RHODES UNIVERSITY
Where leaders learn

Computer Science 301

Programming Language Translation

Practical 4: week beginning 5th August 2018

Hand in this prac sheet before the start of your next practical, correctly packaged in a transparent folder with your solutions and the "cover sheet". Unpackaged and late submissions will not be accepted. Since the practical is 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. 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 yourself 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.

Copies of this handout, the cover sheet, the Coco reference manual and "Pitfalls using Coco" are available on the RUConnected course page.

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 (30 marks)

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

- The solutions to tasks 2, 3, 4 and 5 on the hand-in sheet or an edited copy.
- Listings of your solutions to the grammar problems (tasks 1 and 5), 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) for tasks 1 and 5, submitted via RUConnected.

Please submit your prac in the "hand-in" box inside the laboratory at the start of the next practical session. I do NOT require listings of any C# code produced by Coco/R. Note that some of the tasks do not need you to use a computer, nor should you. Do them by hand.

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

A rule not stated there, but which should be obvious, is that you are not allowed to hand in another student's or group'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 on *all* cover sheets and with suitable comments typed into *all* listings. You are expected to be familiar with the University Policy on Plagiarism.

Task 0 Creating a working directory and unpacking the prac kit

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

Copy the zipped prac kit into a new directory/folder in your file space, either from the server (I:) or RUConnected.

```
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 2 (palindromes), 4 (family) and 5 (reverse polish).

After unpacking this kit attempt to make the parsers, as you did last week. You can get Coco/R to show you the *FIRST* and *FOLLOW* sets for the non-terminals of the grammar by using the "-options f" flag (change the cmake file to reflect this or just execute coco itself from the command line. Verify that the objections (if any) that Coco/R raises to these grammars are the same as you have determined by hand.

Task 1 Revisiting Parva precedence [5 marks] HAND_IN BY 5PM TODAY

The Parva grammar you met last week incorporated the following productions to describe the syntax of an Expression. This uses the hierarchy of operator precedence that is used in Wirth's languages Pascal, Modula-2 and Oberon.

```
Expression        = AddExp [ RelOp AddExp ] .
AddExp            = [ "+" | "-" ] Term { AddOp Term } .
Term              = Factor { MulOp Factor } .
Factor            =   Designator | Constant
                    | "new" BasicType "[" Expression "]"
                    | "!" Factor | "(" Expression ")" .
AddOp             = "+" | "-" | "||" .
MulOp             = "*" | "/" | "&&" | "%" .
RelOp             = "==" | "!=" | "<" | "<=" | ">" | ">=" .
```

Without adding any more operators - that is, restricting yourself to + - * / % && || ! and the relational operators == != < > <= and >= - develop the part of a Parva grammar for Expression that would recognize Expressions where the operator precedence is the same as it is in C#. You may need to look up the operator precedence rules in C#.

If you wish, incorporate this grammar for Expression into the Parva grammar you had last week for test purposes.

Is your new grammar equivalent to the given one?

A listing of your grammar for Task 1 must be handed in (via the hand-in box) by the end of the prac afternoon (5:00pm). Remember to include the names of the group members as a comment at the top!

Task 2 Palindromes [4 marks]

Palindromes are character strings that read the same from either end, like "Hannah" or “madam". The following represent various attempts to find grammars (see `Palinx.atg`) 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 3 Thinking about ambiguity [4 marks]

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.

Task 4 Roman numerals [6 marks]

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 */

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 5 Railway management needs some help! [6 marks]

(A second chance to get this completed!)

In some parts of the world, railway trains fall into three categories - passenger trains, freight trains, and mixed passenger and freight trains. A train may have one (or more) locomotives (or engines). Behind the locomotive(s) in freight or mixed trains come one or more freight trucks. A freight train is terminated by a brake van after the last freight truck; a mixed train, on the other hand, follows the freight trucks with one or more passenger coaches, the last (or only one) of which must be a so-called guard's van. A passenger train has no freight trucks; behind the locomotive(s) appears only a sequence of coaches and the guard's van. Freight trucks come in various forms - open trucks, closed trucks, fuel trucks, coal trucks and cattle trucks. Sometimes locomotives are sent along the line with no trucks or coaches if the intention is simply to transfer locomotives from one point to another.

Here is a Cocol grammar that describes a list of correctly formed trains (`Trains.atg`):

```
COMPILER Trains $CN
/* Grammar for simple railway trains
   P.D. Terry, Rhodes University */

IGNORECASE

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

IGNORE CHR(0) .. CHR(31)

PRODUCTIONS
  Trains    = { OneTrain } EOF .
  OneTrain  = LocoPart [ [ GoodsPart ] HumanPart ] SYNC "." .
  LocoPart  = "loco" { "loco" } .
  GoodsPart = Truck { Truck } .
  HumanPart = "brake" | { "coach" } "guard" .
  Truck     = "coal" | "closed" | "open" | "cattle" | "fuel" .
END Trains.
```

As an interesting variation, and in the interests of safety, modify the grammar to build in the restrictions that fuel trucks may not be placed immediately behind the locomotives, or immediately in front of a passenger coach.

There are, of course, several ways in which these extensions might be done. You may find that you come up with grammars that look plausible but which are, in fact, not quite correct; the problem may be harder than it first appears. A sample set of correct and incorrect trains has been provided in the source kit (`Trains.txt`), and it is easy to make up examples of your own.

Develop a better grammar and generate and test a parser that will accept safe trains and reject incorrect trains, either because the rolling stock is marshalled in the wrong order, or because the safety regulations are violated.
