a);
} // main
```

QUESTION B8 **[36 marks]**

B8 The system as supplied to you allows for the operations of set definition, forming the union of two sets, including further elements into a set, testing a value for membership of a set, cloning (copying) a set, comparing two sets for equality, determining the number of elements a set contains, and writing a set in a simple way, as illustrated by the following otherwise rather pointless program:

```
void main() {                        // egB8A.pav
  set a, b, c, d;
  a = set{5, 9};
  b = set{7, 4, 9};
  b.Incl(9, 1);                      // set b is now {1, 4, 7, 9}
  c = a + b;                         // union (  {1, 4, 5, 7, 9} )
  d = Clone(b);                      // cloning a set to make a distinct copy
  bool e = 3 in b;                   // test for membership (false)
  if (Equals(a, b))                  // in this case they are not equal
    write("compiler error");
  write(Members(b));                 // number of elements (4)
  write(" set b is " , b);           //   set b is {1, 4, 7, 9}
} // main
```

Extend the compiler and interpreter to allow for the operations of forming the difference of two sets, excluding elements from a set, and testing for a set being empty, as illustrated in the following otherwise equally pointless program: [3 x 12 = 36 marks]

```
void main() {                        // egB8B.pav
  set a, b, c, d, e;
  a = set{15, 3, 9};
  b = set{7, 4, 9, 1};
  c = b - a;                         // difference (  {1, 4, 7} )
  d = set{};                         // empty set
  write(IsEmpty(b), IsEmpty(d));     // false, true
  b.Excl(1, 7);                      //  set b is now {4, 9}
} // main
```

QUESTION B9 **[4 marks]**

B9 A tutor spotted a student writing Parva code that included a scan of a string of binary digits:

```
char ch;
read(ch);
while (ch in set{'0', '1'})
  read(ch);
```

"That will work", said the tutor, "but to save both space and time you would be better advised to write the code as follows".

```
set digits = {'0', '1'};
read(ch);
while (ch in digits)
  read(ch);
```

Why did the tutor say that the modification would save both space and time? [4 marks]

QUESTION B10 **[5 marks]**

B10 The compiler supplied to you has the ability to mark variables "final" as was suggested in your last practical exercise. Surely, to all intents and purposes a declaration like

```
const x = 10;
```

has exactly the same effect as the declaration

```
final int x = 10;
```

because we should expect a compilation error if we were to follow either declaration with the statement:

```
x = x + 1;
```

"Not so" said the language lawyer. "While it is true that in this case both would generate the same sort of error, there are subtle differences in the way these declarations can be used".

Lawyers produce evidence. Identify and explain the alleged differences by considering the PVM code that would be generated for each of the following snippets [5 marks]

```
(a)  int y;                 (b)  int y;
     const x = 10;               final int x = 10;
     y = x + 15;                 y = x + 15;
```

QUESTION B11 **[5 marks]**

B11 Oh dear! A rigorous test of the compiler supplied to you will reveal that compiling code like

```
void main() {                     // egB11.pav
  final set vowels  = set{'a', 'e', 'i', 'o', 'u' };
  final char ch = 'T';
  ++ch;
  vowels = vowels + set{'A', 'E', 'I'};
  vowels.Incl('O', ch);
} // main
```

will not report any errors, when of course, it should. Identify and explain the errors and modify the system to detect and report them correctly. [5 marks]

QUESTION B12 **[5 marks]**

B12 You will have been taught that when an argument is passed to a function "by value", any changes made to the corresponding formal parameter should not be felt in the caller. However, if you compile and execute the following Parva code

```
void slave(set a) {               // egB12.pav
  writeLine(a);
  a.Incl(45);                     // corrupt set a
  writeLine(a);
} // slave

void main() {
  set scrap = set{30, 40, 50};
  slave(scrap);                   // pass scrap to the slave (by value)
  writeLine(scrap);
}  // main
```

the output will be

```
{30, 40, 50}
{30, 40, 45, 50}
{30, 40, 45, 50}
```

when of course it should be

{30, 40, 50}
{30, 40, 45, 50}
{30, 40, 50}

You can surely do better than that. Show us! (Hint: this can be solved very simply. Look for the very simple solution and don't get carried away!) [5 marks]

QUESTION B13 **[24 marks]**

B13 In the system as so far developed, there is only one set type, the elements of which are limited to non-negative integers. A more rigorous system might wish to distinguish between sets of characters and sets of integers, as suggested by the following code:

```
void main () {                      // egB13.pav
  set     ints  = set{30, 20, 10};
  charset chars = charset {'A', 'E', 'I', 'O', 'U' };
  chars.Incl('a', 'b');             // legal
  writeLine(chars);                 //  {'A', 'E', 'I', 'O', 'U', 'a', 'b' };
  chars.Incl(10);                   // illegal
  ints.Incl('a', 35);               // legal, by analogy with C# char/int
  writeLine(ints);                  //  {10, 20, 30, 35, 97};
} // main
```

Identify and make the changes that would be needed to the system to accommodate these examples. (Several more changes might be needed were you to explore this fully, which time will not permit.) [24 marks]

QUESTION O14 (Bonus - optional) **[3 + 5 = 8 marks]**

O14 After careful consideration of the overwhelming success of the Parva language as extended this weekend, its inventor has decided to release Parva++. However, in this language, the production for *IfStatement* is to be changed to incorporate a further key word, `fi`.

```
IfStatement     = "if" "(" Condition ")" Statement
                    [ "else" Statement ]
                  "fi" .
```

The proud inventor was overheard to say that he was tired of explaining the famous "dangling else" ambiguity to students and that this syntactic change would eliminate it once and for all.

(a) What do you understand by the "dangling else" problem? [3 marks]

(b) Analyse in some detail whether the inventor's claim is correct. [5 marks]

QUESTION O15 (Bonus - optional) **[10 marks]**

O15 Howls of anguish will arise when Parva++ is released. All those millions of lines of code produced in the Hamilton Laboratory by generations of students will, at a stroke, be rendered syntactically incorrect.

The inventor is unperturbed, and promises to use Coco/R to write a quick Parva to Parva++ translator. He argues that all he needs to do is to write a sort of pretty printer that can reformat existing Parva programs, with extra `fi` tokens inserted into *IfStatements* at the appropriate places, and sketches out a few T-diagrams to show the various steps in the process.

What do the T-diagrams look like? (Make the assumption that you have available the executable versions of the Coco/R compiler-generator for C#, an executable version of a C# compiler, the Cocol grammars for Parva and for Parva++, as well as Parva and Parva++ compilers built from these grammars, and at least one Parva program `sample.pav` to be converted to `sample.p++` and then compiled.) [10 marks]

(A set of blank T-diagrams has been supplied as an addendum and in the electronic copy of this paper.)

END OF EXAMINATION QUESTIONS

Section C

(Summary of free information made available to the students 24 hours before the formal examination.)

Candidates were provided with the basic ideas for adding simple set handling primitives to the Parva language, and were invited to modify a supplied Parva implementation to accommodate these.

They were provided with an exam kit for C#, containing a working Parva compiler like that which they had used in the practical course. They were also given a suite of simple, suggestive test programs. Finally, they were told that later in the day some further ideas and hints would be provided.

Section D

(Summary of free information made available to the students 16 hours before the formal examination.)

A complete Parva system incorporating the features they had been asked to implement was supplied to candidates in a later version of the examination kit. They were encouraged to study it in depth and warned that questions in Section B would probe this understanding; few hints were given as to what to expect, other than that they might be called on to comment on the solution, and perhaps to make some modifications and extensions. They were also encouraged to spend some time thinking how any other ideas they had during the earlier part of the day would need modification to fit in with the solution kit presented to them. The system as supplied at this point was deliberately naïve in some respects, in order to lay the ground for the unseen questions of the following day.

ADDENDUM (Question A4)

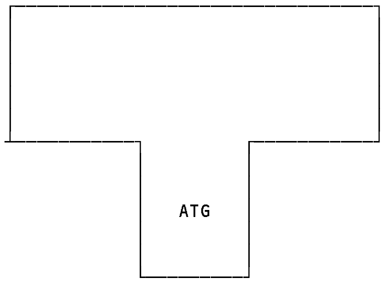
Student number | | | | | | | | eg | 6 | 3 | T | 0 | 8 | 4 | 4 |

```
COMPILER Contents


CHARACTERS
  ...

TOKENS
  ...

PRODUCTIONS // Version 1
  Contents
  = [ Title
      [ "by" Author
        [ Date
        ]
      ]
    ] { Chapter
    }
  .

  Title
  = word
    { word
    } "."
  .

  Author
  = word
    { word
    } "."
  .

  Date
  = "("
    number
    ")"
  .

  Chapter
  = "Chapter"
  number
  Title
  { Subsection
  }
  .

  Subsection
  = section
  word
  { word
  }
  .

END Contents.
```

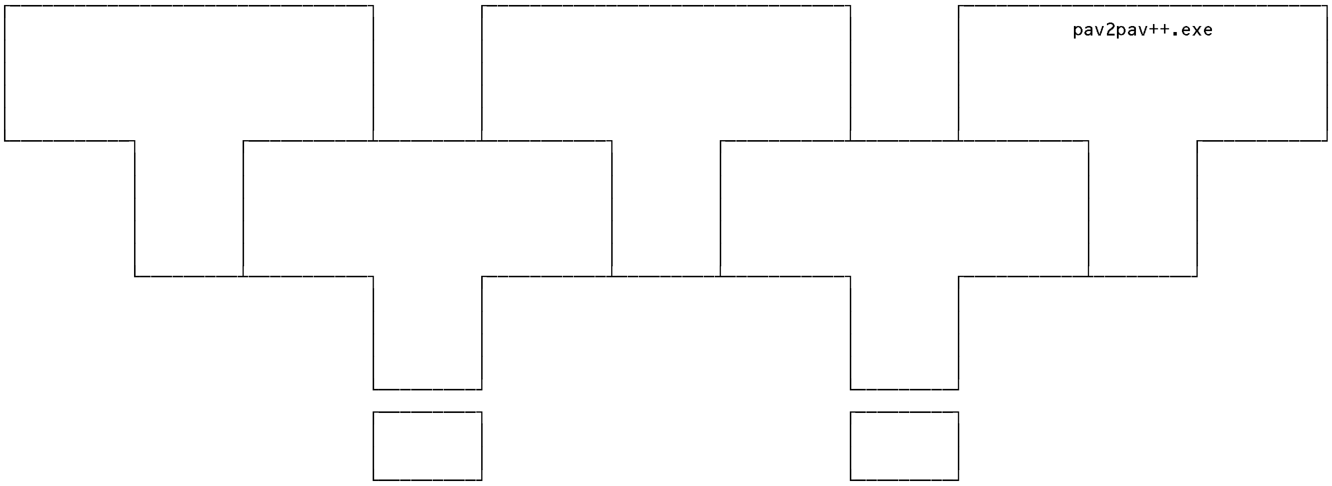
ADDENDUM (Question O15)

Student number								eg	6	3	T	0	8	4	4

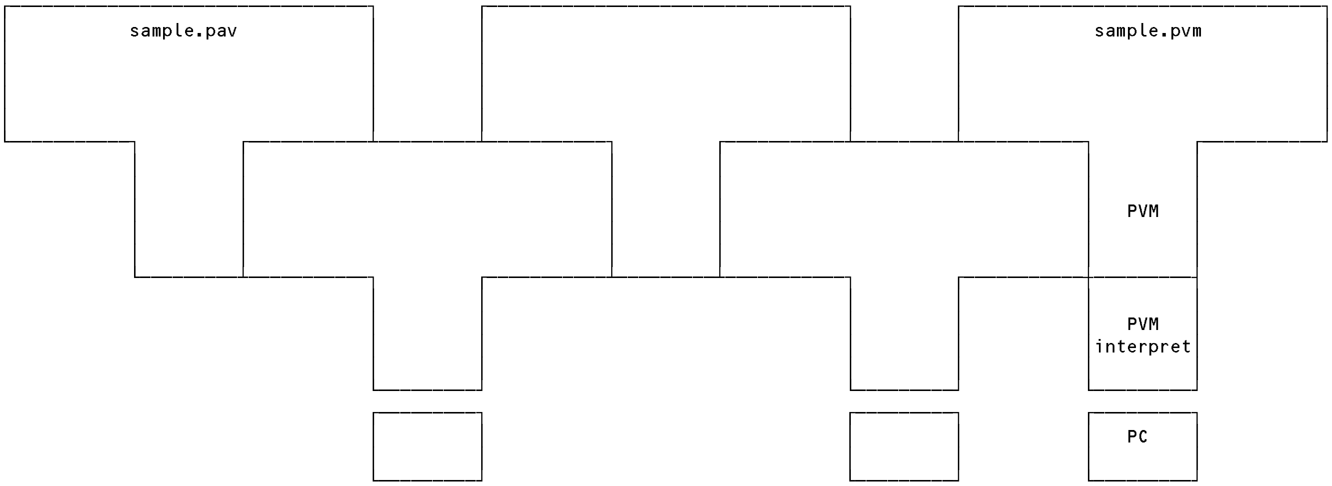
Develop a Cocol grammar for the high level translator (.ATG file)



Develop the high level translator



Convert and compile sample.pav


sample.pav
sample.pvm
PVM
PVM interpret
PC