CSc 4330, Spring 2018
Exam 1 Solutions

---

(1) S => NP VP => the N VP => the N V NP => the N V the N
{cat, song, canary} x {sings, eats}, x {cat, song, canary}
18 combinations

the cat sings the cat
the cat sings the song
the cat sings the canary
the cat eats the cat
the cat eats the song
the cat eats the canary
the song sings the cat
the song sings the song
the song sings the canary
the song eats the cat
the song eats the song
the song eats the canary
the canary sings the cat
the canary sings the song
the canary sings the canary
the canary eats the cat
the canary eats the song
the canary eats the canary

(2)

leftmost derivation
E => + E E
  => + * E E E
  => + * - E E E E
  => + * - x E E E
  => + * - x y E E
  => + * - x y x E
  => + * - x y x y

rightmost derivation
E => + E E
  => + E y
  => + * E E y
  => + * E x y
  => + * - E E x y
  => + * - E y x y
  => + * - x y x y

parse tree

(3) Grammar:

1. S --> aS
2. S --> aSbS
3. S --> epsilon

Two rightmost derivations:

S 2=> aSbS
  3=> aSb
  1=> aaSb
  3=> aab

S 1=> aS
  2=> aaSbS
  3=> aaSb
  3=> aab

Two different parse trees:

(4)

**Attributes:**

(1) max; synthesized; associated with btree - max value stored in tree rooted at btree
(2) min; synthesized; associated with btree - min value stored in tree rooted at btree
(3) value; intrinsic; associated with NUMBER

**Attribute grammar:**

**Syntax Rule:** btree --> NIL
**Semantic Rules:**
  btree.max = -MAXINT
  btree.min = MAXINT

**Syntax Rule:** btree --> LPAREN btree COMMA NUMBER COMMA btree RPAREN
**Semantic Rules:**
  btree[1].max <-- max(btree[3].max, NUMBER.value)
  btree[1].min <-- min(btree[2].min, NUMBER.value)
**Predicate:**
  ((NUMBER.value > btree[2].max) && (NUMBER.value < btree[3].min))

(5) **Universe of objects**: numbers and list of numbers.
Mseq(seq) maps list-expressions to list of numbers
Mlisp(lisp) maps lisp expressions to numbers

```
Mseq(seq) = case seq of
  lisp:
    if (Mlisp(lisp) == error)
      error
    else
      makeEmpty().addNumberToList(Mlisp(lisp))
  lisp seq:
    if ((Mlisp(lisp) == error) || (Mseq(seq) == error))
      error
    else
      Mseq(seq).addNumberToList(Mlisp(lisp))

Mlisp(lisp) = case lisp of
  INT: Mint(INT)
  LPAREN ADD lisp lisp RPAREN:
    if ((Mlisp(lisp[1]) == error || (Mlisp(lisp[2]) == error))
      error
    else
      Mlisp(lisp[1]) + Mlisp(lisp[2])
  LPAREN DIV lisp lisp RPAREN:
    if ((Mlisp(lisp[1]) == error || (Mlisp(lisp[2]) == error))
      error
    else
      if (Mlisp[2] == 0)
        error
      else
        Mlisp(lisp[1]) / Mlisp(lisp[2])
  LPAREN CAR LPAREN seq RPAREN RPAREN:
    if (Mseq(seq) == error)
      error
    else if (Mseq(seq).isEmpty())
      error
    else
      Mseq(seq).first()
```

(6) Modifications shown in red.

```
lisp : INT
     | VAR
     | LPAREN ADD lisp lisp RPAREN
     | LPAREN SUB lisp lisp RPAREN
     | LPAREN MUL lisp lisp RPAREN
     | LPAREN DIV lisp lisp RPAREN
     | LPAREN CAR list RPAREN
     | LPAREN LET LPAREN pairs RPAREN lisp RPAREN
list : LPAREN CDR list RPAREN | LPAREN seq RPAREN
seq : lisp | lisp seq
pairs : pair | pair pairs
pair : LPAREN VAR lisp RPAREN
```

(7)

Rightmost derivation for a + b + b

```
S => A
  => A + B
  => A + b
  => A + B + b
  => A + b + b
  => a + b + b
```

```
0       a+b+b$  S2
0a2     +b+b$   R2 (GOTO[0,A]); A --> a
0A1     +b+b$   S3
0A1+3   b+b$    S5
0A1+3b5 +b$     R3 (GOTO[3,B]); B --> b
0A1+3B4 +b$     R1 (GOTO[0,A]); A --> A + B
0A1     +b$     S3
0A1+3   b$      S5
0A1+3b5 $       R3 (GOTO[3,B]); B --> b
0A1+3B4 $       R1 (GOTO[0,A]); A --> A + B
0A1     $       R0 (GOTO[0,S]); S --> A
0S0     $       accept
```

(8)

```
grammar Prolog;

term : VAR | NUMBER | NAME LPAREN terms RPAREN | LBRACK terms RBRACK;
terms : term | term COMMA terms;

LPAREN : '(';
RPAREN : ')';
LBRACK : '[';
RBRACK : ']';
COMMA : ',';
NAME : ('a'..'z')(('a'..'z')|('A'..'Z')|('0'..'9'))*;
VAR : ('A'..'Z')(('a'..'z')|('A'..'Z')|('0'..'9'))*;
NUMBER : ('0'..'9')('0'..'9')*;
WS : [ \r\n\t]+ -> skip;
```