Lösningsförslag

Översättarteknik, 5p
13 januari 1995

Jag har inte hunnit titta igenom dessa lösningar så noga, men förhoppningsvis ska det väl inte vara så mycket fel.

Uppgift 1 (3 + 2 p)

(a) Lite kortfattat kan man väl t ex ta dessa faser:

- Lexikalanalys: läser tecknen i källkoden och grupperar dessa till en ström av tokens. Associerar också attribut till vissa tokens.
- Mellankodsgenerering: genererar mellankod, t ex 3-adresskod, utgående från den information som vidarebeordrats från de föregående analysfaserna.
- Kodoptimering: går igenom mellankoden och försöker förbättra den kod som genererats tidigare så att man får en effektivare kod.

(b) Ett pass: läsa fil — jobba — skriva fil. Ett pass består vanligen av flera, ofta sammanflätade, faser.
Enpasskompilator: Fördel, slipper läsa och skriva filer flera gånger — tjänar tid. Nackdel, kan behöva hålla hela programmet i minnet — tar upp mycket minne.

Uppgift 2 (3 + 3 p)

(a) E → E_1 **nor** M E_2
{ backpatch(E_1.falselist, M.quad)
E.truelist := E_2.falselist
E.falselist := merge(E_1.truelist, E_2.truelist) }

(b) S → **repeat** M_1 S_1 **until** M_2 E
{ backpatch(E.falselist, M_1.quad)
backpatch(S_1.nextlist, M_2.quad)
S.nextlist := E.truelist }

M → ϵ { M.quad := nextquad }

Uppgift 3 (5 p)

U → V ∘ U
U → V
V → Y × V
V → Y
Y → **id**

Uppgift 4 (2 + 2 + 2 + 2 p)

(a) Eliminera vänsterrekursion:
S → (L)
S → **a**
L → S Q
Q → , S Q
Q → ϵ

(b) FIRST(A) = { (**a** }
FIRST(L) = { (**a** }
FIRST(Q) = { , ϵ }
FOLLOW(A) = { **$** ,) }
FOLLOW(L) = {) }
FOLLOW(Q) = {) }

(c)

	(	)	**a**	,	**$**
S	S → (L)		S → **a**		
L	L → S Q		L → S Q		
Q		Q → ϵ		Q → , S Q	

(d)

Stack	Input	Production
$ S	((**a**)) **$**	S → (L)
$) L (	((**a**)) **$**	match
$) L	(**a**)) **$**	L → S Q
$) Q S	(**a**)) **$**	S → (L)
$) Q) L (	(**a**)) **$**	match
$) Q) L	**a**)) **$**	L → S Q
$) Q) Q S	**a**)) **$**	S → **a**
$) Q) Q **a**	**a**)) **$**	match
$) Q) Q	)) **$**	Q → ϵ
$) Q)	)) **$**	match
$) Q	) **$**	Q → ϵ
$)	) **$**	match
$	**$**	

accept

Uppgift 5 (6 + 2 p)

(a) (0) $Z \rightarrow S$
(1) $S \rightarrow (L)$
(2) $S \rightarrow \mathbf{a}$
(3) $L \rightarrow L , S$
(4) $L \rightarrow S$

FOLLOW(Z) = { **$** }
FOLLOW(S) = { **$**) , }
FOLLOW(L) = {) , }

0:
$Z \rightarrow . S$
$S \rightarrow . (L)$
$S \rightarrow . \mathbf{a}$

goto(0,S) = 1:
$Z \rightarrow S .$

goto(0,() = 2:
$S \rightarrow (. L)$
$L \rightarrow . L , S$
$L \rightarrow . S$
$S \rightarrow . (L)$
$S \rightarrow . \mathbf{a}$

goto(0,**a**) = 3:
$S \rightarrow \mathbf{a} .$

goto(2,L) = 4:
$S \rightarrow (L .)$
$L \rightarrow L . , S$

goto(2,S) = 5:
$L \rightarrow S .$

goto(2,() = 2
goto(2,**a**) = 3
goto(4,)) = 6:
$S \rightarrow (L) .$

goto(4,,) = 7:
$L \rightarrow L , . S$
$S \rightarrow . (L)$
$S \rightarrow . \mathbf{a}$

goto(7,S) = 8:
$L \rightarrow L , S .$

goto(7,() = 2
goto(7,**a**) = 3

Tillstånd	(	)	a	,	$	S	L
0	s2		s3			1	
1					acc		
2	s2		s3			5	4
3		r2		r2	r2		
4		s6		s7			
5		r4		r4			
6		r1		r1	r1		
7	s2		s3			8	
8		r3		r3			

(b)

Stack	Input	Production
$ 0	(**a** , (**a**)) $	
$ 0 (2	**a** , (**a**)) $	
$ 0 (2 **a** 3	, (**a**)) $	S → **a**
$ 0 (2 S 5	, (**a**)) $	L → S
$ 0 (2 L 4	, (**a**)) $	
$ 0 (2 L 4 , 7	(**a**)) $	
$ 0 (2 L 4 , 7 (2	**a**)) $	
$ 0 (2 L 4 , 7 (2 **a** 3	)) $	S → **a**
$ 0 (2 L 4 , 7 (2 S 5	)) $	L → S
$ 0 (2 L 4 , 7 (2 L 4	)) $	
$ 0 (2 L 4 , 7 (2 L 4) 6	) $	S → (L)
$ 0 (2 L 4 , 7 S 8	) $	L → L , S
$ 0 (2 L 4	) $	
$ 0 (2 L 4) 6	$	S → (L)
$ 0 S 1	$	
accept		

Uppgift 6 (2 + 2 p)

(a)
- Måste bevara programmets mening. Den optimerade koden måste väl göra samma sak som originalet annars kan man ju lika gärna optimera bort all kod.
- Ska ge ett bättre program. Åtminstone allt som oftast. Om det oftast ger ett sämre program, varför ska man då lägga ner resurser/tid på att försöka optimera?

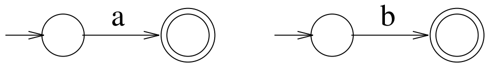
(b)
- Kan försvåra deugging — kan vara svårt att få referenser till källkoden.
- Omfattande optimeringar kräver ganska mycket resurser (endast vid kompilering, men ändå).

Uppgift 7

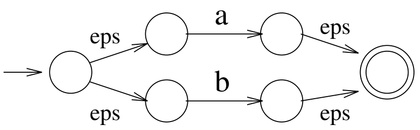
(a)

Maskiner för:

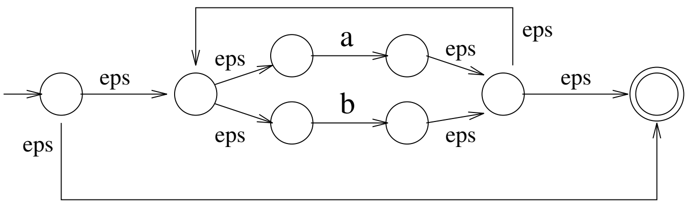
a, b



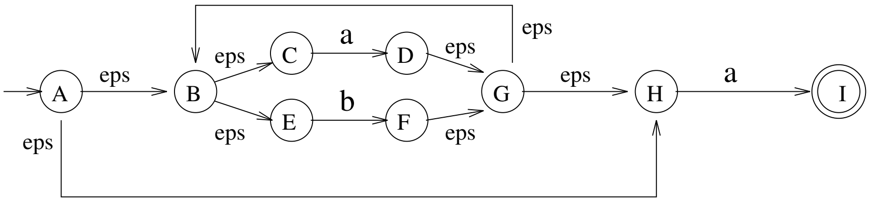
a | b



(a | b)*

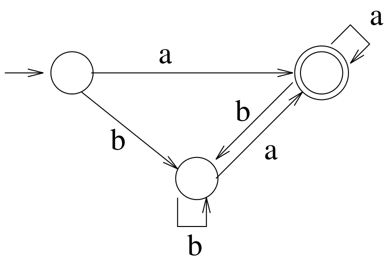


(a | b)*a



(b)

	a	b
ABCEH	BCDEGHI	BCEFGH
BCDEGHI	BCDEGHI	BCEFGH
BCEFGH	BCDEGHI	BCEFGH



(c)