Programming Languages 3

Tutorial Solutions (2013-14)

Here are sample solutions to most of the tutorial exercises. For some of the exercises alternative correct solutions are possible. If in doubt, consult the lecturer.

Attempt each exercise before consulting the sample solution.

Exercises 1 (Syntax) – Solutions

1A. *(Regular expressions)*

(a) Syntax of Cobol identifiers using a single RE:

(‘a’ | ‘b’ | ‘c’ | … | ‘z’) (‘a’ | ‘b’ | ‘c’ | … | ‘z’ | ‘0’ | ‘1’ | ‘2’ | … | ‘9’)*
(‘-’ (‘a’ | ‘b’ | ‘c’ | … | ‘z’ | ‘0’ | ‘1’ | ‘2’ | … | ‘9’)$^{+}$)*

(b) Syntax of Cobol identifiers using EBNF:

id = *letter* (*letter* | *digit*)* (‘-’ (*letter* | *digit*)$^{+}$)*
letter = ‘a’ | ‘b’ | ‘c’ | … | ‘z’
digit = ‘0’ | ‘1’ | ‘2’ | … | ‘9’

1B. *(Regular expressions)*

(a) Expressing the given `grep` patterns in standard RE notation:

‘b’ (‘a’|‘e’|‘i’) ‘t’ ‘b’ (…|…|…) ‘t’
‘b’ ‘e’* ‘t’ ‘b’ (‘a’|‘e’|‘i’|‘o’|‘u’)*‘t’ ‘b’ (‘a’|‘e’|‘i’)$^{+}$ ‘t’

where (…|…|…) is a choice between *all* available graphic characters!

(b) To find the required patterns in file `f`:

(i) `egrep "<H[123456789]>" f` *or*
`egrep "<H[1-9]>" f`
(ii) `egrep "{[a-z]+}" f`
(iii) `egrep "{.*}" f`
(iv) `egrep "M(r|s|rs|iss)" f`
(v) `egrep "b(an)*a" f`

1C. *(BNF)*

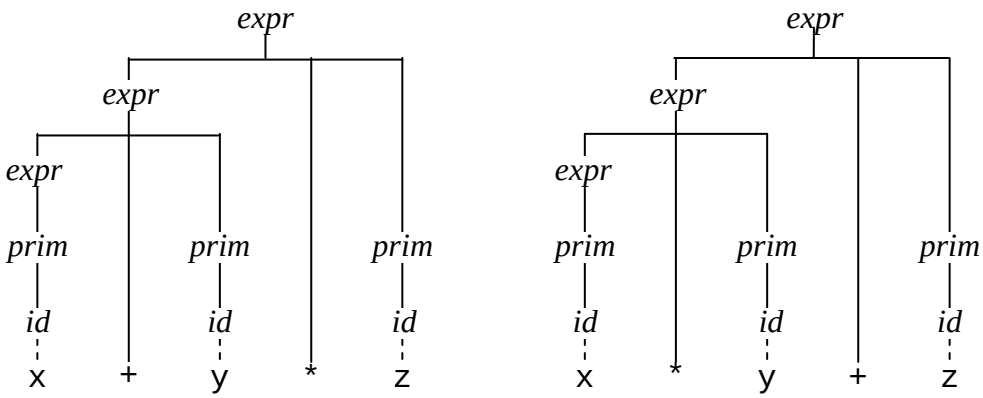
Mini-English grammar modified to enforce subject–verb agreement, including 1st, 2nd, and 3rd persons:

sentence = *subject-1 verb-1 object* ‘.’
| *subject-2 verb-2 object* ‘.’
| *subject-3 verb-3 object* ‘.’
subject-1 = ‘I’
subject-2 = ‘you’
subject-3 = ‘a’ *noun* | ‘the’ *noun*
verb-1 = ‘see’ | ‘smell’
verb-2 = ‘see’ | ‘smell’
verb-3 = ‘sees’ | ‘smells’

The production rules for *object* and *noun* are unaffected.

1D. *(Phrase structure)*

(a) Syntax trees of the expressions “`x+y*z`” and “`x*y+z`”:



(b) Grammar modified so that '`*`' and '`/`' have greater priority:

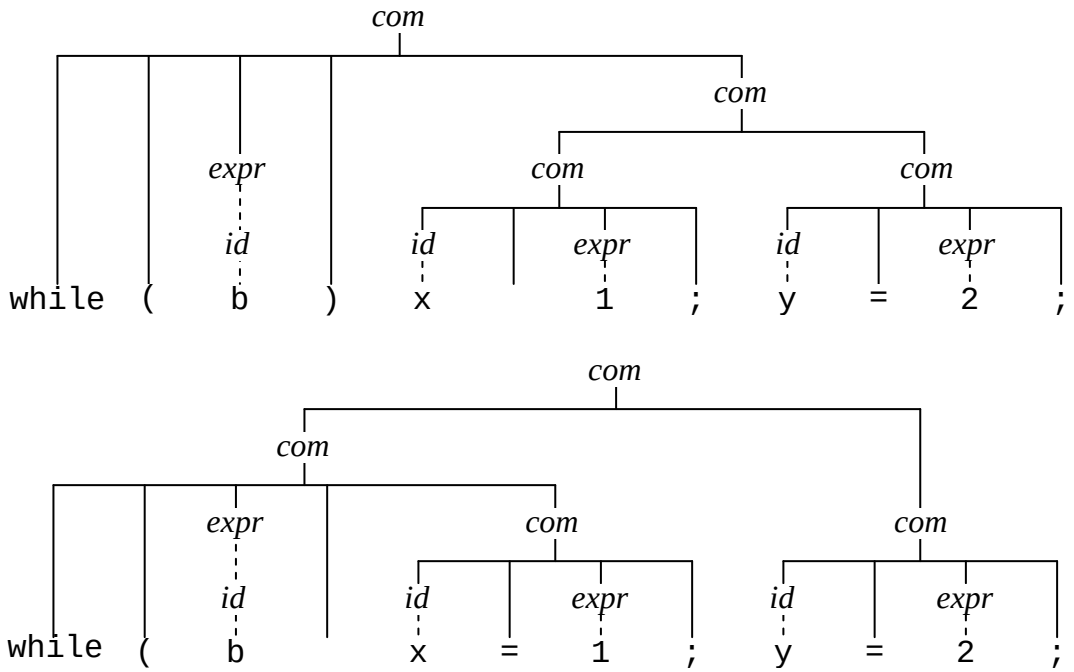
expr = *term*
| *expr* '`+`' *term*
| *expr* '`-`' *term*

term = *prim*
| *term* '`*`' *prim*

prim = *num*
| *id*
| '`(`' *expr* '`)`'

1E. *(Ambiguity)*

(a) The phrase "`while (b) x = 1; y = 2;`" is ambiguous:



(b) The Fun grammar avoids this ambiguity by allowing the body of a while-command to be a sequential command, but insisting that it is terminated by '`.`'.

(c) The Java grammar avoids this ambiguity by insisting that the body of a while-command is a single command, not a sequence of commands. However, a sequence of commands can be made into a single command by enclosing it in curly brackets "`{…}`".

1F. *(EBNF)*

Modified grammar:

com = …
| ‘`for`’ ‘`(`’ *id* ‘`=`’ *expr* ‘`..`’ *expr* ‘`)`’
com
| ‘`case`’ ‘`(`’ *expr* ‘`)`’ ‘`{`’
(‘`when`’ *num* ‘`:`’ *com*)$^{+}$
(‘`otherwise`’ ‘`:`’ *com*)$^{?}$ ‘`}`’
| *id* ‘`(`’ *actuals* $^{?}$ ‘`)`’ ‘`;`’

decl = …
| ‘`procedure`’ *id* ‘`(`’ *formals* $^{?}$ ‘`)`’ ‘`{`’
com ‘`}`’

formals = *type id* (‘`,`’ *type id*)*

actuals = *expr* (‘`,`’ *expr*)*