LR(1) Table Construction

- **for all** input symbols ($V_t \cup \{ \vdash \}$)
  for all possible stack configurations
  Compute action(parseStackConfiguration, inputSymbol)
  Compute nextState(parseStackConfiguration, inputSymbol)
- The key problem in LR parsing is recognizing the handle at the top of the parse stack.
- Use *parser state numbers* to encode stack configurations so decisions can be made based on state number instead of a pattern match on the stack.
- There are a finite number of possible stack configurations for a finite grammar.
- LR(1) tables for real grammars are very large due to the large number of possible stack configurations and lookaheads
  e.g. > 1240 states and > 10,000 table entries for for Pascal.
- For LR(k) iterate over $V_t^k \cup \{ \vdash \}^k$

How to Construct LR(1) Tables

- Compute configuration sets
- Compute lookaheads for each state
- Based on state and lookahead, determine action and next state

Easier Approach

- Compute configuration sets
- Based on state, determine action and next state
- No lookahead is used to determine action - LR(0)
- Order: LR(0), LR(1), SLR, LALR

LR(0) Table Construction

- Each *parser state* in an LR(0) parser is associated with a unique *configuration set* of configuration items (partially completed phrases).
- The configuration items represent what has been seen *prior* to entering the state (i.e. α).
- A configuration item with the marker $\bullet$ at the right end. e.g.
  $G \to \alpha\, \beta\, \gamma\, \bullet$
  is a *REDUCE* production, $\alpha\, \beta\, \gamma$ is the handle, reduce to G
- To generate an LR(0) table start with a configuration set that contains all of the productions with a marker at the start of the right hand side of each rule. This is the start state of the parser (nothing has been see yet).
- Generate additional configuration sets (parser states) by applying closure and completion until all configuration sets have been generated.
  Derive parser state transitions from the configuration sets.

Some Vital Definitions

- Define: *configuration item*
  A configuration item is a marked production rule
  $G \to \alpha\ \bullet\ \beta\, \gamma$
  with $\alpha, \gamma\ \epsilon\ (V_n \cup V_t)^*$ and $\beta\ \epsilon\ (V_n \cup V_t)$
  A configuration item represents a partial phrase,
  α seen so far $\beta\, \gamma$ to be seen
  Example: L $\to$ L , $\bullet$ E
- Define: *closure*
  if $\bullet$ is immediately to the left of a nonterminal symbol B
  Add to the configuration set all *new* configuration items such that
  B is the left hand side of a rule, i.e. $B \to\ \bullet\ \omega$

Definitions (Cont'd)

- Define: *completion*

  Collect together in a new configuration set all configuration items
  that have the same symbol after the $\bullet$ (e.g. β)

  Complete by moving the $\bullet$ past the symbol, e.g. $G \rightarrow \alpha\, \beta \bullet \gamma$
  if $\beta \,\epsilon\, V_t$ then this corresponds to SHIFT β

- Save space by eliminating duplicate configuration sets as they are generated.

- Initial configuration set, s_0, is created by closing an augmenting production:
  s_0 = closure($\{$S $\rightarrow \bullet\, \alpha \vdash \}$)

Sample Grammar (G_1, p. 147)

S $\rightarrow$ E $\vdash$

E $\rightarrow$ E + T $|$ T

T $\rightarrow$ ID $|$ (E)

Configuration sets

0	$\{$ S $\rightarrow \bullet$ E $\vdash$ E $\rightarrow \bullet$ E + T E $\rightarrow \bullet$ T T $\rightarrow \bullet$ id T $\rightarrow \bullet$ (E)$\}$
1	$\{$S $\rightarrow$ E $\bullet \vdash$ E $\rightarrow$ E $\bullet$ + T$\}$
2	$\{$S $\rightarrow$ E $\vdash \bullet$ $\}$
3	$\{$E $\rightarrow$ E + $\bullet$ T T $\rightarrow \bullet$ id T $\rightarrow \bullet$ (E)$\}$
4	$\{$E $\rightarrow$ E + T $\bullet$ $\}$
5	$\{$T $\rightarrow$ id $\bullet$ $\}$
6	$\{$T $\rightarrow$ ($\bullet$ E) E $\rightarrow \bullet$ E + T E $\rightarrow \bullet$ T T $\rightarrow \bullet$ id T $\rightarrow \bullet$ (E)$\}$
7	$\{$T $\rightarrow$ (E $\bullet$) E $\rightarrow$ E $\bullet$ + T $\}$
8	$\{$T $\rightarrow$ (E) $\bullet$ $\}$
9	$\{$E $\rightarrow$ T $\bullet$ $\}$

Characteristic FSM for G_1

Figure 6.11, p. 152 of Fisher and LeBlanc

Constructing Action Tables

- Reduce$_i$ if B $\rightarrow \alpha$ $^\bullet$ $\in$ s and production i is B $\rightarrow \alpha$
- If A $\rightarrow \alpha$ $^\bullet$ a β $\in$ s for a $\in$ V$_t$, then Shift else Error
- If more than one action is possible, have conflict (shift/reduce or reduce/reduce) and grammar is not LR(0).

Example:

G_1 is LR(0):

State	0	1	2	3	4	5	6	7	8	9	10
Action	S	S	A	S	R2	R4	S	S	R5	R3	

Interesting Observation

- Shift/reduce conflicts are unavoidable in grammars with ϵ productions, with precedence and associativity, etc.
  Example: A := B+C*D
- This makes LR(0) parsing hard to write for "real" programming languages
- But if there is an endmarker and can be parsed by any LR-style parsing technique with any amount of lookahead, then an equivalent LR(0) grammar must exist!!!!
- Can be large and cumbersome, though...

LR(1) Parsing

- Define: *Lookahead Set*
  The lookahead set for a production $A \rightarrow \alpha$ is the set of terminal symbols that can legally follow A or α during a *rightmost canonical parse*.
- A terminal symbol b is in $follow(\ A\ )$ if b can legally follow A in a sentential form during *any* parse. Therefore
  $lookahead(\ A\ )\ \subset\ follow(\ A\ )$
- Example:
  Accept $\rightarrow$ $^\bullet$ L $\qquad \{\vdash\}$
- Definition: A LR(1) *configuration* is of the form
  A $\rightarrow$ X$_1$ ... X$_i$ $^\bullet$ X$_{i+1}$ ... X$_j$, $\quad \{l_1 \ldots l_m\}$
- Initial configuration:
  S $\rightarrow$ $^\bullet$ $\alpha \vdash, \{\epsilon\}$

LR(1) Parsing (Cont'd)

- Definition: *closure1*
  In case B $\rightarrow \sigma$ $^\bullet$ A δ, l,
  add all configurations of form A $\rightarrow$ $^\bullet$ γ, u, where $u \in$ First($\delta\ l$)
- Example: closure1 (S $\rightarrow$ $^\bullet$ E $\vdash$, $\{\epsilon\}$
  $\{$ S $\rightarrow$ $^\bullet$ E $\vdash$, $\{\epsilon\}$; E $\rightarrow$ $^\bullet$ E + T, $\{\vdash +\}$;
  E $\rightarrow$ $^\bullet$ T, $\{\vdash +\}$; T $\rightarrow$ $^\bullet$ id $\{\vdash +\}$; T $\rightarrow$ $^\bullet$ (E), $\{\vdash +\}$ $\}$
- Thus can have $|\ V_t\ | \times$ number of LR(0) states
- LR(1) machine and CFSM are closely related. Can expand/contract.

Sample Grammar (G_3, p. 158)

S $\to$ E $\vdash$

E $\to$ E + T $|$ T

T $\to$ T * P $|$ P

P $\to$ id $|$ (E)

LR(1) machine contains 23 states, and CFSM contains just 13.

LR(1) Machine for G_3

Figure 6.16, p. 160

Constructing LR(1) Action Tables

- Reduce$_i$ if B $\to \alpha \bullet$, a $\in$ s and production i is B $\to \alpha$
- If A $\to \alpha \bullet$ a β, b $\in$ s for a $\in$ V$_t$, then Shift else Error
- If more than one action is possible for each lookahead, have a conflict, and grammar is not LR(1).

Example: Action function for G_3

Figure 6.17, p. 162

Practical Table Construction

- Problem: LR(0) is efficient but not powerful. LR(1) is powerful but not efficient at all.
- Approaches to solve the problem:
  Start with LR(0) and *add lookahead* after it is built (SLR(1))
  Start with LR(1) and *merge* inessential LR(1) states. If merge all states that correspond to same CFSM state, we get LALR(1) technique
- LALR parser generation is slower than SLR parser generation.
  LALR parsers are smaller and faster than SLR parsers.
- Expressive power: $\mathcal{L}(\text{LR(0)}) \subset \mathcal{L}(\text{SLR(1)}) \subset \mathcal{L}(\text{LALR(1)}) \subset \mathcal{L}(\text{LR(1)})$

food for slide eater

food for slide eater

food for slide eater