Computer Science 3 - 2008

Programming Language Translation

Practical for Week 22, beginning 22 September 2008

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, 3 and 4 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 a small Courier font (take care; lines can get quite wide!)
- 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 30 page (!) University Policy on Plagiarism, which you can consult at:

`http://www.scifac.ru.ac.za/plag2008.doc`

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

Task 1 - Meet the family

Consider the following grammar:

```
COMPILER Home
IGNORE CHR(0) .. CHR(31)
PRODUCTIONS
  Home      = Family { Pets } [ Vehicle ] "house" .
  Pets      = "dog" [ "cat" ] | "cat" .
  Vehicle   = ( "scooter" | "bicycle" ) "fourbyfour" .
  Family    = Parents { Children } .
  Parents   = [ "Dad" ] [ "Mom" ] | "Mom" "Dad" .
  Child     =   "Helen" | "Margaret" | "Alice" | "Robyn" | "Cathy"
              | "Janet" | "Anne" | "Ntombizodwa" | "Ntombizanele" .
END Home.
```

Analyse this grammar in detail. Is it LL(1) compliant? If not, why not?

Task 2 - Expressions - again

The following grammar attempts to describe expressions incorporating the familiar operators with their correct precedence and associativity.

```
COMPILER Expression
IGNORE CHR(0) .. CHR(31)
PRODUCTIONS
  Expression = Term    { ( "+" | "-" ) Term  } .
  Term       = Factor  { ( "*" | "/" ) Factor } .
  Factor     = Primary [ "↑" Expression ] .
  Primary    = "a" | "b" | "c" .
END Expression.
```

Is this an ambiguous grammar? (Hint: try to find an expression that can be parsed in more than one way). Is it an LL(1) grammar? If not, why not, and can you find a suitable grammar that *is* LL(1)?

Task 3 - Palindromes - again

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 ways of finding 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, 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 6 - Deja vu all over again - the joys of CSC 201 assembler

Develop a Cocol grammar that describes programs written in the ASM code that you struggled with in second year CS201 (the second-best course you have taken in Computer Science; no prizes for guessing which is the best course!). That should be pretty easy, but be careful to describe the language as tightly as possible. To refresh your memory here is a simple example program (BITS.ASM) (programs can get more complicated than this one, so be careful):

```
        BEG             ; Count the bits in a number
        CLA             ; CPU.A := 0
        STA     BITS    ; BITS := 0
        INI             ; Read(CPU.A)
LOOP                    ; REPEAT
        SHR             ;  CPU.A := CPU.A DIV 2
        BCC     EVEN    ;  IF CPU.A MOD 2 # 0 THEN
        PSH             ;    save CPU.A on stack
        LDA     BITS
        INC
        STA     BITS    ;    BITS := BITS + 1
        POP             ;    restore CPU.A
EVEN    BNZ     LOOP    ; UNTIL CPU.A = 0
        LDA     BITS    ;
        OTI             ; Write(BITS)
        HLT             ; terminate execution
BITS    DS      1       ; BYTE BITS
        END
```

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 develop a grammar to describe the language, for which a long-winded summary can be found on the web at

`http://www.scifac.ru.ac.za/lowlevel/app.htm`

Task 7 - Parva expressions are not like those in C# and Java

The grammar for expressions in Parva employs very few levels of operator precedence, corresponding exactly to the levels found in Pascal and its derivatives. Modify the Parva grammar from last week's practical so that it recognizes expressions whose precedence rules are equivalent to those found in C# or Java. Do nnt try to include `++` and `--` operators in your expressions, however.

Since I am a kindly old soul, this week's prac kit contains a correct version of `Parva.atg` that you can use if you did not get your own solution completed.

Something else to think about. Why do you suppose languages derived from C have so many levels of precedence and the rules they have for associativity? What do all these levels offer to a programmer that languages modelled on Pascal might appear to withhold? Do Pascal-like languages *really* withhold these features?

Hand in sheet:

Task 1 - Meet the family

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 - Expressions - again

Is this an ambiguous grammar? If so, why? is it an LL(1) grammar? If not, why not, and can you find a suitable grammar that *is* LL(1)?

Task 3 - Palindromes - again

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.