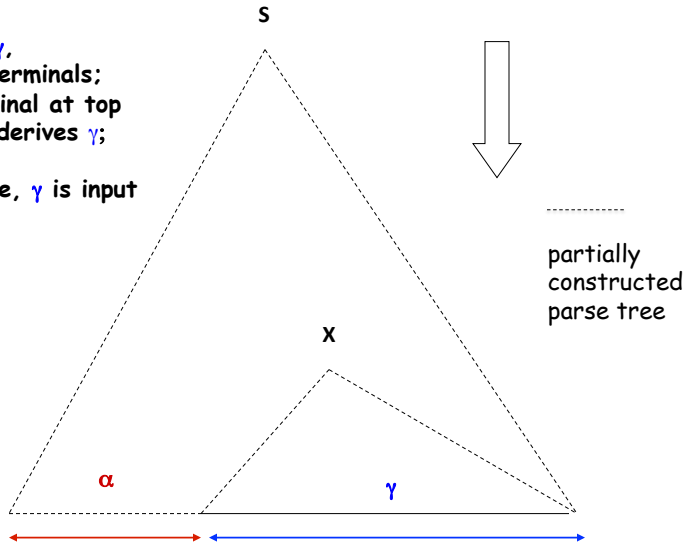
Parsing -3

- Deterministic table-driven parsing techniques
  - Pictorial view of TD and BU parsing
  - BU (shift-reduce) Parsing
    - Handle, viable prefix, items, closures, goto's
    - LR(k): SLR(1), LR(1)
    - Problems with SLR
    - LALR(k) an optimization
  - Using ambiguity to an advantage

Aho, Sethi, Ullman, Compilers : Principles, Techniques and Tools
Aho + Ullman, Theory of Parsing and Compiling, vol II.

A View During TD Parsing

S derives α γ,
a string of terminals;
X is nonterminal at top of stack, X derives γ;
Initially
X==S, a == e, γ is input

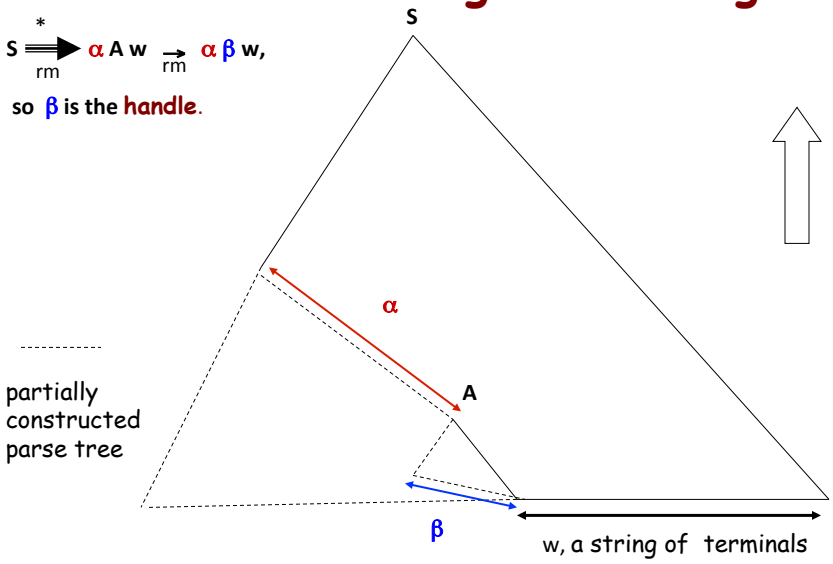


partially constructed parse tree

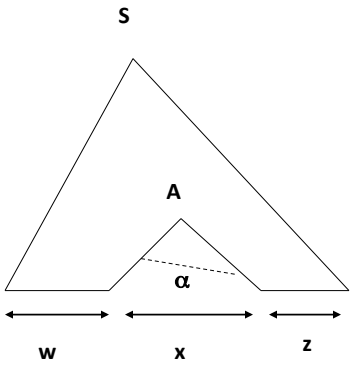
A View During BU Parsing

$S \overset{*}{\underset{rm}{\Longrightarrow}} \alpha A w \underset{rm}{\rightarrow} \alpha \beta w$,

so β is the handle.



Intuitive Comparison



LR(k) can recognize $A \rightarrow \alpha$ knowing w, x, and $First_k(z)$.
LL(k) can recognize $A \rightarrow \alpha$ knowing only w and $First_k(x)$.
Therefore, the set of languages recognizable by LR(k) contain those recognizable by LL(k).

BU Parsing (Shift-Reduce)

Handle - part of sentential form last added in a rightmost derivation.

BU parsing is *"handle hunting"*

(1) S → E
(2) E → E + T
(3) E → T
(4) T → *id*

Rightmost derivation of a+b+c, handles in red

S → E
→ E + T
→ E + *id*
→ E + T + *id*
→ E + *id* + *id*
→ T + *id* + *id*
→ *id* + *id* + *id*

Shift-Reduce Parser, Example

Actions: shift, reduce, accept, error

(1) S → E
(2) E → E + T
(3) E → T
(4) T → id

Stack	Input	Action
$	id1 + id2 + id3 $	shift
$ id1	+ id2 + id3 $	reduce (4)
$ T	+ id2 + id3 $	reduce (3)
$ E	+ id2 + id3 $	shift
$ E +	id2 + id3 $	shift
$ E + id2	+ id3 $	reduce(4)
$ E + T	+ id3 $	reduce (2)
$ E	+ id3 $	shift
$ E +	id3 $	shift
$ E + id3	$	reduce (4)
$ E + T	$	reduce(2)
$ E	$	reduce (1)
$ S	$	accept

Problems

Shift-reduce conflicts

S → if E then S | if E then S else S | other

On stack: if E then S

Input: else

Should shift trying for 2nd alternative or reduce by first rule?

Reduce-reduce conflicts

if A → α and B → α both in grammar

When α on stack, how do we know which production to choose?

Predictive Parsing

- Top Down: LL(k), Bottom Up: LR(k)
- Avoids backtracking while parsing by using lookahead into input
- NO cases where more than 1 action possible
- LR parsing algorithms developed in the mid-1970s; powerful enablers of table-driven compilation

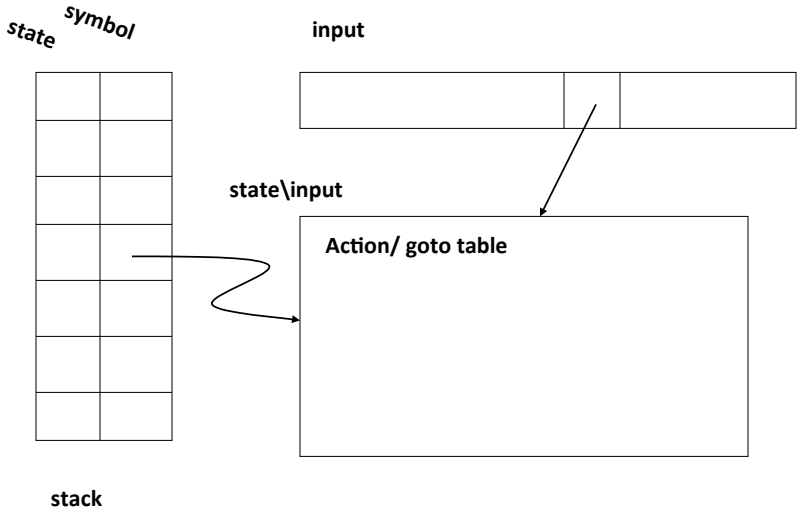
LR(k)

- Left to right scan parsing does a rightmost derivation in reverse, using k symbols of lookahead into input
- Examples
  - Simple LR - SLR(1)
    - Cheap but doesn't always work
  - LR(k)
    - Most powerful and most expensive
- All SLR(1) languages are also LR(1), but parsers generated by corresponding grammars for the same language will differ in size.
- LR(k) catches syntax errors as early as possible in a left-to-right scan of the input and works for most modern PLs

LR Parsing

- FSA is embedded in parser which is a Pushdown automaton
- (top_{stack} , input_symbol) accesses a particular entry in the parser table
  - Shift to state s
  - Reduce by $A \rightarrow \beta$
  - Accept
  - Error
- Goto: (state, top_{stack}) → state

LR Parser



LR Parsing

- *Idea:* continue to stack inputs until have handle on top of stack and then reduce to its non-terminal symbol
- Viable prefix - set of prefixes of right sentential forms which can appear on a stack of a shift/reduce parser
- Goto function is transition function of DFA that recognizes viable prefixes of the grammar
- Idea is that while a viable prefix is on top of the stack the goto function continues the parse towards getting a handle on top of the stack that can be reduced

Building an SLR Parser

- Need states, goto's, Follow sets
- Item - rule with embedded dot

  $S \rightarrow . E$
- Closure of item I

  $I \cup \{B \rightarrow .\gamma \text{ , if } A \rightarrow \alpha . B \beta \text{ in } I\}$
- States built from items and their closures

SLR(1) Example - States

$S \rightarrow E$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow id$

I_0: $S \rightarrow . E$
$E \rightarrow . E + T$
$E \rightarrow . T$
$T \rightarrow . id$

Closure of $S \rightarrow . E$

I_1: $S \rightarrow E .$
$E \rightarrow E . + T$

I_2: $E \rightarrow T .$

I_3: $T \rightarrow id .$

I_4: $E \rightarrow E + . T$
$T \rightarrow . id$

I_5: $E \rightarrow E + T .$

Example - Goto's + Follow sets

goto (0, E) = 1
goto (0, T) = 2
goto (4, T) = 5
goto (0, *id*) = 3
goto (1, +) = 4
goto (4, *id*) = 3

$S \rightarrow E$
$E \rightarrow E + T$
$E \rightarrow T$
$T \rightarrow id$

goto ({set of items} , X) =
  closure {[A → α X . β] |
    [A → α . X β] in {set of items}}
  where X is a terminal or nonterminal

Follow(S) = {$}
Follow(E) = Follow(T) = { +, $}

Rules for forming Follow Sets

ASU p 189

1. Follow(S) contains $
2. If $A \rightarrow \alpha X \beta$ then everything in First(β) except ε, is put into Follow (X)
3. If $A \rightarrow \alpha X$ or $A \rightarrow \alpha X \beta$ where First(β) contains ε, then Follow(A) is contained in Follow(X)

Example - Parser Table

si, shift to state I; r(j) reduce by rule j

States\ inputs:				goto's	
	id	+	$	E	T
0	s3			1	2
1		s4	accept		
2		r(3)	r(3)		
3		r(4)	r(4)		
4	s3				5
5		r(2)	r(2)		

Example

Stack	input	action
0	*id1* + *id2* $	s3
0 *id1* 3	+ *id2* $	r(4), goto on T
0 T 2	+ *id2* $	r(3), goto on E
0 E 1	+ *id2* $	s4
0 E 1 + 4	*id2* $	s3
0 E 1 + 4 *id2* 3	$	r(4), goto on T
0 E 1 + 4 T 5	$	r(2), goto on E
0 E 1	$	accept

SLR(1) Parser Rules

- If $A \rightarrow \alpha \,.\, a\, \beta$ is in state I_j and goto(I_j, a) is I_r then (I_j, a) transitions by shift r (sr)
- If $A \rightarrow \alpha \,.$ is in state I_j, set action [j,a] to reduce $A \rightarrow \alpha$ for all a in Follow(A)
  - Note: A != S
- If $S \rightarrow E \,.$ in I_j, action (j,$) is accept
- Any table entry not defined is error.

Problems

- Shift-reduce conflicts happen when Ab can occur in some sentential form and b ∈ Follow(A).

S → L = R
S → R
L → * R
L → *id*
R → L

I_0 :
S → . L = R
S → . R
R → . L
L → . * R
L →. *id*

I_1 :
S → L . = R (1)
R → L . (2)

In state I_1 1st choice: shift when see = in input(item 1);
2nd choice: reduce on = because = in Follow(R) (item 2);
Note: S → L = R → * R = R …, but this is not a rightmost derivation!

Problems, cont.

Can see that a rightmost derivation is:

S → L = R → L = L → L = *id* → * R = *id* →

*L = *id* → * *id* = id

Therefore, should reduce *R to L when see =, not shift in order to get *R onto the stack.

Problem is that we can't distinguish those Follow elements corresponding to a rightmost derivation in a specific context.

Nomenclature in ASU

- An item [A → β . γ] is *valid* for *viable prefix* α β if S $\overset{*}{\underset{rm}{\Longrightarrow}}$ α A w $\underset{rm}{\Longrightarrow}$ α β γ w.
  - Means can continue towards accumulating an handle on the stack by shifting
  - Previously, shift would have changed viable prefix *R to nonviable prefix *R=
- If I is set of items valid for *viable prefix* β then goto(I, X) is set of items valid for *viable prefix* βX where X is terminal or nonterminal

LR(1) Parsing

- LR items include a lookahead symbol, (into the input) which helps in conflict resolution
- Need new closure rule:
  - For [A → α . B γ , a] item add [B → . δ , b] for every b in First(γ a).

Example

I_0: S → . E, $ - initial item

E → . E + T, $ - closure initial item

E → . T, $

E → . E + T, + - closure 1st red item

E → . T, +

T → .*id* , $ - closure 2nd red item

T → .*id*, + - closure 2nd blue item

Will write these in more compact form by combining lookaheads.

For [A → a . B g ,a] item add [B → . d ,b] for every b in First(g a).

Example, LR(1) Parser

I_0: S → .E, $
  E → .E + T, $/+
  E → . T, $/+
  T → .*id* , +/$

I_4:[goto(I_1 , +)]
  E → E + . T, $/+
  T → . *id*, $/+

I_5: [goto (I_4, T)]
  E → E + T . , $/+

I_1: [goto (I_0 , E)]
  S → E ., $
  S → E . + T, $/+

I_2:[goto (I_0 , T)]
  E → T., $/+

I_3: [goto (I_0 , *id*)]
  T → *id* . , $/+

LR(1) Parser

- Reduce based on lookaheads in item which are a subset of Follow set
- Rules similar to SLR(1)
  - Shift in I_k, [A → α . a β, b], goto (I_k, a) = I_j
  - Reduce [A → α . , b] reduce α to A on b
  - Accept [S → E., $], accept on $

LALR Parsing

- *Idea:* merge all states with common first components in their LR(1) items
- Implementation problem: need to reduce number of states to get smaller parser table
- Reduced size parser will perform
  - Same as LR on correct inputs (will be parsed by LALR)
  - On incorrect inputs, LR may find error faster; LALR will never do an incorrect shift but may do some wrong reductions

LALR Parsing

- Conceptually, build LALR(1) parser from LR(1) parser
  - Merge all states with same first components
  - Union all goto's of these merged states (goto's are independent of second components)
- Correctness of conceptual derivation
  - Can never produce a shift-reduce conflict or else $[A \rightarrow \alpha \, . \, , a]$ and $[B \rightarrow \beta \, . \, a \, \gamma \, , b]$ existed in some LR(1) state

Useful Ambiguous Grammars

- Used to build compact parse trees
  - Get rid of useless nonterminal to nonterminal productions (e.g., S-->E-->T)
- Conflicts resolvable through desired properties of operators (e.g., precedence)
- Generate smaller parsers
  - Example of expression grammar

Example

$S \rightarrow E$
$E \rightarrow E + E$
$E \rightarrow id$

I_0: $S \rightarrow . E$
$E \rightarrow . E + E$
$E \rightarrow . id$

I_1: goto (I_0 ,E)
$S \rightarrow E.$
$E \rightarrow E . + E$

I_2: goto (I_1 ,+)
$E \rightarrow E + . E$
$E \rightarrow . E + E$
$E \rightarrow . id$

I_3: goto (I_2 ,E)
$E \rightarrow E + E .$
$E \rightarrow E . + E$
(reduce on + in Follow(E), shift on +)

I_4: goto(I_0 ,*id*)
$E \rightarrow id .$

Choose reduce action making + left associative; can resolve operator precedence clashes the same way (e.g., + versus *)

Grammar Classification