$S \to L = R \mid R$

$L \to *R \mid id$

$R \to L$

FIRST	
S	*, id
L	*, id
R	*, id

FOLLOW	
S	$
L	$, =
R	$, =

SLR(1) states (sets of LR(0) items)

S_0 = start state:
$S \to .L=R$
$S \to .R$
$L \to .*R$
$L \to .id$
$R \to .L$

S_1 = trans(S_0, L): $S \to L.=R$, $R \to L.$

S_2 = trans(S_0, R): $S \to R.$

S_3 = trans(S_0, *): $L \to *.R$
$R \to .L$
$L \to .*R$
$L \to .id$

action[S_1, =] = "shift S_6"

S_6 : trans(S, =)
$S \to L=.R$
$R \to .L$
$L \to .*R$
$L \to .id$

action[S_2, =] = "reduce $R \to L$"
because '=' ∈ Follow(R)

shift-reduce conflict.

$\underbrace{*id}_{L} = id$
S_4

(Canonical) LR(1) parser:

state is a set of LR(1) items.

An LR(1) is a pair:

$[\underbrace{A \to \alpha . \beta}_{\text{LR(0) item}}, \underbrace{a}_{\text{token (or \$)}}]$

"core" — "lookahead"

Intuition:
If this item is in the current state, then that means that we could be working our way through $A \to \alpha\beta$ (as before) and we expect to see a as the lookahead when it is time to reduce.

If $[A \to \alpha., a]$ is in the current state, then only reduce if we see a on the lookahead.

start state (if $S' \to S$ for augmented grammar)

closure$[S' \to .S, \$]$

closure(S) S set of LR(1) items:

if $[A \to \alpha.B\beta, a] \in S$,
then add $[B \to .\gamma, b]$
for all B-productions $B \to \gamma$
and all $b \in FIRST(\beta a)$
(apply repeatedly until can't add anything more)

trans(S, X):
For each item in S of the form $[A \to \alpha.X\beta, a]$
put $[A \to \alpha X.\beta, a]$
into trans(S, X)
then take the closure

action[S, a]:
if $[A \to \alpha.a\beta, b] \in S$,
then set action[S, a] = "shift T"
where T = trans(S, a)
(same as before)

If $[A \to \alpha., b] \in S$,
then set
action[S, b] = "reduce $A \to \alpha$"

goto(S, A) := trans(S, A)
(same as before)

Apr 17-11:00 AM

$S \to L=R \mid R$
$L \to *R \mid id$
$R \to L$

S_0 = start state:
$S \to .L=R, \$$
$S \to .R, \$$
$L \to .*R, =$
$L \to .id, =$
$R \to .L, \$$ ← FIRST($)
$L \to .*R, \$$
$L \to .id, \$$

S_1 = trans(S_0, L):
$S \to L.=R, \$$
$R \to L., \$$

no conflict!
action[S_1, =] = "shift T"
T = trans(S_1, =)
action[S_1, $] = "reduce R→L"

S_2 = trans(S_0, R): $S \to R., \$$

S_3 = trans(S_0, *):
$L \to *.R, =$
$L \to *.R, \$$
$R \to .L, =$
$R \to .L, \$$
$L \to .*R, =$
$L \to .*R, \$$
$L \to .id, =$
$L \to .id, \$$

etc.

LALR(1) parser

Canonical LR(1) often produces too many states

Notice: cores of each LR(1) parser state are same as for SLR. (Indep of lookahead)

State of LALR(1) parser is a set of LR(1) items.

Easy description:
Start with an (canonical) LR(1) parser,
merge states with the same set of cores
union
goto & action tables constructed similarly

(Time & space inefficient)

efficient methods exist to construct an LALR(1) parser.

[no further details — not on the test]

Code generation & optimization

generic low-level target language: three-address code

L: $x := y$ op z
$x :=$ op y
goto L
if x cmp y then goto L

Apr 17-11:45 AM