Parsing

- TD parsing - LL(1)
  - First and Follow sets
  - Parse table construction
- BU Parsing
  - Handle, viable prefix, items, closures, goto's
  - LR(k): SLR(1), LR(1), LALR(1)
    - Problems with SLR
- *Aho, Sethi, Ullman, __Compilers : Principles, Techniques and Tools__*
- *Aho + Ullman, __Theory of Parsing and Compiling, vol II.__*

TD Parsing

Elimination of left recursion.

$E \rightarrow E\,\alpha \mid \beta$ **becomes** $E \rightarrow \beta\, A$

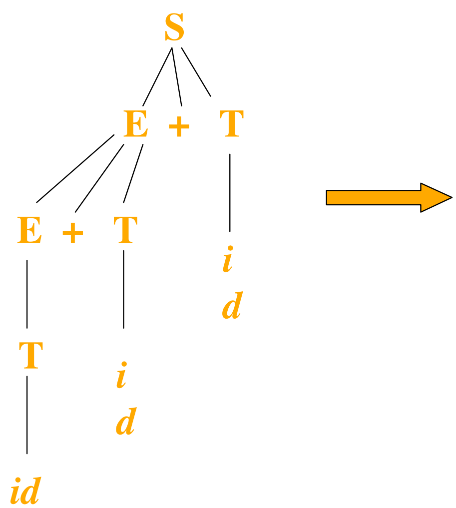
$A \rightarrow \alpha\, A \mid \varepsilon$

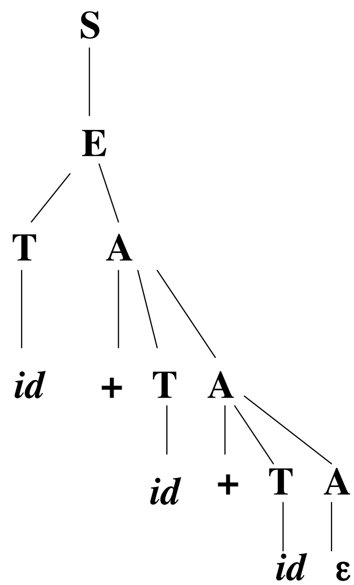
Example:

$S \rightarrow E$		$S \rightarrow E$
$E \rightarrow E + T$		$E \rightarrow T\ A$
$E \rightarrow T$	$\Longrightarrow$	$A \rightarrow +\ T\ A \mid \varepsilon$
$T \rightarrow id$		$T \rightarrow id$

Can also left factor the grammar removing shared prefixes of right-hand-sides.

Parse Tree





Parse tree converted from left recursive to right recursive.

TD Parsing

- ***Problem:*** **predicting which nonterminal to expand next, from a leading string of symbols**
- ***Idea:*** **generate parse tree top down so its frontier is always a sentential form**
  - **Use First and Follow sets to understand the shape of sentential forms possibly generated by the grammar**

TD Stack Parser, EG

$S \rightarrow E$
$E \rightarrow T\ A$
$A \rightarrow +\ T\ A \mid \varepsilon$
$T \rightarrow id$

Stack	Input	Production
\$S	*id+id+id\$*	
\$E	*id+id+id\$*	$S \rightarrow E$
\$A T	*id+id+id\$*	$E \rightarrow T\ A$
\$A	*+id+id\$*	$T \rightarrow id$
\$A T	*id+id\$*	$A \rightarrow +\ T\ A$
\$A	*+id\$*	$T \rightarrow id$
\$A T	*id\$*	$A \rightarrow +\ T\ A$
\$A	\$	$T \rightarrow id$
\$	\$	$A \rightarrow \varepsilon$

See algm in ASU Fig 4.14, p 187

How to mechanize?

- **Define α to be string of nonterminals and terminals**
- **First(α) is the set of terminals that begin strings derivable from α.**

  If $\alpha \overset{*}{\Rightarrow} \varepsilon$, then ε is in First(α).
- **Follow(A) is the set of terminals that can appear directly to the right of A in a sentential form**

  $S \overset{*}{\Rightarrow} \alpha\ A\ a\ \beta$ means a is in Follow(A).

  If A can be rightmost symbol in a sentential form, that is, $X \overset{*}{\Rightarrow} \alpha\ A\ \delta$ where $\delta \overset{*}{\Rightarrow} \varepsilon$, then Follow(A) $\supseteq$ Follow(X).

Example

- **First(S) = First(E) = First(T) = {*id*}**
- **First(A) = { +, ε }**
- **Follow(S) = Follow(E) = Follow(A) = {$}**
- **Follow(T) = {+, $}**

S → E
E → T A
A → + T A | ε
T → *id*

LL(k) Grammars

- **Can choose next production to expand by during TD phase, by looking k symbols ahead into input**
- **Use First sets to choose production**
- **Use Follow sets to handle ε cases**

Example: LL(1)

Nonterms\Inputs:	id	+	$
S	$S \rightarrow E$		
E	$E \rightarrow T\ A$		
T	$T \rightarrow id$		
A		$A \rightarrow + T\ A$	$A \rightarrow \varepsilon$

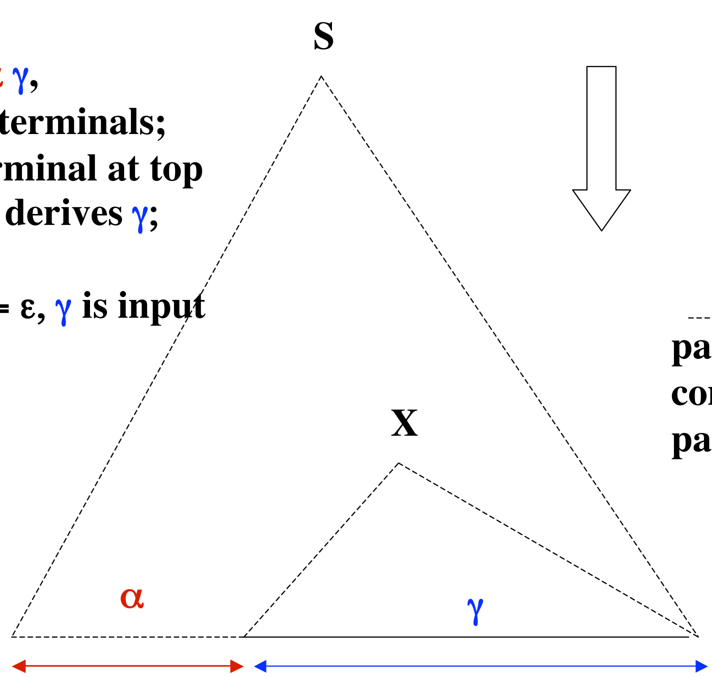
Ambiguous or left recursive grammars result in multiply defined entries in table

First(S) = First(E) = First(T) = {*id*}
First(A) = { +, ε }
Follow(S) = Follow(E) = Follow(A) = {$}
Follow(T) = {+, $}

$S \rightarrow E$
$E \rightarrow T\ A$
$A \rightarrow +\ T\ A \mid \varepsilon$
$T \rightarrow id$

A View During TD Parsing

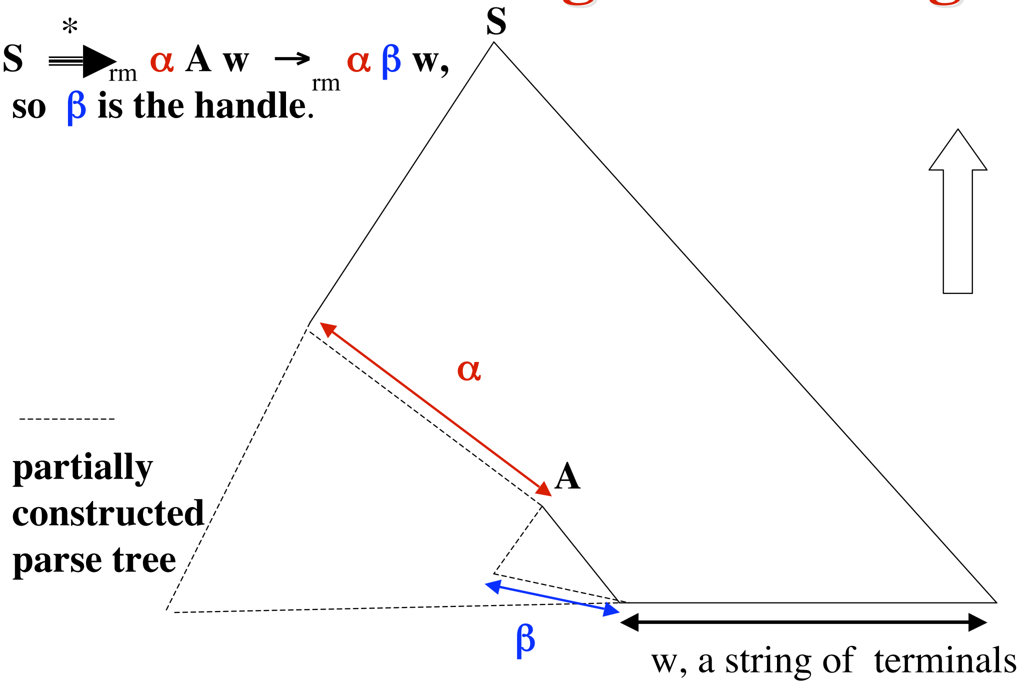
**S derives α γ,
a string of terminals;
X is nonterminal at top of stack, X derives γ;
Initially
X==S, α == ε, γ is input**



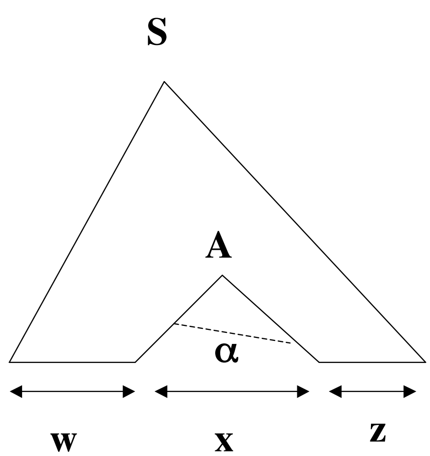
partially constructed parse tree

A View During BU Parsing

$S \overset{*}{\Rightarrow}_{rm} \alpha A w \rightarrow_{rm} \alpha \beta w$,
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 as *"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

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

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

Problems

Shift-reduce conflicts

$S \rightarrow$ 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 $A \rightarrow \alpha$ and $B \rightarrow \alpha$ both in grammar

When α on stack, how 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(k)

- Left to right scan parsing does a rightmost derivation in reverse, using k symbols of lookahead into input
- Three flavors
  - Simple LR, SLR(1)
    - Cheap
    - Doesn't always work
  - LR
    - Most powerful
    - Most expensive

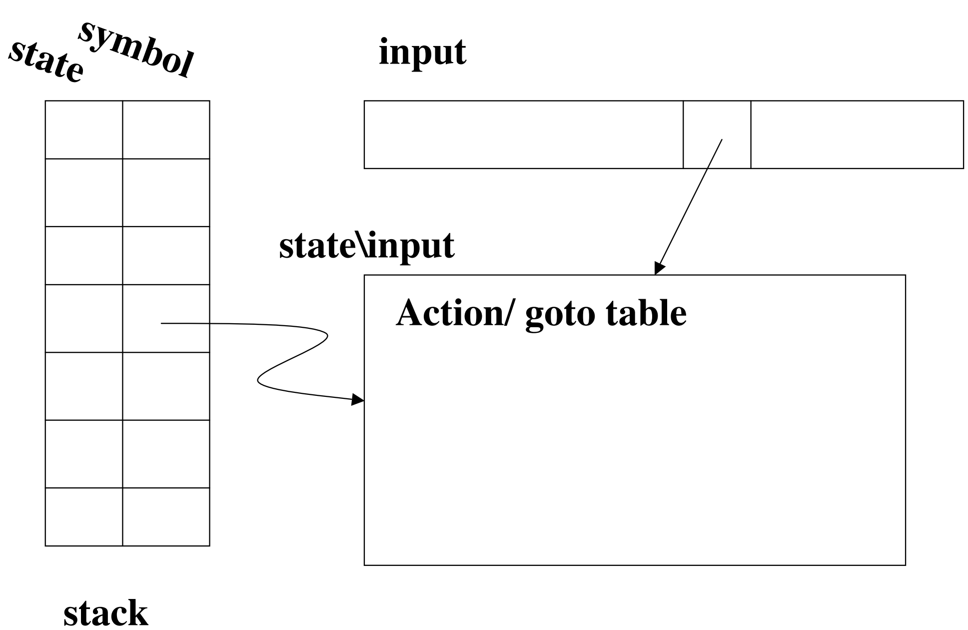
LR(k)

  - LALR
    - Intermediate in cost and power
- 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
- Covers most programming languages

LR Parsing

- DFA is embedded in parser which is a PDA
- (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

- **Viable prefix - set of prefixes of right sentential forms which can appear on a stack of a shift/reduce parser**
  - **Prefix of right sentential form that doesn't contain symbols beyond the handle**
- **Goto function is transition function of DFA that recognizes viable prefixes of the grammar**
- ***Idea:* continue to stack inputs until have handle on top of stack and then reduce**

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**

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

goto ({set of items} , X) =

closure {[$A \rightarrow \alpha X . \beta$] |

[$A \rightarrow \alpha . X \beta$] in {set of items}}

where X is a terminal or nonterminal

Follow(S) = {$}

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

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 $b \in$ 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 I_1 *shift* when see = in input(item 1); *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 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 $\overset{*}{\underset{rm}{\Rightarrow}}$ α A w $\underset{rm}{\Rightarrow}$ 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 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 → α . B γ , a] item add [B → . δ , b] for every b in First(γ 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
  - 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 → α . , a] and [B →β . a γ , b] existed in some LR(1) state**

LALR, cont.

– **But can create** **reduce-reduce** **conflicts**

State 1	**State 2**
[A → c. , d]	[A → c. , e]
[B → c . , e]	[B → c . , d]

After merge:

[A → c. , d/e]

[B → c . , e/d]

Handles (Informal)

Show the handle always remains on the top of the stack in shift-reduce parsing

Assume γ B w is a right sentential form with γ B on the stack.

1. Assume handle includes B.

$$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha_1 A \alpha_2 \underset{rm}{\Rightarrow} \alpha_1 \overbrace{\boxed{\beta_1 \; B \; \beta_2}}^{\text{handle}} \alpha_2$$

γ w

where β_1 or β_2 can be ε.

Handles (Informal)

2. Assume handle included in w.

$$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha_1 \, B \, \alpha_2 \, A \, \alpha_3 \underset{rm}{\Rightarrow} \underbrace{\alpha_1}_{\gamma} \, B \, \underbrace{\alpha_2 \, \boxed{\beta} \, \alpha_3}_{w}$$

(handle: β)

3. Assume handle included in γ.

$$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha_1 \, A \, \alpha_2 \underset{rm}{\Rightarrow} \underbrace{\alpha_1 \, \boxed{\beta}}_{\gamma} \, B \, w$$

(handle: β)

But if β is handle, then A isn't rightmost nonterminal, B is. This is a contradiction!

Handles, More Formally

- $A \rightarrow \beta_1 . \beta_2$ valid for viable prefix $\alpha \, \beta_1$ means $\exists \; S \overset{*}{\underset{rm}{\Rightarrow}} \alpha \; A \; w \underset{rm}{\Rightarrow} \alpha \; \beta_1 \; \beta_2 \; w$

  If $\beta_2 = \varepsilon$ then should reduce by $A \rightarrow \beta_1$

  If $\beta_2 \mathrel{!=} \varepsilon$ then should shift

- Note can have two valid items indicating different actions for same viable prefix

  $A \rightarrow \beta_1 . \beta_2$ and $A \rightarrow \beta_1 .$ Lookahead chooses which action is taken

Handles, More Formally

- **Previous argument shows if $A \rightarrow \beta$. is valid item for viable prefix $\alpha\,\beta$ then α A is viable prefix (i.e., *we needn't rescan the parse after a reduction*)**

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 → E
E → E + E
E → *id*

I_0 :
S → . E
E → . E + E
E → . *id*

I_1 : goto (I_0 ,E)
S → E.
E → E . + E

I_2 : goto (I_1 ,+)
E → E + . E
E → . E + E
E → . *id*

I_3 : goto (I_2 ,E)
E → E + E .
E → E . + E
(reduce on + in Follow(E)
shift on +)

I_4 :goto(I_0 ,*id*)
E → *id* .

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

Grammar Classification