Examination in Compilers, EDAN65

Department of Computer Science, Lund University

2023–10–27, 14.00-19.00

SOLUTIONS

Max points: 60
For grade 3: Min 30
For grade 4: Min 40
For grade 5: Min 50

1 Lexical analysis

a) (5p)

Strings for `F`:

```
bc
abc
aabc
aaabc
```

Strings for `G`:

```
ac
abbc
abbbc
```

We can note that the string `abc` is an `F` token rather than a `G` token, because of rule priority.

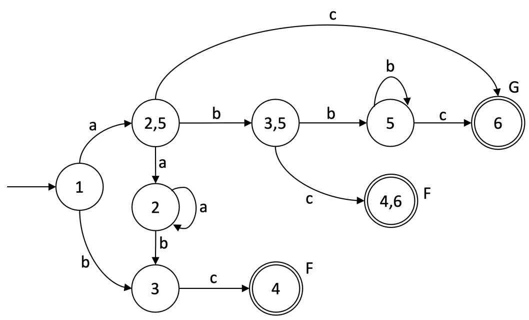
b) (5p)

Regular expressions for `F` and `G`:

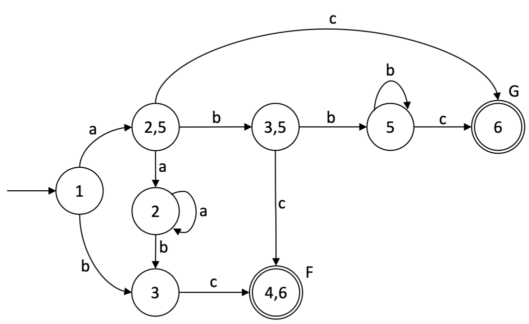
```
F = a*bc
G = ab*c
```

c) (5p)

The DFA:



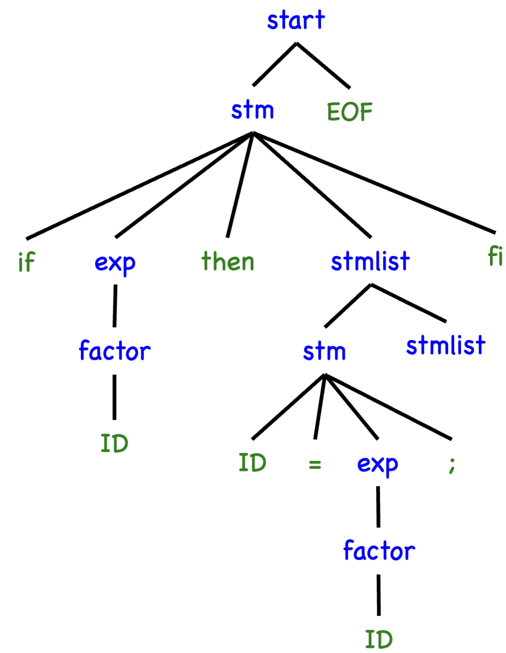
It is possible to minimize this DFA further by joining the states $\{4\}$ and $\{4, 6\}$, resulting in the following DFA:



2 Grammars

a) (5p)

Derivation tree:



b) (5p)

LR parsing sequence with `stack • input ; action` in each step

• "if" ID "then" ID "=" ID ";" "fi" EOF	; SHIFT
"if" • ID "then" ID "=" ID ";" "fi" EOF	; SHIFT
"if" ID • "then" ID "=" ID ";" "fi" EOF	; REDUCE p_8 (factor → ID)
"if" factor • "then" ID "=" ID ";" "fi" EOF	; REDUCE p_7 (exp → factor)
"if" exp • "then" ID "=" ID ";" "fi" EOF	; SHIFT
"if" exp "then" • ID "=" ID ";" "fi" EOF	; SHIFT
"if" exp "then" ID • "=" ID ";" "fi" EOF	; SHIFT
"if" exp "then" ID "=" • ID ";" "fi" EOF	; SHIFT
"if" exp "then" ID "=" ID • ";" "fi" EOF	; REDUCE p_8 (factor → ID)
"if" exp "then" ID "=" factor • ";" "fi" EOF	; REDUCE p_7 (exp → factor)
"if" exp "then" ID "=" exp • ";" "fi" EOF	; SHIFT
"if" exp "then" ID "=" exp ";" • "fi" EOF	; REDUCE p_3 (stm → ID "=" exp ";")
"if" exp "then" stm • "fi" EOF	; REDUCE p_5 (stmlist → ϵ)
"if" exp "then" stm stmlist • "fi" EOF	; REDUCE p_4 (stmlist → stm stmlist)
"if" exp "then" stmlist • "fi" EOF	; SHIFT
"if" exp "then" stmlist "fi" • EOF	; REDUCE p_1 (stm → "if" exp "then" stmlist "fi")
stm • EOF	; ACCEPT

c) (5p)

Equivalent LL(1) grammar:

p_0 : `start → stm EOF`
p_1 : `stm → "if" exp "then" stmlist optelse "fi"`
p_2 : `optelse → "else" stmlist`
p_3 : `optelse →` ϵ
p_4 : `stm → ID "=" exp ";"`
p_5 : `stmlist → stm stmlist`
p_6 : `stmlist →` ϵ
p_7 : `exp → factor expprime`
p_8 : `expprime → "+" exp`
p_9 : `expprime →` ϵ
p_{10} : `factor → ID`
p_{11} : `factor → "(" exp ")"`

d) (5p)

The LL(1) table:

	EOF	"if"	"then"	"fi"	"else"	ID	"="	";"	"+"	"("	")"
start		p0				p0					
stm		p1				p4					
optelse				p3	p2						
stmlist		p5		p6	p6	p5					
exp						p7				p7	
expprime			p9					p9	p8		p9
factor						p10				p11	

Since there is no conflict, the grammar is LL(1).

3 Program analysis

a) (3p)

The abstract grammar:

```
Program ::= Statement*;
abstract Statement;
Decl : Statement ::= IdDecl;
Assignment : Statement ::= IdUse Expr;
FuncCallStmt : Statement ::= FuncCall;
abstract Expr;
IntLit : Expr ::= <INT>;
Add : Expr ::= Left:Expr Right:Expr;
IdUseExpr : Expr ::= IdUse;
FuncCall : Expr ::= IdUse Arg:Expr*;
```

b) (4p)

```
coll Set<IdUse> IdDecl.uses() [new HashSet<IdUse>()]
  with add root Program;

IdUse contributes this
  to IdDecl.uses()
  for decl();
```

c) (4p)

```
inh boolean IdUse.valueAccessed();
eq  Assignment.getIdUse().valueAccessed() = false;
eq  Program.getChild().valueAccessed() = true;
```

d) (4p)

```
syn boolean IdDecl.dead() {
  for (IdUse u : uses()) {
    if (u.valueAccessed()) return false;
  }
  return true;
}
```

4 Code generation and run-time systems

a) (5p)

```
d:
# Set up frame
push rbp            # push dynamic link
mov rsp rbp         # set new base pointer
sub 8 rsp           # allocate room for local vars

mov 32(rbp) -8(rbp) # z -> r

mov 16(rbp) rax     # compute second arg to b (x+1)
inc rax
push rax            # push second arg to b

push -8(rbp)        # push second arg to a (r)

push 24(rbp)        # push arg to c (y)
call c
add 8 rsp           # pop arg to c

push rax            # push first arg to a (result of c(...))
call a
add 16 rsp          # pop both args to a

push rax            # push first arg to b (result of a(...))
call b
add 16 rsp          # pop both args to b

mov rax -8(rbp)     # rax -> r

mov -8(rbp), rax    # Compute return value (r+1) and place in rax
inc rax

# Take down frame and return
mov rbp rsp         # deallocate local vars
pop rbp             # reset base pointer to caller frame
ret




Address table:
------------------
x   16(rbp)
y   24(rbp)
z   32(rbp)
r   -8(rbp)
```

b) (5p)

Stack at ** PC **: