Bottom-Up Syntax Analysis

Reinhard Wilhelm, Sebastian Hack, Mooly Sagiv
Saarland University, Tel Aviv University

W2015

Saarland University, Computer Science

Subjects

- Functionality and Method
- Example Parsers
- Derivation of a Parser
- Conflicts
- $LR(k)$–Grammars
- $LR(1)$–Parser Generation
- Bison

Bottom-Up Syntax Analysis

Input: A stream of symbols (tokens)

Output: A syntax tree or error

Method: **until** input consumed or error **do**

- shift next symbol or reduce by some production
- decide what to do by looking k symbols ahead

Properties:

- Constructs the syntax tree in a bottom-up manner
- Finds the rightmost derivation (in reversed order)
- Reports error as soon as the already read part of the input is not a prefix of a program (valid prefix property)

Parsing $aabb$ in the grammar G_{ab} with $S \to aSb|\epsilon$

Stack	Input	Action	Dead ends
$\$$	$aabb\#$	**shift**	**reduce** $S \to \epsilon$
$\$a$	$abb\#$	**shift**	**reduce** $S \to \epsilon$
$\$aa$	$bb\#$	**reduce** $S \to \epsilon$	**shift**
$\$aaS$	$bb\#$	**shift**	**reduce** $S \to \epsilon$
$\$aaSb$	$b\#$	**reduce** $S \to aSb$	**shift**, **reduce** $S \to \epsilon$
$\$aS$	$b\#$	**shift**	**reduce** $S \to \epsilon$
$\$aSb$	$\#$	**reduce** $S \to aSb$	**reduce** $S \to \epsilon$
$\$S$	$\#$	**accept**	**reduce** $S \to \epsilon$

Issues:

- Shift vs. Reduce
- Reduce $A \to \beta$, Reduce $B \to \alpha\beta$

Parsing aa in the grammar $S \to AB, S \to A, A \to a, B \to a$

Stack	Input	Action	Dead ends
$\$$	$aa\#$	**shift**	
$\$a$	$a\#$	**reduce** $A \to a$	**reduce** $B \to a$, **shift**
$\$A$	$a\#$	**shift**	**reduce** $S \to A$
$\$Aa$	$\#$	**reduce** $B \to a$	**reduce** $A \to a$
$\$AB$	$\#$	**reduce** $S \to AB$	
$\$S$	$\#$	**accept**	

Issues:

- Shift vs. Reduce
- Reduce $A \to \beta$, Reduce $B \to \alpha\beta$

Shift-Reduce Parsers

- The bottom–up Parser is a shift–reduce parser, each step is a

  shift: consuming the next input symbol or

  reduction: reducing a suffix of the stack contents by some production.

- problem is to decide when to stop shifting and make a reduction

- a next right side to reduce is called a handle if

  reducing too early leads to a dead end,

  reducing too late buries the handle

LR-Parsers – Deterministic Shift–Reduce Parsers

Parser decides whether to shift or to reduce based on

- the contents of the stack and
- k symbols lookahead into the rest of the input

Property of the LR–Parser: it suffices to consider the topmost state on the stack instead of the whole stack contents.

From P_G to LR–Parsers for G

- P_G has non-deterministic choice of expansions,
- LL–parsers eliminate non–determinism by looking ahead at expansions,
- LR–parsers pursue all possibilities in parallel (corresponds to the subset–construction in NFSM $\rightarrow$ DFSM).

Derivation:

1. Characteristic finte-state machine of G, a description of P_G
2. Make deterministic
3. Interpret as control of a push down automaton
4. Check for “inedaquate” states

Characteristic Finite-State Machine of G

... is a NFSM $ch(G) = (Q_c, V_c, \Delta_c, q_c, F_c)$:

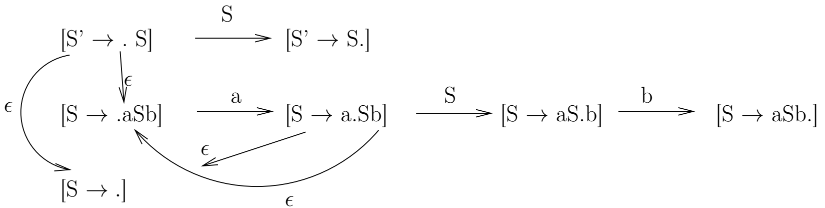
- states are the items of G
  $Q_c = It_G$
- input alphabet are terminals and non-terminals
  $V_c = V_T \cup V_N$
- start state $q_c = [S' \to .S]$
- final states are the complete items
  $F_c = \{[X \to \alpha.] \mid X \to \alpha \in P\}$
- Transitions:

$$\begin{aligned}\Delta_c = & \{([X \to \alpha.Y\beta], Y, [X \to \alpha Y.\beta]) \mid X \to \alpha Y\beta \in P \text{ and } Y \in V_N \cup V_T\} \\ \cup & \{([X \to \alpha.Y\beta], \varepsilon, [Y \to .\gamma]) \mid X \to \alpha Y\beta \in P \text{ and } Y \to \gamma \in P\}\end{aligned}$$

Item PDA and Characteristic NFA

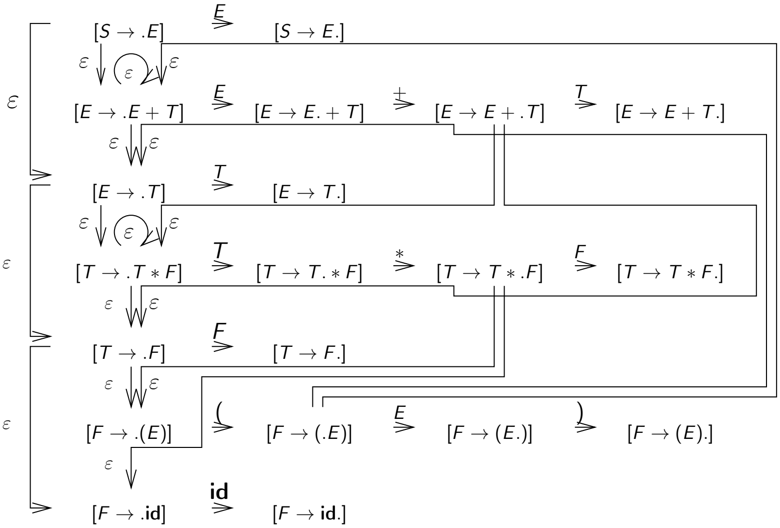
for G_{ab}: $S \rightarrow aSb|\epsilon$ and $ch(G_{ab})$

Stack	Input	New Stack
$[S' \rightarrow .S]$	ϵ	$[S' \rightarrow .S][S \rightarrow .aSb]$
$[S' \rightarrow .S]$	ϵ	$[S' \rightarrow .S][S \rightarrow .]$
$[S \rightarrow .aSb]$	a	$[S \rightarrow a.Sb]$
$[S \rightarrow a.Sb]$	ϵ	$[S \rightarrow a.Sb][S \rightarrow .aSb]$
$[S \rightarrow a.Sb]$	ϵ	$[S \rightarrow a.Sb][S \rightarrow .]$
$[S \rightarrow aS.b]$	b	$[S \rightarrow aSb.]$
$[S \rightarrow a.Sb][S \rightarrow .]$	ϵ	$[S \rightarrow aS.b]$
$[S \rightarrow a.Sb][S \rightarrow aSb.]$	ϵ	$[S \rightarrow aS.b]$
$[S' \rightarrow .S][S \rightarrow aSb.]$	ϵ	$[S' \rightarrow S.]$
$[S' \rightarrow .S][S \rightarrow .]$	ϵ	$[S' \rightarrow S.]$



Characteristic NFSM for G_0

$$S \to E, \quad E \to E + T \mid T, \quad T \to T * F \mid F, \quad F \to (E) \mid \mathbf{id}$$



Interpreting $ch(G)$

State of $ch(G)$ is the *current* state of P_G, i.e. the state on top of P_G's stack. Adding actions to the transitions and states of $ch(G)$ to describe P_G:

ε–transitions: push new state of $ch(G)$ onto stack of P_G: new current state.

reading transitions: shifting transitions of P_G: replace current state of P_G by the shifted one.

final state: Correspond to the following actions in P_G:

- pop final state $[X \to \alpha.]$ from the stack,
- do a transition from the new topmost state under X,
- push the new state onto the stack.

Handles and Reliable Prefixes

Some Abbreviations:
RMD: rightmost derivation
RSF: right sentential form

Consider a RMD of cfg G:

$$S' \overset{*}{\underset{rm}{\Longrightarrow}} \beta X u \underset{rm}{\Longrightarrow} \beta \alpha u$$

- α is a handle of $\beta\alpha u$.
  The part of a RSF next to be reduced.

- Each prefix of $\beta\alpha$ is a reliable prefix.
  A prefix of a RSF stretching at most up to the end of the handle,
  i.e. reductions if possible then only at the end.

Examples in G_0

RSF (<u>handle</u>)	reliable prefix	Reason
$E + \underline{F}$	$E,\ E+,\ E+F$	$S \underset{rm}{\Longrightarrow} E \underset{rm}{\Longrightarrow} E+T \underset{rm}{\Longrightarrow} E+F$
$T * \underline{\mathbf{id}}$	$T,\ T*,\ T*\mathbf{id}$	$S \overset{3}{\underset{rm}{\Longrightarrow}} T*F \underset{rm}{\Longrightarrow} T*\mathbf{id}$
$\underline{F} * \mathbf{id}$	F	$S \overset{4}{\underset{rm}{\Longrightarrow}} T*\mathbf{id} \underset{rm}{\Longrightarrow} F*\mathbf{id}$
$T * \underline{\mathbf{id}} + \mathbf{id}$	$T,\ T*,\ T*\mathbf{id}$	$S \overset{3}{\underset{rm}{\Longrightarrow}} T*F \underset{rm}{\Longrightarrow} T*\mathbf{id}$

Valid Items

$[X \to \alpha.\beta]$ is valid for the reliable prefix $\gamma\alpha$, if there exists a RMD

$$S' \underset{rm}{\overset{*}{\Longrightarrow}} \gamma X w \underset{rm}{\Longrightarrow} \gamma\alpha\beta w$$

An item valid for a reliable prefix gives one interpretation of the parsing situation.

Some reliable prefixes of G_0

Reliable Prefix	Valid Items	Reason	γ	w	X	α	β
$E+$	$[E \to E+.T]$	$S \underset{rm}{\Longrightarrow} E \underset{rm}{\Longrightarrow} E+T$	ε	ε	E	$E+$	T
	$[T \to .F]$	$S \underset{rm}{\overset{*}{\Longrightarrow}} E+T \underset{rm}{\Longrightarrow} E+F$	$E+$	ε	T	ε	F
	$[F \to .\mathbf{id}]$	$S \underset{rm}{\overset{*}{\Longrightarrow}} E+F \underset{rm}{\Longrightarrow} E+\mathbf{id}$	$E+$	ε	F	ε	$\mathbf{id}$
$(E+($	$[F \to (.E)]$	$S \underset{rm}{\overset{*}{\Longrightarrow}} (E+F)$	$(E+$	$)$	F	$($	$E)$
		$\underset{rm}{\Longrightarrow} (E+(E))$					

Valid Items and Parsing Situations

Given some input string $xuvw$.

The RMD $\quad S' \overset{*}{\underset{rm}{\Longrightarrow}} \gamma X w \underset{rm}{\Longrightarrow} \gamma\alpha\beta w \overset{*}{\underset{rm}{\Longrightarrow}} \gamma\alpha v w \overset{*}{\underset{rm}{\Longrightarrow}} \gamma u v w \overset{*}{\underset{rm}{\Longrightarrow}} x u v w$

describes the following sequence of partial derivations:

$$\gamma \overset{*}{\underset{rm}{\Longrightarrow}} x \qquad \alpha \overset{*}{\underset{rm}{\Longrightarrow}} u \qquad \beta \overset{*}{\underset{rm}{\Longrightarrow}} v \qquad X \underset{rm}{\Longrightarrow} \alpha\beta \qquad S' \overset{*}{\underset{rm}{\Longrightarrow}} \gamma X w$$

executed by the bottom-up parser in this order.

The valid item $[X \to \alpha\ .\ \beta]$ for the reliable prefix $\gamma\alpha$ describes the situation after partial derivation 2,
that is, for RSF $\gamma\alpha v w$

Theorems

$ch(G) = (Q_c, V_c, \Delta_c, q_c, F_c)$

Theorem

For each reliable prefix there is at least one valid item.

Every parsing situation is described by at least one valid item.

Theorem

Let $\gamma \in (V_T \cup V_N)^*$ and $q \in Q_c$.
$(q_c, \gamma) \vdash^*_{ch(G)} (q, \varepsilon)$ iff γ is a reliable prefix and q is a valid item for γ.

A reliable prefix brings $ch(G)$ from its initial state to all its valid items.

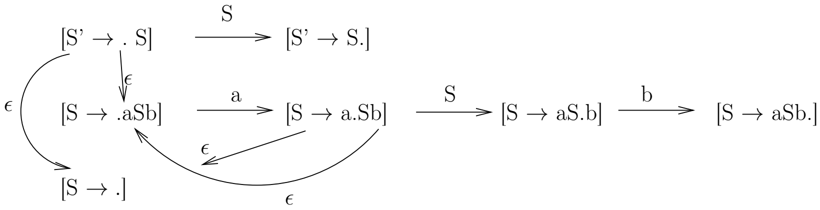
Theorem

The language of reliable prefixes of a cfg is regular.

Making $ch(G)$ deterministic

Apply **NFSM** $\rightarrow$ **DFSM** to $ch(G)$: Result $LR_0(G)$.

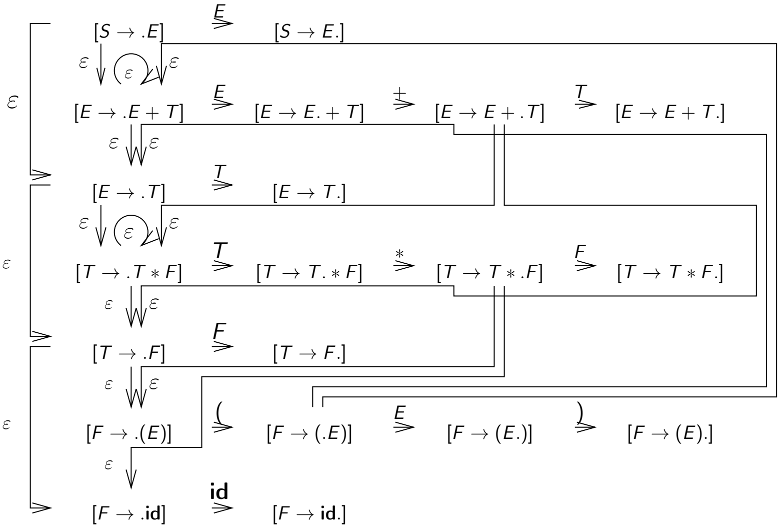
Example: $ch(G_{ab})$



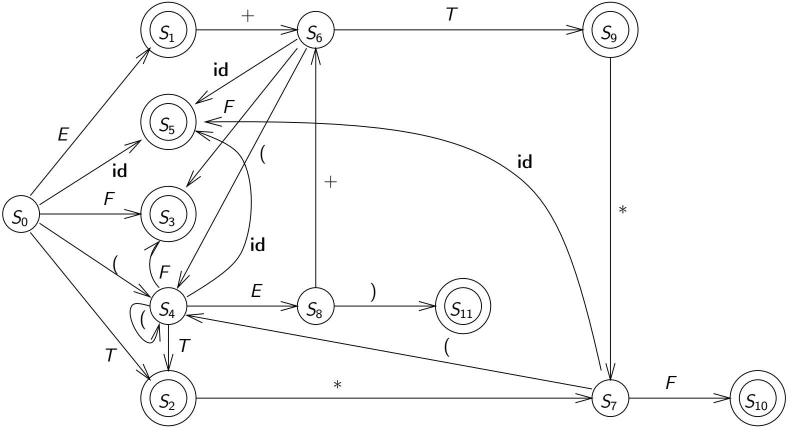
$LR_0(G_{ab})$:

Characteristic NFSM for G_0

$$S \to E, \quad E \to E + T \mid T, \quad T \to T * F \mid F, \quad F \to (E) \mid \mathbf{id}$$



$LR_0(G_0)$



The States of $LR_0(G_0)$ as Sets of Items

$$
\begin{aligned}
S_0 &= \{ [S \to .E], [E \to .E + T], [E \to .T], [T \to .T * F], [T \to .F], [F \to .(E)], [F \to .\mathbf{id}]\} \\
S_1 &= \{ [S \to E.], [E \to E. + T]\} \\
S_2 &= \{ [E \to T.], [T \to T. * F]\} \\
S_3 &= \{ [T \to F.]\} \\
S_4 &= \{ [F \to (.E)], [E \to .E + T], [E \to .T], [T \to .T * F] [T \to .F] [F \to .(E)] [F \to .\mathbf{id}]\} \\
S_5 &= \{ [F \to \mathbf{id}.]\} \\
S_6 &= \{ [E \to E + .T], [T \to .T * F], [T \to .F], [F \to .(E)], [F \to .\mathbf{id}]\} \\
S_7 &= \{ [T \to T * .F], [F \to .(E)], [F \to .\mathbf{id}]\} \\
S_8 &= \{ [F \to (E.)], [E \to E. + T]\} \\
S_9 &= \{ [E \to E + T.], [T \to T. * F]\} \\
S_{10} &= \{ [T \to T * F.]\} \\
S_{11} &= \{ [F \to (E).]\}
\end{aligned}
$$

Theorems

$ch(G) = (Q_c, V_c, \Delta_c, q_c, F_c)$ and $LR_0(G) = (Q_d, V_N \cup V_T, \Delta, q_d, F_d)$

Theorem

Let γ be a reliable prefix and $p(\gamma) \in Q_d$ be the uniquely determined state, into which $LR_0(G)$ transfers out of the initial state by reading γ, i.e., $(q_d, \gamma) \vdash^*_{LR\mathbf{0}(G)} (p(\gamma), \varepsilon)$. Then

(a) $p(\varepsilon) = q_d$

(b) $p(\gamma) = \{q \in Q_c \mid (q_c, \gamma) \vdash^*_{ch(G)} (q, \varepsilon)\}$

(c) $p(\gamma) = \{i \in It_G \mid i \text{ valid for } \gamma\}$

(d) *Let Γ the (in general infinite) set of all reliable prefixes of G. The mapping $p : \Gamma \to Q_d$ defines a finite partition on Γ.*

(e) $L(LR_0(G))$ *is the set of reliable prefixes of G that end in a handle.*

G_0

$\gamma \; = \; E + F$ is a reliable prefix of G_0.
With the state $p(\gamma) = \; S_3$ are also associated:
$F, \; (F, \; ((F, \; (((F, \ldots$
$T * (F, \; T * ((F, \; T * (((F, \ldots$
$E + F, \; E + (F, \; E + ((F, \ldots$

Regard S_6 in $LR_0(G_0)$.
It consists of all valid items for the reliable prefix $E+$,
i.e., the items $[E \to E + .T], [T \to .T * F], [T \to .F], [F \to .\mathbf{id}], [F \to .(E)]$.

Reason:
$E+$ is prefix of the RSF $E + T$;

$$\begin{array}{lccccccc}
S \underset{rm}{\Longrightarrow} E \underset{rm}{\Longrightarrow} & E + T & \underset{rm}{\Longrightarrow} & E + F & \underset{rm}{\Longrightarrow} & E + \mathbf{id} & & \\
& \uparrow & & \uparrow & & \uparrow & & \text{are valid.} \\
\text{Therefore} & [E \to E + .T] & & [T \to .F] & & [F \to .\mathbf{id}] & &
\end{array}$$

What the $LR_0(G)$ describes

$LR_0(G)$ interpreted as a PDA $P_0(G) = (\Gamma, V_T, \Delta, q_0, \{q_f\})$

- Γ (stack alphabet): the set Q_d of states of $LR_0(G)$.
- $q_0 = q_d$ (initial state): in the stack of $P_0(G)$ initially.
- $q_f = \{[S' \to S.]\}$ the final state of $LR_0(G)$,
- $\Delta \subseteq \Gamma^* \times (V_T \cup \{\varepsilon\}) \times \Gamma^*$ (transition relation):
  Defined as follows:

$LR_0(G)$'s Transition Relation

shift: $(q, a, q\,\delta_d(q, a)) \in \Delta$, if $\delta_d(q, a)$ defined.
Read next input symbol a and push successor state of q under a (item $[X \to \cdots .a \cdots] \in q$).

reduce: $(q\,q_1 \dots q_n, \varepsilon, q\,\delta_d(q, X)) \in \Delta$,
if $[X \to \alpha.] \in q_n,\ |\alpha| = n$.
Remove $|\alpha|$ entries from the stack.
Push the successor of the new topmost state under X onto the stack.

Note the difference in the stacking behavior:

- the Item PDA P_G keeps on the stack only one item for each production under analysis,
- the PDA described by the $LR_0(G)$ keeps $|\alpha|$ states on the stack for a production $X \to \alpha\beta$ represented with item $[X \to \alpha.\beta]$

Reduction in PDA $P_0(G)$

$$\begin{tikzcd}
\left\{ \begin{array}{c} [\cdots \to \cdots .X \cdots] \\ [X \to .\alpha] \\ \cdots \end{array} \right\} \arrow[r, "X"] \arrow[dr, squiggly, "\alpha"] & \left\{ \begin{array}{c} [\cdots \to \cdots X. \cdots] \\ \cdots \end{array} \right\} \\
& \left\{ \begin{array}{c} [X \to \alpha.] \\ \cdots \end{array} \right\}
\end{tikzcd}$$

Some observations and recollections

- also works for reductions of ϵ,
- each state has a unique entry symbol,
- the stack contents uniquely determine a reliable prefix,
- current state (topmost) is the state associated with this reliable prefix,
- current state consists of all items valid for this reliable prefix.

Non-determinism in $P_0(G)$

$P_0(G)$ is non-deterministic if either

Shift–reduce conflict: There are shift as well as reduce transitions out of one state, or

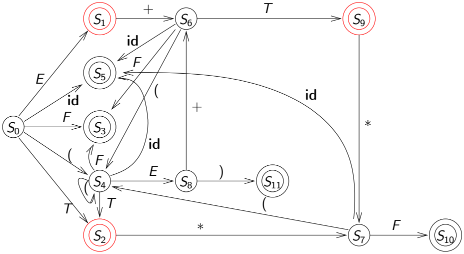
Reduce–reduce conflict: There are more than one reduce transitions from one state.

States with a shift–reduce conflict have at least one read item $[X \to \alpha\, .a\, \beta]$ and at least one complete item $[Y \to \gamma.]$.

States with a reduce–reduce conflict have at least two complete items $[Y \to \alpha.]$, $[Z \to \beta.]$.

A state with a conflict is **inadequate**.

Some Inadequate States



$LR_0(G_0)$ has three inadequate states, S_1, S_2 and S_9.

S_1: Can reduce E to S (complete item $[S \to E.]$) or read "+" (shift–item $[E \to E. + T]$);

S_2: Can reduce T to E (complete item $[E \to T.]$) or read "$*$" (shift-item $[T \to T. * F]$);

S_9: Can reduce $E + T$ to E (complete item $[E \to E + T.]$) or read "$*$" (shift–item $[T \to T. * F]$).

Adding Lookahead

- LR(k) item $[X \to \alpha_1.\alpha_2, L]$
  if $X \to \alpha_1\alpha_2 \in P$ and $L \subseteq V_{T\#}^{\leq k}$
- LR(0) item $[X \to \alpha_1.\alpha_2]$ is called core of $[X \to \alpha_1.\alpha_2, L]$
- lookahead set L of $[X \to \alpha_1.\alpha_2, L]$
- $[X \to \alpha_1.\alpha_2, L]$ is valid for a reliable prefix $\alpha\alpha_1$ if

$$S'\# \underset{rm}{\overset{*}{\Longrightarrow}} \alpha X w \underset{rm}{\Longrightarrow} \alpha\alpha_1\alpha_2 w$$

  and

$$L = \{u \mid S'\# \underset{rm}{\overset{*}{\Longrightarrow}} \alpha X w \underset{rm}{\Longrightarrow} \alpha\alpha_1\alpha_2 w \quad \text{and} \quad u = k : w\}$$

The context–free items can be regarded as LR(0)-items if $[X \to \alpha_1.\alpha_2, \{\varepsilon\}]$ is identified with $[X \to \alpha_1.\alpha_2]$.

Example from G_0

1. $[E \rightarrow E + .T, \{), +, \#\}]$ is a valid LR(1)–item for $(E+$
2. $[E \rightarrow T., \{*\}]$ is not a valid LR(1)-item for any reliable prefix

Reasons:

1. $S' \overset{*}{\underset{rm}{\Longrightarrow}} (E) \underset{rm}{\Longrightarrow} (E + T) \overset{*}{\underset{rm}{\Longrightarrow}} (E + T + \mathbf{id})$ where

$$\alpha = (,\ \alpha_1 = E+,\ \alpha_2 = T,\ u = +,\ w = +\mathbf{id})$$

2. The string $E*$ can occur in no RMD.

LR–Parser

Take their decisions (to shift or to reduce) by consulting

- the reliable prefix γ in the stack, actually the by γ uniquely determined state (on top of the stack),
- the next k symbols of the remaining input.
- Recorded in an **action**–table.
- The entries in this table are:

shift:	read next input symbol;
reduce $(X \to \alpha)$:	reduce by production $X \to \alpha$;
error:	report error
accept:	report successful termination.

A **goto**–table records the transition function of characteristic automaton

The action– and the goto–table

action-table

$V_{T\#}^{\leq k}$

	u
q	parser–action for (q, u)

goto-table

$V_N \cup V_T$

Q

	X
q	$\delta_d(q, X)$

Parser Table for $S \to aSb|\epsilon$

Action–table

state	sets of items	a	b	$\#$
0	$\left\{\begin{array}{l}[S' \to .S],\\ [S \to .aSb],\\ [S \to .]\end{array}\right\}$	s		$r(S \to \epsilon)$
1	$\left\{\begin{array}{l}[S \to a.Sb],\\ [S \to .aSb],\\ [S \to .]\end{array}\right\}$	s	$r(S \to \epsilon)$	
2	$\{[S \to aS.b]\}$		s	
3	$\{[S \to aSb.]\}$		$r(S \to aSb)$	$r(S \to aSb)$
4	$\{[S' \to S.]\}$			$accept$

Goto–table

state	a	b	$\#$	S
0	1			4
1	1			2
2		3		
3				
4				

Parsing *aabb*

Stack	Input	Action
\$ 0	$aabb\#$	**shift** 1
\$ 0 1	$abb\#$	**shift** 1
\$ 0 1 1	$bb\#$	**reduce** $S \rightarrow \epsilon$
\$ 0 1 1 2	$bb\#$	**shift** 3
\$ 0 1 1 2 3	$b\#$	**reduce** $S \rightarrow aSb$
\$ 0 1 2	$b\#$	**shift** 3
\$ 0 1 2 3	$\#$	**reduce** $S \rightarrow aSb$
\$ 0 4	$\#$	**accept**

Algorithm LR(1)–PARSER

```
type state = set of item;
var  lookahead: symbol;
     (* the next not yet consumed input symbol *)
     S : stack of state;
proc scan;
     (* reads the next symbol into lookahead *)
proc acc;
     (* report successful parse; halt *)
proc err(message: string);
     (* report error; halt *)
```

```
scan; push(S, q_d);
forever do
   case  action[top(S), lookahead] of
      shift:  begin  push(S, goto[top(S), lookahead]);
                     scan
              end ;
      reduce (X→α) :    begin
                             pop^|α|(S); push(S, goto[top(S), X]);
                             output("X → α")
                        end ;
      accept:   acc;
      error:    err("...");
   end case
od
```

LR(1)–Conflicts

Set of LR(1)-items I has a

shift-reduce-conflict:

if exists at least one item $[X \to \alpha.a\beta, L_1] \in I$
and at least one item $[Y \to \gamma., L_2] \in I$,
and if $a \in L_2$.

reduce-reduce-conflict:

if it contains at least two items $[X \to \alpha., L_1]$
and $[Y \to \beta., L_2]$ where $L_1 \cap L_2 \neq \emptyset$.

A state with a conflict is called **inadequate**.

Example from G_0

$$S'_0 = \; Closure(\text{Start}) = \{[S \to .E, \{\#\}], [E \to .E + T, \{\#, +\}], [E \to .T, \{\#, +\}], [T \to .T * F, \{\#, +, *\}], [T \to .F, \{\#, +, *\}], [F \to .(E), \{\#, +, *\}], [F \to .\mathbf{id}, \{\#, +, *\}] \;\}$$

$$S'_1 = \; Closure(Succ(S'_0, E)) = \{[S \to E., \{\#\}], [E \to E. + T, \{\#, +\}] \;\}$$

$$S'_2 = \; Closure(Succ(S'_0, T)) = \{[E \to T., \{\#, +\}], [T \to T. * F, \{\#, +, *\}] \;\}$$

$$S'_6 = \; Closure(Succ(S'_1, +)) = \{[E \to E + .T, \{\#, +\}], [T \to .T * F, \{\#, +, *\}], [T \to .F, \{\#, +, *\}], [F \to .(E), \{\#, +, *\}], [F \to .\mathbf{id}, \{\#, +, *\}] \;\}$$

$$S'_9 = \; Closure(Succ(S'_6, T)) = \{[E \to E + T., \{\#, +\}], [T \to T. * F, \{\#, +, *\}] \;\}$$

Inadequate LR(0)–states S_1, S_2 und S_9 are adequate after adding lookahead sets.

S'_1 shifts under "$+$", reduces under "$\#$".

S'_2 shifts under "$*$", reduces under "$\#$" and "$+$",

S'_9 shifts under "$*$", reduces under "$\#$" and "$+$".

Operator Precedence Parsing

G_0 encodes operator precedence and associativity and used lookahead in an LR(1) parser to disambiguate.

Idea: Use ambiguous grammar G_0':

$$E \to E + E \mid E * E \mid \mathbf{id} \mid (E)$$

and operator precedence and associativity to disambiguate directly.

Deterministic $ch(G_0')$

...contains two states:

$$S_7 : \begin{array}{lcl} E & \rightarrow & E + E. \\ E & \rightarrow & E. + E \\ E & \rightarrow & E. * E \end{array} \qquad S_8 : \begin{array}{lcl} E & \rightarrow & E * E. \\ E & \rightarrow & E. + E \\ E & \rightarrow & E. * E \end{array}$$

with shift reduce conflicts.

In both states, the parser can reduce or shift either $+$ or $*$.

$ch(G_0')$ conflicts in detail

- Consider the input $\mathbf{id} + \mathbf{id} * \mathbf{id}$

  and let the top of the stack be S_7.
  - If reduce, then $+$ has higher precendence than $*$
  - If shift, then $+$ has lower precendence than $*$

- Consider the input $\mathbf{id} + \mathbf{id} + \mathbf{id}$

  and let the top of the stack be S_7.
  - If reduce, $+$ is left-associative
  - If shift, $+$ is right-associative

Simple Implementation for Expression Parser

- Model precedence/assoc with left and right precedence
- Shift/reduce mechanism can be implemented with loop and recursion:

```
Expression parseExpression(Precedence precedence) {
    Expression expr = parsePrimary();
    for (;;) {
        Token     t    = currToken;
        TokenKind kind = t.getKind();

        // if operator in lookahead has less left precedence: reduce
        if (kind.getLPrec() < precedence)
            return expr;

        // else shift
        nextToken();

        // and parse other operand with right precedence
        Expression right = parseExpression(kind.getRPrec());
        expr = factory.createBinaryExpression(t, expr, right);
    }
}
```