Bottom-Up Syntax Analysis

Sebastian Hack
(based on slides by Reinhard Wilhelm and Mooly Sagiv)

http://compilers.cs.uni-saarland.de

Compiler Construction Core Course 2017
Saarland University
Topics

- Functionality and Method
- Example Parsers
- Derivation of a Parser
- Conflicts
- $LR(k)$–Grammars
- $LR(1)$–Parser Generation
- Precedence Climbing
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 $S \rightarrow aSb|\epsilon$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
<th>Dead ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>$aabb#$</td>
<td>shift</td>
<td>reduce $S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$a$</td>
<td>$abb#$</td>
<td>shift</td>
<td>reduce $S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$aa$</td>
<td>$bb#$</td>
<td>reduce $S \rightarrow \epsilon$</td>
<td>shift</td>
</tr>
<tr>
<td>$aaS$</td>
<td>$bb#$</td>
<td>shift</td>
<td>reduce $S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$aaSb$</td>
<td>$b#$</td>
<td>reduce $S \rightarrow aSb$</td>
<td>shift, reduce $S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$aS$</td>
<td>$b#$</td>
<td>shift</td>
<td>reduce $S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$aSb$</td>
<td>$#$</td>
<td>reduce $S \rightarrow aSb$</td>
<td>reduce $S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$S$</td>
<td>$#$</td>
<td>accept</td>
<td>reduce $S \rightarrow \epsilon$</td>
</tr>
</tbody>
</table>

### Issues:

- Shift vs. Reduce
- Reduce $A \rightarrow \beta$, Reduce $B \rightarrow \alpha\beta$
Parsing $aa$ in the grammar $S \rightarrow AB, S \rightarrow A, A \rightarrow a, B \rightarrow a$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
<th>Dead ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>$</td>
<td>aa#</td>
<td>shift</td>
<td></td>
</tr>
<tr>
<td>$a$</td>
<td>a#</td>
<td>reduce $A \rightarrow a$</td>
<td>reduce $B \rightarrow a$, shift</td>
</tr>
<tr>
<td>$A$</td>
<td>a#</td>
<td>shift</td>
<td>reduce $S \rightarrow A$</td>
</tr>
<tr>
<td>$Aa$</td>
<td>#</td>
<td>reduce $B \rightarrow a$</td>
<td>reduce $A \rightarrow a$</td>
</tr>
<tr>
<td>$AB$</td>
<td>#</td>
<td>reduce $S \rightarrow AB$</td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>#</td>
<td>accept</td>
<td></td>
</tr>
</tbody>
</table>

Issues:

- Shift vs. Reduce
- Reduce $A \rightarrow \beta$, Reduce $B \rightarrow \alpha\beta$
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
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 $\text{NFSM} \rightarrow \text{DFSM}$).

Derivation:

1. Characteristic finite-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 $G$
  
  $Q_c = lt_G$

- input alphabet are terminals and non-terminals
  
  $V_c = V_T \cup V_N$

- start state $q_c = [S' \rightarrow .S]$

- final states are the complete items
  
  $F_c = \{[X \rightarrow \alpha.] \mid X \rightarrow \alpha \in P\}$

- Transitions:
  
  $\Delta_c = \{([X \rightarrow \alpha.Y\beta], Y, [X \rightarrow \alpha Y.\beta]) \mid X \rightarrow \alpha Y\beta \in P \text{ and } Y \in V_N \cup V_T\}$

  $\cup \{([X \rightarrow \alpha.Y\beta], \varepsilon, [Y \rightarrow .\gamma]) \mid X \rightarrow \alpha Y\beta \in P \text{ and } Y \rightarrow \gamma \in P\}$
Item PDA and Characteristic NFA

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

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>New Stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[S' \rightarrow .S]$</td>
<td>$\epsilon$</td>
<td>$[S' \rightarrow .S][S \rightarrow .aSb]$</td>
</tr>
<tr>
<td>$[S' \rightarrow .S]$</td>
<td>$\epsilon$</td>
<td>$[S' \rightarrow .S][S \rightarrow .]$</td>
</tr>
<tr>
<td>$[S \rightarrow .aSb]$</td>
<td>$a$</td>
<td>$[S \rightarrow a.Sb]$</td>
</tr>
<tr>
<td>$[S \rightarrow a.Sb]$</td>
<td>$\epsilon$</td>
<td>$[S \rightarrow a.Sb][S \rightarrow .aSb]$</td>
</tr>
<tr>
<td>$[S \rightarrow a.Sb]$</td>
<td>$\epsilon$</td>
<td>$[S \rightarrow a.Sb][S \rightarrow .]$</td>
</tr>
<tr>
<td>$[S \rightarrow aS.b]$</td>
<td>$b$</td>
<td>$[S \rightarrow aSb.]$</td>
</tr>
<tr>
<td>$[S \rightarrow a.Sb][S \rightarrow .]$</td>
<td>$\epsilon$</td>
<td>$[S \rightarrow aS.b]$</td>
</tr>
<tr>
<td>$[S \rightarrow a.Sb][S \rightarrow aSb.]$</td>
<td>$\epsilon$</td>
<td>$[S \rightarrow aS.b]$</td>
</tr>
<tr>
<td>$[S' \rightarrow .][S \rightarrow aSb.]$</td>
<td>$\epsilon$</td>
<td>$[S' \rightarrow S.]$</td>
</tr>
<tr>
<td>$[S' \rightarrow .][S \rightarrow .]$</td>
<td>$\epsilon$</td>
<td>$[S' \rightarrow S.]$</td>
</tr>
</tbody>
</table>
Characteristic NFSM for $G_0$

$S \rightarrow E, \quad E \rightarrow E + T \mid T, \quad T \rightarrow T * F \mid F, \quad F \rightarrow (E) \mid \text{id}$
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 $P_G$:

$\varepsilon$–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 $[X \rightarrow \alpha.]$ from the stack,
- do a transition from the new topmost state under $X$,
- push the new state onto the stack.
Some Abbreviations:
RMD: rightmost derivation
RSF: right sentential form

Consider a RMD of cfg G:

\[ S' \xrightarrow{\ast} \beta Xu \xrightarrow{rm} \beta \alpha u \]

- \( \alpha \) is a handle of \( \beta \alpha u \).
  The part of a RSF next to be reduced.
- Each prefix of \( \beta \alpha \) is a viable 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 $G_0$

<table>
<thead>
<tr>
<th>RSF (handle)</th>
<th>viable prefix</th>
<th>Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E + F$</td>
<td>$E, E+, E + F$</td>
<td>$S \xrightarrow{rm} E \xrightarrow{rm} E + T \xrightarrow{rm} E + F$</td>
</tr>
<tr>
<td>$T * \text{id}$</td>
<td>$T, T*, T * \text{id}$</td>
<td>$S \xrightarrow{3 , rm} T * F \xrightarrow{rm} T * \text{id}$</td>
</tr>
<tr>
<td>$F * \text{id}$</td>
<td>$F$</td>
<td>$S \xrightarrow{4 , rm} T * \text{id} \xrightarrow{rm} F * \text{id}$</td>
</tr>
<tr>
<td>$T * \text{id} + \text{id}$</td>
<td>$T, T*, T * \text{id}$</td>
<td>$S \xrightarrow{3 , rm} T * F \xrightarrow{rm} T * \text{id}$</td>
</tr>
</tbody>
</table>
[\[X \rightarrow \alpha \cdot \beta\]] is valid for the viable prefix \(\gamma \alpha\), if there exists a RMD

\[ S' \xrightarrow{\ast} \gamma Xw \xrightarrow{rm} \gamma \alpha \beta w \]

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

Some viable prefixes of \(G_0\):

<table>
<thead>
<tr>
<th>Viable Prefix</th>
<th>Valid Items</th>
<th>Reason</th>
<th>(\gamma)</th>
<th>(w)</th>
<th>(X)</th>
<th>(\alpha)</th>
<th>(\beta)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(E+)</td>
<td>([E \rightarrow E + .T])</td>
<td>(S \xrightarrow{rm} E \xrightarrow{rm} E + T)</td>
<td>(\varepsilon)</td>
<td>(\varepsilon)</td>
<td>(E)</td>
<td>(E+)</td>
<td>(T)</td>
</tr>
<tr>
<td></td>
<td>([T \rightarrow .F])</td>
<td>(S \xrightarrow{rm} E + T \xrightarrow{rm} E + F)</td>
<td>(E+)</td>
<td>(\varepsilon)</td>
<td>(T)</td>
<td>(\varepsilon)</td>
<td>(F)</td>
</tr>
<tr>
<td></td>
<td>([F \rightarrow .id])</td>
<td>(S \xrightarrow{\ast} E + F \xrightarrow{rm} E + id)</td>
<td>(E+)</td>
<td>(\varepsilon)</td>
<td>(F)</td>
<td>(\varepsilon)</td>
<td>(id)</td>
</tr>
<tr>
<td>((E + (_E)))</td>
<td>([F \rightarrow (_E))]</td>
<td>(S \xrightarrow{\ast} (E + F) \xrightarrow{rm} (E + (E)))</td>
<td>((E+)</td>
<td>(_E)))</td>
<td>(F)</td>
<td>(_E)))</td>
<td>(E)</td>
</tr>
</tbody>
</table>
Given some input string $xuvw$.

The RMD

$$S' \xrightarrow{\ast} \gamma Xw \xrightarrow{rm} \gamma \alpha \beta w \xrightarrow{\ast} \gamma \alpha vw \xrightarrow{rm} \gamma uvw \xrightarrow{\ast} \gamma uvw \xrightarrow{rm} xuvw$$

describes the following sequence of partial derivations:

$$\gamma \xrightarrow{\ast} x \quad \alpha \xrightarrow{\ast} u \quad \beta \xrightarrow{\ast} v \quad X \xrightarrow{rm} \alpha \beta$$

$$S' \xrightarrow{\ast} \gamma Xw$$

performed by the bottom-up parser in this order.

The valid item $[X \rightarrow \alpha \cdot \beta]$ for the viable prefix $\gamma \alpha$ describes the situation after partial derivation 2, that is, for RSF $\gamma \alpha vw$
\[ ch(G) = (Q_c, V_c, \Delta_c, q_c, F_c) \]

**Theorem**

*For each viable 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* \( \gamma \) *is a viable prefix and* \( q \) *is a valid item for* \( \gamma \).*

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

**Theorem**

*The language of viable prefixes of a cfg is regular.*
Making $ch(G)$ deterministic

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

**Example:** $ch(G_{ab})$

$$
\begin{align*}
[S' \rightarrow . S] & \xrightarrow{S} [S' \rightarrow S.] \\
[S \rightarrow .aSb] & \xrightarrow{a} [S \rightarrow a.Sb] \\
[S \rightarrow .] & \xrightarrow{\epsilon} [S \rightarrow aS.b] \\
[S \rightarrow aS.b] & \xrightarrow{b} [S \rightarrow aSb.]
\end{align*}
$$

$LR_0(G_{ab})$: 

Characteristic NFSM for $G_0$

$$S \rightarrow E, \quad E \rightarrow E + T \mid T, \quad T \rightarrow T \ast F \mid F, \quad F \rightarrow (E) \mid \text{id}$$
\[ LR_0(G_0) \]

\[ S \rightarrow E, \quad E \rightarrow E + T \mid T, \quad T \rightarrow T \ast F \mid F, \quad F \rightarrow (E) \mid \text{id} \]
The States of $LR_0(G_0)$ as Sets of Items

$S_0 = \{ [S \rightarrow .E], [E \rightarrow .E + T], [E \rightarrow .T], [T \rightarrow .T * F], [T \rightarrow .F], [F \rightarrow .(E)], [F \rightarrow .id] \}$

$S_5 = \{ [F \rightarrow \text{id}] \}$

$S_6 = \{ [E \rightarrow E + .T], [T \rightarrow .T * F], [T \rightarrow .F], [F \rightarrow .(E)], [F \rightarrow .id] \}$

$S_0 = \{ [S \rightarrow E.], [E \rightarrow E. + T] \}$

$S_7 = \{ [T \rightarrow T * .F], [F \rightarrow .(E)], [F \rightarrow .id] \}$

$S_1 = \{ [S \rightarrow T.], [T \rightarrow T * F] \}$

$S_8 = \{ [F \rightarrow (E.)], [E \rightarrow E. + T] \}$

$S_2 = \{ [T \rightarrow F.], [E \rightarrow .E + T], [E \rightarrow .T], [T \rightarrow .T * F], [T \rightarrow .F], [F \rightarrow .(E)], [F \rightarrow .id] \}$

$S_9 = \{ [E \rightarrow E + T.], [T \rightarrow T * F] \}$

$S_3 = \{ [T \rightarrow F.], [E \rightarrow .E + T], [E \rightarrow .T], [T \rightarrow .T * F], [T \rightarrow .F], [F \rightarrow .(E)], [F \rightarrow .id] \}$

$S_10 = \{ [T \rightarrow T * F.] \}$

$S_4 = \{ [F \rightarrow (.E)], [E \rightarrow .E + T], [E \rightarrow .T], [T \rightarrow .T * F], [T \rightarrow .F], [F \rightarrow .(E)], [F \rightarrow .id] \}$

$S_11 = \{ [F \rightarrow (E).] \}$

$S_5 = \{ [F \rightarrow \text{id}] \}$

$S_6 = \{ [E \rightarrow E + .T], [T \rightarrow .T * F], [T \rightarrow .F], [F \rightarrow .(E)], [F \rightarrow .id] \}$
Theorems

\[ ch(G) = (Q_c, V_c, \Delta_c, q_c, F_c) \text{ and } LR_0(G) = (Q_d, V_N \cup V_T, \Delta, q_d, F_d) \]

<table>
<thead>
<tr>
<th>Theorem</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Theorem</strong></td>
</tr>
<tr>
<td><strong>Let</strong> ( \gamma ) <strong>be a viable prefix and</strong> ( p(\gamma) \in Q_d ) <strong>be the uniquely determined state, into which</strong> ( LR_0(G) ) <strong>transfers out of the initial state by reading</strong> ( \gamma ), <strong>i.e.,</strong> ( (q_d, \gamma) \xrightarrow{\ast}_{LR_0(G)} (p(\gamma), \varepsilon) ). <strong>Then</strong></td>
</tr>
<tr>
<td>(a) ( p(\varepsilon) = q_d )</td>
</tr>
<tr>
<td>(b) ( p(\gamma) = { q \in Q_c \mid (q_c, \gamma) \xrightarrow{\ast}_{ch(G)} (q, \varepsilon) } )</td>
</tr>
<tr>
<td>(c) ( p(\gamma) = { i \in lt_G \mid i \text{ valid for } \gamma } )</td>
</tr>
<tr>
<td>(d) <strong>Let</strong> ( \Gamma ) <strong>the (in general infinite) set of all viable prefixes of</strong> ( G ). <strong>The mapping</strong> ( p : \Gamma \rightarrow Q_d ) <strong>defines a finite partition on</strong> ( \Gamma ).</td>
</tr>
<tr>
<td>(e) ( L(LR_0(G)) ) <strong>is the set of viable prefixes of</strong> ( G ) <strong>that end in a handle.</strong></td>
</tr>
</tbody>
</table>
\( \gamma = E + F \) is a viable prefix of \( G_0 \). With the state \( p(\gamma) = S_3 \) are also associated:

- \( F, (F, (((F, \ldots \)
- \( T \ast (F, T \ast (((F, \ldots \)
- \( E + F, E + (F, E + ((F, \ldots \)

Consider \( S_6 \) in \( LR_0(G_0) \). It consists of all valid items for the viable prefix \( E + \), i.e., the items

\[
[E \rightarrow E + .T], [T \rightarrow .T \ast F], [T \rightarrow .F], [F \rightarrow .id], [F \rightarrow .(E)].
\]

Reason:

\( E + \) is prefix of the RSF \( E + T \);

\[
S \xrightarrow{rm} E \xrightarrow{rm} E + T \xrightarrow{rm} E + F \xrightarrow{rm} E + id
\]

\( \uparrow \) \( \uparrow \) \( \uparrow \) are

Therefore

\[
[E \rightarrow E + .T] \quad [T \rightarrow .F] \quad [F \rightarrow .id]
\]

are valid.
What the $LR_0(G)$ describes

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

- $\Gamma$ (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.
- $q_f = \{[S' \rightarrow S.]\}$ 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:** \((q, a, q \delta_d(q, a)) \in \Delta, \text{ if } \delta_d(q, a) \text{ defined.}\)

Read next input symbol \(a\) and push successor state of \(q\) under \(a\) (item \([X \rightarrow \cdots a \cdots] \in q\)).

**reduce:** \((q q_1 \ldots q_n, \varepsilon, q \delta_d(q, X)) \in \Delta, \text{ if } [X \rightarrow \alpha.] \in q_n, |\alpha| = n.\)

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 \(X \rightarrow \alpha \beta\) represented with item \([X \rightarrow \alpha.\beta]\).
Reduction in PDA $P_0(G)$
Some observations and recollections

- also works for reductions of $\epsilon$,
- each state has a unique entry symbol,
- the stack contents uniquely determine a viable prefix,
- current state (topmost) is the state associated with this viable prefix,
- current state consists of all items valid for this viable 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 $[X \rightarrow \alpha . a \beta]$ and at least one complete item $[Y \rightarrow \gamma.]$.

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

A state with a conflict is **inadequate**.
Some Inadequate States

\[ LR_0(G_0) \text{ has three inadequate states, } S_1, S_2 \text{ and } S_9. \]

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

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

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

- **LR(k) item** \([X \rightarrow \alpha_1.\alpha_2, L]\)
  
  if \(X \rightarrow \alpha_1\alpha_2 \in P\) and \(L \subseteq V_{T}^{\leq k}\)

- **LR(0) item** \([X \rightarrow \alpha_1.\alpha_2]\) is called **core** of \([X \rightarrow \alpha_1.\alpha_2, L]\)

- **lookahead set** \(L\) of \([X \rightarrow \alpha_1.\alpha_2, L]\)

- \([X \rightarrow \alpha_1.\alpha_2, L]\) is **valid** for a viable prefix \(\alpha \alpha_1\) if

\[
S' \# \xrightarrow{*} \alpha X w \xrightarrow{rm} \alpha \alpha_1 \alpha_2 w
\]

and

\[
L = \{ u \mid S' \# \xrightarrow{*} \alpha X w \xrightarrow{rm} \alpha \alpha_1 \alpha_2 w \text{ and } u = k : w \}\]

The context–free items can be regarded as LR(0)-items if

\([X \rightarrow \alpha_1.\alpha_2, \{\varepsilon\}]\) is identified with \([X \rightarrow \alpha_1.\alpha_2]\).
Example from $G_0$

1. $[E \rightarrow E + . T, \{\}, +, \#]$ is a valid LR(1)-item for $(E+$

2. $[E \rightarrow T., \{\ast\}]$ is not a valid LR(1)-item for any viable prefix

Reasons:

1. $S' \xrightarrow{\ast} (E) \xrightarrow{rm} (E + T) \xrightarrow{\ast} (E + T + \text{id})$ where

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

2. The string $E\ast$ can occur in no RMD.
LR–Parser

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

- the viable prefix $\gamma$ in the stack, actually the by $\gamma$ 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 ($X \rightarrow \alpha$)*: reduce by production $X \rightarrow \alpha$;
  - *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

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$V_{\lessapprox T#}^{\leq_k}$</th>
<th>$u$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>parser action for $(q, u)$</td>
<td></td>
</tr>
</tbody>
</table>

### Goto-table

<table>
<thead>
<tr>
<th>$Q$</th>
<th>$V_N \cup V_T$</th>
<th>$X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q$</td>
<td>$\delta_d(q, X)$</td>
<td></td>
</tr>
</tbody>
</table>
### Parser Table for $S \rightarrow aSb|\epsilon$

#### Action table

<table>
<thead>
<tr>
<th>state</th>
<th>sets of items</th>
<th>symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>{ [S' \rightarrow .S], [S \rightarrow .aSb], [S \rightarrow .] }</td>
<td>$s$</td>
</tr>
<tr>
<td>1</td>
<td>{ [S \rightarrow a.Sb], [S \rightarrow .aSb], [S \rightarrow .] }</td>
<td>$s$</td>
</tr>
<tr>
<td>2</td>
<td>{ [S \rightarrow aS.b] }</td>
<td>$s$</td>
</tr>
<tr>
<td>3</td>
<td>{ [S \rightarrow aSb.] }</td>
<td>$r(S \rightarrow aSb)$</td>
</tr>
<tr>
<td>4</td>
<td>{ [S' \rightarrow S.] }</td>
<td></td>
</tr>
</tbody>
</table>

#### Goto table

<table>
<thead>
<tr>
<th>state</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>state</th>
<th>symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>
## Parsing $aabb$

<table>
<thead>
<tr>
<th>Stack</th>
<th>Input</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0$</td>
<td>$aabb$</td>
<td>shift 1</td>
</tr>
<tr>
<td>$01$</td>
<td>$abb$</td>
<td>shift 1</td>
</tr>
<tr>
<td>$011$</td>
<td>$bb$</td>
<td>reduce $S \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$0112$</td>
<td>$bb$</td>
<td>shift 3</td>
</tr>
<tr>
<td>$01123$</td>
<td>$b$</td>
<td>reduce $S \rightarrow aSb$</td>
</tr>
<tr>
<td>$012$</td>
<td>$b$</td>
<td>shift 3</td>
</tr>
<tr>
<td>$0123$</td>
<td>$#$</td>
<td>reduce $S \rightarrow aSb$</td>
</tr>
<tr>
<td>$04$</td>
<td>$#$</td>
<td>accept</td>
</tr>
</tbody>
</table>
**Algorithm LR(1)–PARSER**

```plaintext
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 ($X \rightarrow \alpha$) : begin
            pop|\alpha|(S); push($S$, goto[top($S$), $X$]);
            output("$X \rightarrow \alpha$")
        end;
        accept: acc;
        error: err("...");
    end case
od
Set of LR(1)-items $I$ has a

**shift-reduce-conflict:**
if exists at least one item $[X \rightarrow \alpha.a\beta, L_1] \in I$
and at least one item $[Y \rightarrow \gamma., L_2] \in I$,
and if $a \in L_2$.

**reduce-reduce-conflict:**
if it contains at least two items $[X \rightarrow \alpha., L_1]$
and $[Y \rightarrow \beta., L_2]$ where $L_1 \cap L_2 \neq \emptyset$.

A state with a conflict is called **inadequate**.
Example from $G_0$

\[ S'_0 = \text{Closure(Start)} \]

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

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

\[ S'_6 = \text{Closure(Succ}(S'_1, +)) \]
\[ = \{ [E \rightarrow E + .T, \{\#, +\}], [T \rightarrow .T * F, \{\#, +, *\}], [T \rightarrow .F, \{\#, +, *\}], [F \rightarrow .(E), \{\#, +, *\}], [F \rightarrow .id, \{\#, +, *\}] \} \]

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

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

Idea: Use ambiguous grammar $G'_0$:

$$E \rightarrow E + E \mid E \times E \mid \text{id} \mid (E)$$

and operator precedence and associativity to disambiguate directly.
Deterministic $ch(G_0')$

...contains two states:

$$S_7 : E \rightarrow E + E.$$  
$$E \rightarrow E. + E$$  
$$E \rightarrow E. * E$$

$$S_8 : E \rightarrow E * E.$$  
$$E \rightarrow E. + E$$  
$$E \rightarrow E. * E$$

with shift reduce conflicts.

In both states, the parser can reduce or shift either $+$ or $\ast$. 
Consider the input $id + id \ast id$

and let the top of the stack be $S_7$.

- If reduce, then $+$ has higher precedence than $\ast$
- If shift, then $+$ has lower precedence than $\ast$

Consider the input $id + id + id$

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 implemented with loop and recursion:

```java
Expression parseExpression(Precedence precedence) {
    Expression expr = parsePrimary();
    for (; ;) {
        TokenKind kind = currToken.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);
    }
    return expr;
}
```