FIRST(β) set of terminals and/or ε

β is any string of grammar symbols.

FOLLOW(A)

A is a nonterminal

set of all tokens (or $) that could come right after A in some sentential form

Assume from now on, that the start symbol does not appear in the body of any production.

Easy to convert any grammar into an equivalent one of this form:

G grammar with start symbol S.

G' is same grammar, except, new, fresh nonterminal S' is new start symbol, then add a single production

$S' \to S$

to G. [G' is an augmented grammar]

Given G' augmented grammar with start symbol S'

Construct FOLLOW sets as follows:

1. put $ into FOLLOW(S')
2. If $A \to \alpha B \gamma$ is a production, where A, B nonterminals, α, γ are strings of grammar symbols:
   a) But every token in FIRST(γ) into FOLLOW(B)
   b) If $\varepsilon \in$ FIRST(γ) [e.g. $A \to \alpha B$] put all of FOLLOW(A) into FOLLOW(B)

Start by with all FOLLOW sets empty, apply rules 1, 2a, 2b repeatedly until nothing else can get in.

$S \to E$

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to c \mid v \mid (E)$

A	FOLLOW(A)
S	$
E	$, +,)
T	$, +, *
F	$, +, *

Building an SLR(1) parser.

Items: An item (LR(0) item) is an expression of the form

$A \to \alpha . \beta$

where $A \to \alpha\beta$ is a production of the grammar.

Eg. $E \to E+T$ gives 4 possible items:

$E \to .E+T$

$E \to E.+T$

$E \to E+.T$

$E \to E+T.$

$A \to \varepsilon$ yields $A \to .$ as only item

A state of an SLR parser is a set of items. We build states starting the start state and following transitions.

Closure: Let S be a set of items. Compute the closure of S:

Closure(S):

T := S

repeat

let $A \to \alpha . B\gamma$ be some item in T, where A, B nonterminals, α, γ are strings of grammar symbols

For each production with head B

$B \to \delta$,

add $B \to .\delta$ to T.

until nothing new gets added

return T

Now we define the transition function trans(S, x) S is a state x is a grammar symbol

Apr 5-11:02 AM

trans(s, X) = t (state)

where:

t := ∅

repeat

if A → α . X β is in s,

where A nonterm,

α, β strings of grammar symbols,

X is a grammar symbol

add A → α X . β to t

// A → α . X β

until all such items are added to t

return closure(t)

Start state:

s_0 = closure({S' → . S})

where S' is the start symbol of the augmented grammar

s_0 = start state:

S → . E] kernel item

E → . E + T] non-kernel items

E → . T

T → . T * F

T → . F

F → . c

F → . (E)

S → E

E → E + T | T

T → T * F | F

F → c | (E)

nonkernel items are those added by closure op.

s_1 = trans(s_0, E):

S → E .

E → E . + T

s_2 = trans(s_0, T):

E → T .

T → T . * F

s_3 = trans(s_0, F)

T → F .

s_4 = trans(s_0, c)

F → c .

s_5 = trans(s_0, '(')

F → (. E)

E → . E + T

E → . T

T → . T * F

T → . F

F → . c

F → . (E)

s_6 = trans(s_1, +):

E → E + . T

T → . T * F

T → . F

F → . c

F → . (E)

Etc.

Construct the goto and action tables for the SLR(1) parser.

For every nonterminal A and state S, define

goto[S, A] := t = trans(S, A)

For every state S containing the item S' → S. (exactly one such state)

set action[t, $] := "accept"

For every state S containing an item of the form A → α . b β

where A nonterminal

b is a token

α, β strings

set action[S, b] = "shift t",

where t = trans(S, b)

[t contains A → α b . β]

For every state S containing

A → α . ,

set action[S, a] = "reduce A → α"

for each token a ∈ FOLLOW(A).

Apr 5-11:40 AM