KØBENHAVNS UNIVERSITET

Implementering af programmeringssprog - Skriftlig 4 timer

12

23 august 2023

Planlagt: 09:00 - 13:00

Eksamensnr: 12

Plads: EH-0136

Side 1 af 11

1.1
Note: Due to time constrains the set parentheses {} are omitted, but should read as e.g. ec({1, 2, 3}) = (1, 2, 3…).

Alphabet = {f, t}

ec(1, 2) = (1, 2) =: s0 REJ

move(s0, f) = ec(3) = (3) =: s1 REJ

move(s0, t) = ec(3, 4) = (3, 4, 1, 2) =: s2 ACC

move(s1, f) = ec(2) = (2) =: s3 REJ

move(s1, t) = ec(4) = (4, 1, 2) =: s4 ACC

move(s2, f) = ec(2, 3) = (2, 3) =: s5 REJ

move(s2, t) = ec(4,3) = s2

move(s3, f) = ec() undefined

move(s3, t) = ec(3, 4) = s2

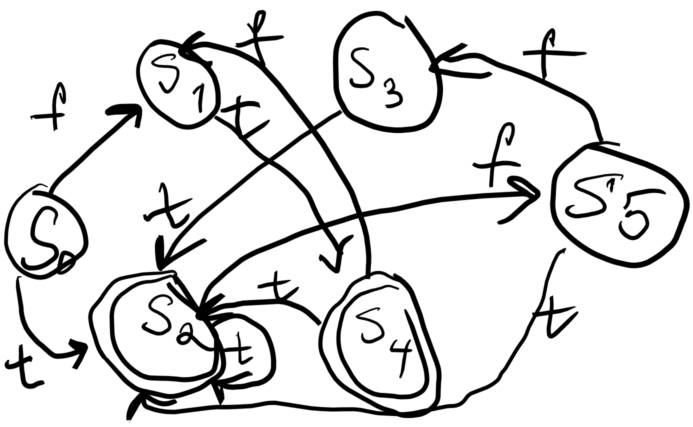
move(s4, f) = ec(3) = s1

move(s4, t) = ec(3,4) = s2

move(s5, f) = ec(2) = s3

move(s5, t) = ec(3, 4) = s2

s´ = {s0, s1, **s2,** s3, **s4**, s5}



1.2

Preprocessing: add new, rejecting state 7 with:

- c transition from 0
- c transition from 1
- c transition from 3
- b, c transitions from 5
- b transition from 6
- a, b, c transitions to itself

G1 ≔ {1, 4}

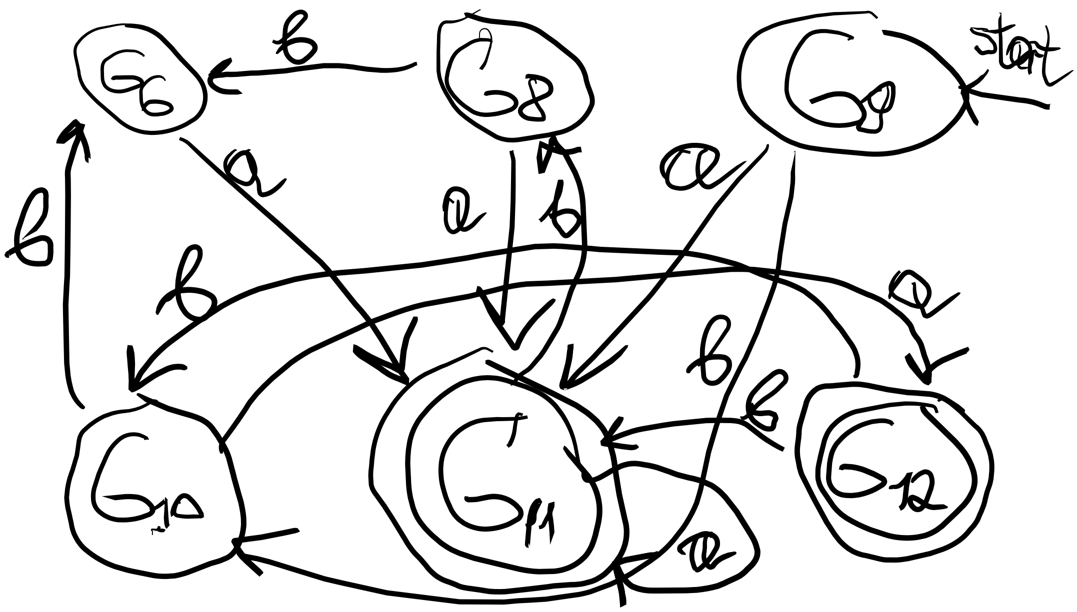
G2 ≔ {0, 2, 3, 5, 6, 7}

G2 a b c 0 G1 G2 G2 => G3 2 G1 G2 G2 => G3 3 G1 G2 G2=> G3 5 G1 G2 G2=> G3 6 G2 G2 G2=> G4 7 G2 G2 G2=> G4 Split G2 into G3 = {0, 2, 3, 5} and G4 = {6, 7} G3 a b c 0 G1 G3 G4 => G5 2 G1 G3 G4=> G5 3 G1 G3 G4=> G5 5 G1 G4 G4=> G6 Split G3 into G5 = {0, 2, 3} and **G6 = {5}** =>	G5 a b c 0 G1 G5 G4 => G7 2 G1 G6 G4=> G7 3 G1 G6 G4=> G8 Split G5 into G7 ={0,2} and **G8 = {3}** G7 a b c 0 G1 G7 G4 => G 9 2 G1 G6 G4 => G10 split G7 into **G9 ={0}** and **G10={2}** G1 a b c 1 G1 G8 G4 => G11 4 G1 G10 G4 => G12 Split G1 into **G11 = {1}** and **G12 ={4}**

G4 a b c 6 G4 G4 G4 7 G4 G4 G4 **G6 a b c** 5 G11 G4 G4 **G8 a b c** 3 G11 G6 G4 =>	**G9 a b c** 0 G11 G10 G4 **G10 a b c** 2 G12 G6 G4 **G11 a b c** 1 G11 G8 G4 **G12 a b c** 4 G11 G10 G4

Postprocessing: G4 contains the added dead state 7, and can be removed together with transitions to it. The final transition table is:

	a.	b.	c	
G6	G11	-	-	REJ
G8	G11	G6	-	REJ
G9	G11	G10	-	REJ, START
G10	G12	G6	-	REJ
G11	G11	G8	-	ACC
G12	G11	G10	-	ACC



2.1

a. Yes. s -> 1 X -> 2 X + Y -> 2 X + Y + Y -> 3 Y + Y + Y -> 5 + Y + Y -> 5 ++ Y -> 5 ++
b. Yes. s -> 1 X -> 3 Y ->4 Y b -- -> 4 Y b -- b-- -> 5 b--b—
c. This G generates strings of the alphabet {b, +, -} and describes a palindrome language, e.g. b--+b--. This language requires a counter and is too complex to be handled by regular expressions, a NFA or a DFA, whose scope is limited due to computer's limited memory. That is, this L(G) is irregular.
d. To eliminate left recursion, we introduce new non-terminals for production X -> X + Y (X´) and Y -> Yb (Y´).

```
S -> X
X -> Y X´
X´ -> + Y X´
X´ ->
Y -> b -- Y´
Y´ -> b -- Y´
Y´ ->
```

2.2

a) Compute nullable(N) for all N:

null(S) = false [always because of $]

null(X) = null(A) && null(B) = ? && ? = true && ? = true && true = true

null(A) = null(a) || null(eps) = false || true = true

null(B) = null(bAB) || null(eps) = false || true = true

b) Compute first(N) for all N:

first(S) = first (X) U first($) = {$} [since =null(X)]

first(X) = first (A) U first(B) [since =null(A), null(B)]

first(A) = first(a) U first(eps) = {a} [since !=null(a)] first(B) = first(bAB) U first(eps) = first(b) U {} = {b} [since !=null(b)]

simplifying
first(X) = {a, b}

c) Write constraints on follow sets:

0. S → X $
    1. {$} <= follow(X) [because first($) = {$}]

1. X → A B
    2. {b} <= follow(A) [because first(B) = {b}]

    3. follow(X) <= follow(B) [because B last in production]

2. A → a
3.A → ϵ
4.B → b A B
    4. {b} <= follow(A) [because first(B) = {b}]
    5. follow(B) <= follow(B) [trivial]

5.B → ϵ

d) Find least solution:

Seed rules {a1,…,an} <= follow(N) : 1., 2., 4.

Propagation rules follow(N1) <= follow(N2) : 3., 5.

set	seed	prop1	final
follow(X)	$ [1.]		$
follow(A)	b [2.] , [4.]		b
follow(B)		$ [3.]	$

e) Compute look-ahead sets:

la (S → X$) = first(X$) = first (X) U {$} = {a, b, $}

la (X → AB) = first(A) U follow(X) = {a, $}

la (A → a) = first(a) = {a} OK: disjoint

la (A→) = first(eps) U follow(A) = {b}

la (B → bAB) = first(bAB) = {b}

la (B →) = first(eps) U follow(B) = {$}　　　OK: disjoint

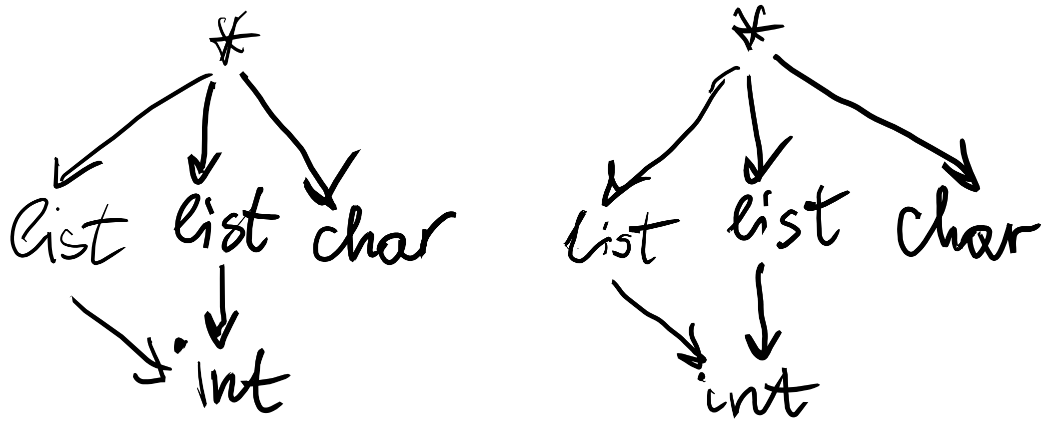
f. Write a recursive-descent parser for non-terminals S and B:

```
function parseS () =
        if input = 'a' or input = 'b' or input = '$' then
                parseX() ; match('$')
        else reportError()
function parseB() =
        if input = '$' then
                (* do nothing, just return *)
        else if input = 'b' then
                match('b') ; parseA(); parseB()
        else reportError()
```

3.2

a.
-1 Apply rule (IV): both root nodes are the same type constructor, unify hteir children.
-2 unify(list(alpha), beta); apply rule III : union (list(alpha), beta) => beta = list(alpha)
-3 unify(alpha, list(int)); apply rule III: union(alpha, list(int)) => alpha = list(int)
-4 unify(char, gamma); apply rule III: union(char, gamma) => gamma = char

Modify 2 after 3: beta = list(list(int))



b. alpha = list(int); beta = list(list(int)); gamma = char.

c. list(list(int)) * list(int) * char

4.

```
t0 =: 1
v1 =: t0

LABEL lab1                  //start repeat-until loop
t1 := v1
t1 := t1 * 4
t1 := t1 + v3
t2 := M[t1]
t3 =: 10
IF t2 > t3 THEN lab2 ELSE lab3          //lab2 if FALSE

LABEL lab2
t5 := 0                                 //Cond returns false

LABEL lab3
t5 := 1                                 //Cond returns true
t7 := v1
v1 := CALL f100(t7, t5)

t8 := v1
t8 := t8 *4
```

```
t8 := t8 + v3
t9 := 5
M[8] := t9

t10 := v1
t11 := CALL f200(t10)

IF t11= 0 THEN lab1 ELSE lab4

LABEL lab4                          // after repeat-until loop
```

4.2

```
  lw     y, 51(x)
  slt    R1, y, z
  bne    R1, R0, lab1
  j      lab2
lab3:
   slt    R1, z, y
   beq    R1, R0, lab5
lab4:
```

5.1

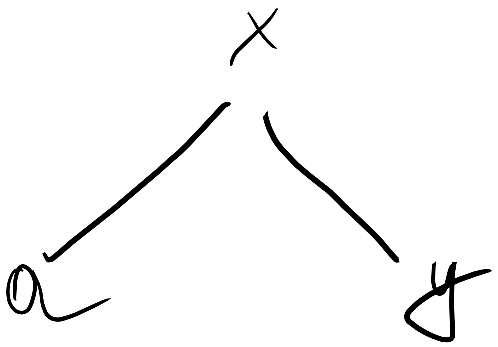
a.

A) In {a, b, d} =? gen {} U (out {a, c, d} \ kill {c}) = {a, d}
B) In {a, b, d} =? gen {} U (out {a, c, d} \ kill {}) = {a, c, d}
C) In {a, b, d} =? gen {c, b} U (out {a, c, d} \ kill {}) = {a, b, c, d}
D) In {a, b, d} =? gen {b, c} U (out {a, c, d} \ kill {c}) = {a, d, b, c}
E) None of the above: CORRECT

b. A) In {b, d} =? gen {b, d} U (out {a} \ kill {a}) = {b, d} CORRECT
B) In {b, d} =? gen {a} U (out {a} \ kill {}) = {a}
C) In {b, d} =? gen {b, d, e} U (out {a} \ kill {a}) = {b, d, e}
D) In {b, d} =? gen {b, d} U (out {a} \ kill {}) = {a, b, d}

5.2
a and b.

i	succ[i]	gen[i]	kill[i]	1. out	1. in	2.out	2.in
1	2	x	x	y, x	y, x	y, x	y, x
2	3			y, x	y, x	y, x	y, x
3	4	y, x	a	x, a	y, x	x, a	y, x
4	5	a	y	x, a	x, a	y, x, a	x, a
5	6	x, a	x	x	x, a	y, x	y, x, a
6	2, 7	x		x	x	y, x	y, x
7	8			x	x	x	x
8		x			x		x



c and d.

i	kill	out	interferes
1	x	y, x	y
3	a	x, a	x
4	y	y, x, a	x
5	x	y, x	y

NB: 4: y:= a, thus y does not interfere with a

e.
Nodes a and y have < 2 neighbors, so we can start with either of them.

Node	neighbors	color
x		1
y	x	2
a	x	2