Computer Science 3 - 2009

Programming Language Translation

Practical for Week 22, beginning 28 September 2009

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 g03A1234.** 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/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 and 3 by 18h00 this afternoon on the attached sheet*.
- Listings of your solutions to the grammar problems, produced on the laser printer by using the LPRINT utility or UltraEdit in "two up" format.
- Electronic copies of your grammar files (ATG files).

I do NOT require listings of any Java 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 at:

`http://www.scifac.ru.ac.za/plagiarism_policy.pdf`

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

Task 1 - 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 Tim, Tsepiso, Isaac, Nancy, Ashraf, Elvis, Karabo and their comrades present what are generously called shows. These include music and advertisements (since 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 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 "music" 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 attempt to describe a day's activities might be on the lines of

Radio = { *TalkShow* | *NewsBulletin* | "music" | "advert" } EOF .
NewsBulletin = "advert" *NewsItem* { *NewsItem* } [*Weather*] *Filler* .
NewsItem = "ANC" ["cosatu"] | "strike" | ["cosatu"] "ANC" | "zuma" | "corruption" | "murder" .
TalkShow = "host" { "listener" "host" } [*Filler*] .
Filler = "music" | "advert" .
Weather = { "snow" | "rain" | "cloudy" | "windy" } .

Analyze this grammar in detail. 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. 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.

Task 2 - Palindromes

Palindromes are character strings that read the same from either end, like "Hannah" or my brother's favourite line when he did the CTM ads: "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 3 - 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 3 before continuing.

Task 4 - Grab a mug of hot Coco and press on

There are several files that you need, zipped up this week in the file `PRAC22.ZIP` (Java version) or `PRAC22C.ZIP` (C# version)

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

```
j:
md  prac22
cd  prac22
copy  i:\csc301\trans\prac22.zip
unzip  prac22.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 5 - Phools rush in (where angels fear to tread, apparently)

In the land of Phools the official language is Phoolish. A Phoolish professor noticed that many of his students had not mastered the syntax of Phoolish well. Tired of correcting their many syntactical mistakes, he decided to challenge the students and asked them to write a program that could check the syntactical correctness of any sentence they wrote. Similar to the nature of Phools, the syntax of Phoolish is also pleasantly simple. Here are the rules:

0. The characters in the language are the 26 letters of the standard western alphabet and the language distinguishes between lower and upper case letters.

1. Every lower case letter is a simple correct sentence.

2. If σ is a correct sentence, then so is a sentence consisting of an upper case vowel followed by σ, that is Aσ Eσ Iσ Oσ or Uσ

3. If σ and τ are correct sentences, then so is a sentence consisting of an upper case consonant followed by $\sigma\tau$, for example B$\sigma\tau$ C$\sigma\tau$ D$\sigma\tau$ … Z$\sigma\tau$.

4. Rules 0 to 3 are the only rules that determine the syntactical correctness of a sentence.

5. As a special case. the character "." is used to terminate a sequence of sentences.

Develop a parser for a sequence of Phoolish sentences, and test it out on some snippets of Phoolish language.

Task 6 - How are things stacking up?

Develop a Cocol grammar that describes programs written in the PVM code that you struggled with in Practical 20. 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 here:

```
        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. 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 last week 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 - Reverse Polish Notation

You may be familiar with RPN or "Reverse Polish Notation" as a notation that can describe expressions without the need for parentheses. The notation eliminates parentheses by using "postfix" operators after the operands. To evaluate such expressions one uses a stack architecture, such as forms the basis of the PVM machine studied in the course. Examples of RPN expressions are:

```
3 4 +                   - equivalent to    3 + 4
3 4 5 + *               - equivalent to    3 * (4 + 5)
```

In many cases an operator is taken to be "binary" - applied to the two preceding operands - but the notation is sometimes extended to incorporate "unary" operators - applied to one preceding operand:

```
4 sqrt                  - equivalent to    sqrt(4)
5 -                     - equivalent to    -5
```

Here are two attempts to write grammars describing an RPN expression:

```
(G1)    RPN       =     RPN RPN binOp
                      | RPN unaryOp
                      | number .
        binOp     =   "+" | "-" | "*" | "/" .
        unaryOp   =   "-" | "sqrt" .
```

and

```
(G2)    RPN       =   number REST .
        REST      =   [ number REST binOp REST | unaryOp ].
        binOp     =   "+" | "-" | "*" | "/" .
        unaryOp   =   "-" | "sqrt" .
```

Are these grammars equivalent? Is either (or both) ambiguous? Do either or both conform to the LL(1) conditions? If not, explain clearly where the rules are broken, and come up with an LL(1) grammar that describes RPN notation, or else explain why it might be necessary to modify the language itself to overcome any problems you have uncovered.

Task 9 - Eliminating MetaBrackets

One of last week's tasks was to extend the four function calculator. A solution to that task is given below.

Modify the solution to give a syntactically equivalent grammar, but your solution may not use the { } or [ ] "metabrackets", although it can use the ( ) parentheses if you like. The modified grammar must still be LL(1).

```
COMPILER Calc1 $CN
/* Simple four function calculator - extended
   P.D. Terry, Rhodes University, 2009 */

CHARACTERS
  digit      = "0123456789" .
  hexdigit   = digit + "ABCDEF" .

TOKENS
  decNumber  = digit { digit } .
  hexNumber  = "$" hexdigit { hexdigit } .

IGNORE CHR(0) .. CHR(31)

PRODUCTIONS
  Calc1      = { Expression "=" } EOF .
  Expression = ["+" | "-" ] Term { "+" Term  | "-" Term } .
  Term       = Factor { "*" Factor | "/" Factor } .
  Factor     = Primary { "!" } .
  Primary    = decNumber | hexNumber | [ "abs" ] "(" Expression ")" .
END Calc1.
```

Hint: You may not yet have noticed that Coco allows you to write an empty option to a production simply by leaving it empty, for example

```
Title = "Mr" | "Mrs" | "Ms" | "Dr" | Professor" | .  /* nothing beteween the last | and the . */
```

Hand in sheet:

Task 1 - Listen to Steam Radio

What form does your grammar take when you eliminate the meta-brackets?

Which, if any, productions break the LL(1) rules, and why?

Can you find an equivalent grammar that does obey the LL(1) constraints?
If so, give it. If not, explain why you think it canot be done.

Task 2 - 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 3

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.