Naam: André Wolters
Adres: (0967734)
Postcode en Woonplaats:

Geb. datum: 20-11-73
Studierichting: Technische Informatica
Jaar van eerste inschrijving: '96

Bladnr.: 1
Tentamen: Vertalerbouw 1
Datum: 16-03-98
Naam docent: J. Jongejan

1

a) Als eerste heb ik de regel $Z \to S \perp$ toegevoegd.

First(S) = {a, (} Follow(S) = {⊥,)}

$Z \to S \perp$ ← toegevoegd

↑ follow op S.

First(E) = {a, (} Follow(E) = {⇐, ⇒, ⊥,)}

First(F) = {a, (} Follow(F) = {⇐, ⇒, ⊥,)}

↑ (slechtheidje (zit in followset)

b) ~~Conflicten~~

LL(1) valt er af omdat er linkerrecursie aanwezig is $E \to E \Leftarrow F$ en $E \to E \Rightarrow F$

Nu misschien LR(0)?

Eerst

$Z \to .S \perp$ toevoegen geeft

↓ nieuw toestand (nummer)

(1) $Z \to .S \perp$ (2)
$S \to .E$ (3)
$E \to .E \Leftarrow F$ (3)
$E \to .E \Rightarrow F$ (3)
$E \to .F$ (4)
$F \to .i$ (5)
$F \to .(S)$ (6)

(2) $Z \to S . \perp$

(3) $S \to E .$ ⇐/⇒/⊥ 2x ⇐/⇒ conflict dus ~~geen~~ LR(0)
$E \to E . \Leftarrow F$ (3) conflict 2x ? ~~[illegible]~~
$E \to E . \Rightarrow F$ (3) Misschien hoe SLR(1)?

[Intermezzo] →

kijken dan of we alle R/R en R/S conflicten kunnen oplossen. ~~[illegible]~~ Wanneer dit mogelijk is, dan is de grammatica SLR(1).
Indien niet mogelijk dan kijken naar LALR(1) en LR(1)

↓ Vervolg

(4) $E \to F.$

(5) $F \to a.$

(6) $F \to (.S)$ (9)
$S \to .E$ (3)
$E \to .E \in F$ (3)
$E \to .E \Rightarrow F$ (3)
$E \to .F$ (4)
$F \to .a$ (5)
$F \to .(S)$ (6)

(7) $E \to E \in .F$ (10)
$F \to .a$ (5)
$F \to .(S)$ (6)

(8) $E \to E \Rightarrow .F$ (11)
$F \to .a$ (5)
$F \to .(S)$ (6)

(9) $F \to (S.)$ (12)

(10) $E \to E \in F.$

(11) $E \to E \Rightarrow F.$

(12) $F \to (S).$

In toestand (3) zijn 2x R/S conflicten

$S \to E.$
$E \to E. \in F$
} $\in \notin$ Follow(S) ↳ zie vraag a ?

$S \to E.$
$E \to E. \Rightarrow F$
} $\Rightarrow \notin$ Follow S ?

In beide gevallen is het R/S probleem opgelost dus de grammatica is SLR(1) ?

Woonplaats: | Jaar van eerste inschrijving: | Naam docent:

c)

F → (S) wordt F → (E)

Dan heb je in toestand 6 (vraag b) de volgende situatie.

6 F → (.E) 9 ⟶ 9 F → (E.)

E → .E <= F ⑦
E → .E => F ⑧
E → .F
F → .a
F → .(E)

E → E. <= F
E → E. => F

wordt bij toestand 9

Er gebeurt niks de grammatica blijft SLR(1) & creëert geen conflicten op deze manier.

2 De volgende attributen worden gebruikt

id } inherited voor alle nonterminals
const }

Vat = synthetic attr. voor → P, E, E', T, T' en F

Verder worden er de volgende action symbols gebruikt

- <add> inherited V1 en V2 synthetic Result
- <sub> inherited V1 en V2 synthetic Result
- <mult> inherited V1 en V2 Synthetic Result
- <check> inherited id uniek synthetic Result

P : E (id = const)

P.val = E.val
E.id = id.Name
E.const = const.Val

E : T E' <add>

(kom negatief gelet zijn)

E.val = add.Result
T.id = E.id
T.const = E.const
E'.id = E.id
E'.const = E.const

add.v1 = T.val
add.v2 = E'.val

E' : + T E' <add>

E'.val = add.Result
T.id = E'.id
T.const = E'.const
E'.id = E'.id
E'.const = E'.const

add.v1 = T.val
add.v2 = E'.val

E' : - T E' <sub>

E'.val = sub.Result
T.id = E'.id
T.const = E'.const
E'.id = E'.id
E'.const = E'.const

sub.v1 = T.val
sub.v2 = E'.val

E' :

E'.val = 0 // optellen of aftrekken

T : F T' mul

T.val = mul.Result
mul.v1 = F.val
mul.v2 = T'.val
F.id = T.id
F.const = T.const
T'.id = T.id
T'.const = T.const

Postcode en Woonplaats: | Jaar van eerste inschrijving: | Datum: Naam docent:

2

T' : * F T' <mul>   T'.val = mul.Result
mul.v_1 = F.val
mul.v_2 = T'.val
F.id = T'.id
F.const = T'.const
T'.id = T'.id
T'.const = T'.const

T' :   T'.val = 1 // (vermenigvuldigen)

F : id <check>   F.val = check.Result
check.id.uniek [illegible] = F.id
check.const = F.const

F : const   F.val = const

Ik neem aan dat dit een getal is! dus geen check!
bv (2 * x (x=3)
id — check nodig!

Nu volgen de action symbols!

<add> = [illegible] Result = $v_1 + v_2$

X <sub> = Result = ~~$v_1 - v_2$~~ $v_2 - v_1$!

<mul> = Result = $v_1 * v_2$

<check> : if (id.uniek = id) Then Result = const
else ERROR

check is de belangrijkste action symbol. deze checkt de id en levert de waarde hiervan op (Indien de id juist is). Verder wordt er in de boom naar boven toe de waarden uitgerekend! en doorgegeven tot aan de wortel P!

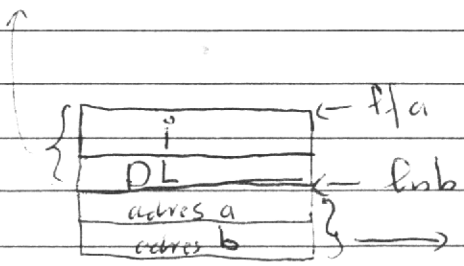
~~Het~~ De belangrijkste ~~van deze~~ attributen zijn de inherited attributen die naar beneden worden doorgegeven om helemaal beneden bij de bladeren een check te kunnen uitvoeren.
Wanneer er geen error optreedt dan worden de waarden recht toe recht aan uitgerekend en naar boven doorgegeven, de synthetic attributen. Het eindresultaat wordt in P.val opgeleverd!

3

a. add 1) als eerste P.PN = 1 dus geen static link! alleen een dynamic link! = (DL)

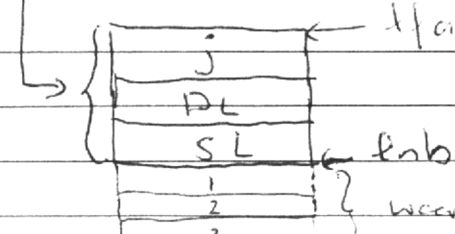
2) 1 variabele integer i

dit geeft AR van add



→ deze adressen staan onder de AR van add. dit zijn de adressen van de VAR parameters die add heeft meegekregen. Deze plaatsen zijn ~~bij~~ na aanroep bekend!

AR van extra : P.PN = 2 dus wel een static link ook een dynamic link



} waarden van b i.p.v. een var wordt de hele array meegegeven.

Deze waarden krijgt … b is namelijk van het type vector: array

Postcode en Woonplaats:

Studierichting: TI

Jaar van eerste inschrijving:

Tentamen: Oef toets boom

Datum: 16-03-98

Naam docent:

// = commentaar

b

entry extra M[lfa] = env // statie link

M[lfa+1] = lnb // dynamic link

lnb = lfa

lfa = lfa + 2 + 1 // DL + SL

exit extra = lfa = lnb

lnb = M[lfa+1]

return

c

* 1 * $R_0 = M[lnb+2]$ // waarde van j

$R_1 = M[lnb - R_0]$ // waarde van b[j]

deze staat direct onder de AR van extra!

$R_2 = M[lnb]$ // adres van add is statie link!

$R_3 = M[R_2]$ // adres van programma tentamen is dynamic link naar tentamen.

X ingeval add recursieve calls bevat helpt dit niet!

v is global, dus te adresseren bv. GP

deze heeft namelijk add aangeroepen

$R_4 = M[R_3]$ // adres van array V.

Lowerbound correction V $R_5 = R_4 + R_0^{-1}$ // adres + waarde j

$R_6 = M[R_5]$ // waarde op deze gebergenplaats = V[j]

$R_7 = R_6 + R_1$ // V[j] + b[j]

$R_8 = M[R_2 - 1]$ // adres van array a

idem V $R_9 = R_8 + R_0^{-1}$ // adres + waarde j

$M[R_9] = R_7$ // waarde van V[j] + b[j] wordt op deze geheugenlokatie geplaatst

* 2 *

```
R0 = M[fnb+1] : // waarde van i
~~R1 = M[fnb] adres~~
R1 = M[fnb-1]   // adres van array a
R2 = M[fnb-2]   // adres van array b
R3 = R1 + R0-1  // adres a+i
R4 = M[R3]      // waarde op adres(a+i)
R5 = R2 + R0-1  // adres b+i
R6 = M[R5]      // waarde op adres(b+i)
M[R3] = R4 + R6 // waarde opgeslagen
                   in locatie(a+i)
```

* 3 *

```
R0 = M[fnb-2] : // adres van array b
R1 = M[R0]    : // ~~eerste~~ element b[1]
R2 = M[R0+1]  // element b[2]
R3 = M[R0+2]  // element b[3]
M[lfa] = R3   // } elementen
M[lfa+1] = R2 // } worden boven de
M[lfa+2] = R1 // } AR van add
lfa := lfa+3  // } gezet.
~~code~~       // extra waarde
env = fnb     // } aangeroepen
call fab extra // } en de omgeving
lfa := lfa-3  // } wordt gezet
```

* 4 *

```
R0 = M[gp]  : // adres van V
              // gp = beginadres
              // van programma
              // tentamen
M[lfa] = R0   // } adres wordt boven
M[lfa+1] = R0 // } de AR geplaatst
              // } 2 maal in dit geval
lfa := lfa+2  // } het zelfde adres !
env = fnb     // zet de omgeving
call fab add  // roep add aan!
lfa := lfa-2
```

Jaar van eerste inschrijving: 96 | Naam docent:

a)

E → E ⇐ F : linkerrecursie wegwerken!

E → T Etail1
T → E indirect linksrecursief!
Etail1 → ⇐ F Etail1
Etail1 →

E → E => F idem op dezelfde manier

E → T Etail2
T → E idem
Etail2 → => F Etail2
Etail2 →

Nu wg dezelfde prefer wegwerken geen tijd meer voor!
↳ { E → T Etail1
  { E → T Etail2

b)

Program stacks.

Initstack;
Initscanner;
while (not-empty)
begin

case symbol ? (bovenste 'top' ?

: tpar, rpar, leftarrow, rightarrow, asym, eofs.
↓
while sym (invoer) = top (op de stack)
begin
pop // pop van de stack
nextsym // lees het nieuwe invoer
end // teken de volgende

: S
  push (E)

: E
  ~~push (E tail1) , push (T)~~
  if Sym = push (T) push (E tail 1)
  else
    push ?

: T
  push (F)

: E tail 1
  if Sym ∈ { a, (} then
    push (E tail 1) , push (F)
  else *skip*

: E tail 2
  if Sym ∈ { a, (} then
    push (E tail 2), push (F)
  else *skip*

: F
  if Sym ∈ = { a then
    push a
  else push (S)

in what is de error recovery ?