Compiler Construction
Lent Term 2021
Lecture 5 : Theoretical foundations of Bottom-up (LR) parsing

1. **This lecture develops a general theory for non-deterministic bottom-up parsing**
2. **Next lecture will present two techniques for imposing determinism --- SLR(1) parsing and LR(1) parsing.**

Timothy G. Griffin
tgg22@cam.ac.uk
Computer Laboratory
University of Cambridge

This grammar will be our running example

$$G_2 = (N_2, T_1, P_2, E')$$

$N_2 = \{\mathrm{E}', \mathrm{E}, \mathrm{T}, \mathrm{F}\}$ $T_1 = \{+, *, (,), \mathrm{id}\}$

$$
\begin{aligned}
P_2 : \mathrm{E}' &\rightarrow \mathrm{E} \\
\mathrm{E} &\rightarrow \mathrm{E} + \mathrm{T} \mid \mathrm{T} && \text{(expressions)} \\
\mathrm{T} &\rightarrow \mathrm{T} * \mathrm{F} \mid \mathrm{F} && \text{(terms)} \\
\mathrm{F} &\rightarrow (\mathrm{E}) \mid \mathrm{id} && \text{(factors)}
\end{aligned}
$$

Note: E’ was added for convenience to ensure that there is a single starting production.

Rightmost derivations

$$w \in T^* \qquad \alpha, \beta \in (N \cup T)^*$$

Given : $\alpha A w$ and a production $A \rightarrow \beta$

a rightmost derivation step is written as

$$\alpha A w \Rightarrow_{rm} \alpha \beta w$$

A rightmost derivation of *(x+y)*

$$
\begin{aligned}
E' &\Rightarrow_{rm} E \\
&\Rightarrow_{rm} T \\
&\Rightarrow_{rm} F \\
&\Rightarrow_{rm} (E) \\
&\Rightarrow_{rm} (E+T) \\
&\Rightarrow_{rm} (E+F) \\
&\Rightarrow_{rm} (E+y) \\
&\Rightarrow_{rm} (T+y) \\
&\Rightarrow_{rm} (F+y) \\
&\Rightarrow_{rm} (x+y)
\end{aligned}
$$

Top-down (LL) parsing is based on <u>left-most</u> derivations.

Bottom-up (LR) parsing is based on <u>right-most</u> derivations.

But Bottom-up parsers perform the derivation in reverse!

$$
\begin{aligned}
S &\Rightarrow_{rm} E \\
&\Rightarrow_{rm} T \\
&\Rightarrow_{rm} F \\
&\Rightarrow_{rm} (E) \\
&\Rightarrow_{rm} (E+T) \\
&\Rightarrow_{rm} (E+F) \\
&\Rightarrow_{rm} (E+y) \\
&\Rightarrow_{rm} (T+y) \\
&\Rightarrow_{rm} (F+y) \\
&\Rightarrow_{rm} (x+y)
\end{aligned}
$$

FLIP!

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow E'
\end{aligned}
$$

Start parse

Finish

Can we transform a backwards derivation into an execution of a stack machine?

$$
\begin{aligned}
(x+y) &\Leftarrow \\
(F+y) &\Leftarrow \\
(T+y) &\Leftarrow \\
(E+y) &\Leftarrow \\
(E+F) &\Leftarrow \\
(E+T) &\Leftarrow \\
(E) &\Leftarrow \\
F &\Leftarrow \\
T &\Leftarrow \\
E &\Leftarrow E'
\end{aligned}
$$

View the reversed derivation as a stack machine (use \$ as stack bottom and end-of-input).

Can we make this work?

stack	input
$\$$	$(x+y)\$$
$\$(F$	$+y)\$$
$\$(T$	$+y)\$$
$\$(E$	$+y)\$$
$\$(E+F$	$)\$$
$\$(E+T$	$)\$$
$\$(E)$	$\$$
$\$F$	$\$$
$\$T$	$\$$
$\$E$	$\$$
$\$E'$	$\$$

Let's try to formalize such a parser

An LR parser configuration has the form

$$\$\alpha, x\$$$

(α is the stack, x the remaining input)

The configuration is valid when there exists a right-most derivation of the form

$$S \Rightarrow^{*}_{rm} \alpha\, x$$

Let's try to formalize our (non-deterministic) parser

Suppose

$$\alpha A x \Rightarrow_{rm} \alpha\beta B z x$$

Our "backwards" parser MIGHT move

from one configuration to another like so:

$$\$\alpha\beta B z,\ x\$ \xrightarrow{reduce} \$\alpha A,\ x\$$$

This action is called a reduction

using production $A \rightarrow \beta B z$

Are reduction actions sufficient?

Suppose we have the derivation

$$\alpha A x \Rightarrow_{rm} \alpha\beta B z x \Rightarrow_{rm} \alpha\beta\gamma z x$$

using $A \to \beta B z$ and then $B \to \gamma$.

Simulating this in reverse, our parser gets stuck :

$$\$\alpha\beta\gamma,\ zx\$$$

$$\xrightarrow{reduce} \$\alpha\beta B,\ zx\$$$

$$\xrightarrow{???} ???$$

We want $\beta B z$ on top of the stack!

We need an action that <u>shifts</u> a terminal onto the stack!

$$\alpha A x \Rightarrow_{rm} \alpha\beta B z x \Rightarrow_{rm} \alpha\beta\gamma z x$$

$$\$\alpha\beta\gamma,\ zx\$$$

$$\xrightarrow{reduce} \$\alpha\beta B,\ zx\$$$

$$\xrightarrow{shift(s)} \$\alpha\beta B z,\ x\$$$

$$\xrightarrow{reduce} \$\alpha A,\ x\$$$

How do we know when to stop shifting? Here we don't want to gobble up *x*!

Sanity check.

Let's make sure that this can work when B does not appear in the right-hand side of A's production,

$$\alpha BxAz \Rightarrow_{rm} \alpha Bxyz \Rightarrow_{rm} \alpha\gamma xyz$$

using production $A \rightarrow y$, then $B \rightarrow \gamma$.

Our parser's possible actions:

$$\$\alpha\gamma,\ xyz\$$$

$$\xrightarrow{reduce} \$\alpha B,\ xyz\$$$

$$\xrightarrow{shift(s)} \$\alpha Bxy,\ z\$$$

$$\xrightarrow{reduce} \$\alpha BxA,\ z\$$$

All good! But again, how do we know when to reduce and when to stop shifting?

Shift and reduce are sufficient.

The previous two slides demonstrate that if we have a derivation

$$S \Rightarrow^{*}_{\mathrm{rm}} w$$

Then we can always "replay it" in reverse using shift/reduce actions

$$\$, w\$ \rightarrow^{*} \$S, \$$$

This tells us that shift and reduce are sufficient.

However, when we are parsing a *w* we won't have access to a derivation to replay! So our parser will be non-deterministic and GUESS what the future holds!

Replay parsing of *(x+y)* using shift/reduce actions. X=top-of-stack, a = next input token

stack	input	action[X, a]
$\$$	$(x+y)\$$	shift
$\$($	$x+y)\$$	shift
$\$(x$	$+y)\$$	reduce $F \to id$
$\$(F$	$+y)\$$	reduce $T \to F$
$\$(T$	$+y)\$$	reduce $E \to T$
$\$(E$	$+y)\$$	shift
$\$(E+$	$y)\$$	shift

... informal shift/reduce parse continued

stack	input	action[X, a]
$\$(E + y$	$)\$$	reduce $F \rightarrow id$
$\$(E + F$	$)\$$	reduce $T \rightarrow F$
$\$(E + T$	$)\$$	reduce $E \rightarrow E + T$
$\$(E$	$)\$$	shift
$\$(E)$	$\$$	reduce $F \rightarrow (E)$
$\$F$	$\$$	reduce $T \rightarrow F$
$\$T$	$\$$	reduce $F \rightarrow E$
$\$E$	$\$$	reduce $\mathrm{S} \rightarrow \mathrm{E}$
$\$E'$	$\$$	accept!

How do we decide when to shift and when to reduce?

Suppose $A \to \beta\gamma$ is a production. When our parser is in the configuration

$$\$\alpha\beta\gamma,\ x\$$$

we MIGHT want to reduce with $A \to \beta\gamma$.

However, if we have

$$\$\alpha\beta,\ x\$$$

we MIGHT want to continue parsing with the hope of eventually getting $\beta\gamma$ on top of the stack so that we can then reduce to A.

LR(0) items record how much of a production's right-hand side we have already parsed

For every grammar production

$$A \rightarrow \beta\gamma \qquad (\beta, \gamma \in (N \cup T)^*)$$

produce the LR(0) item

$$A \rightarrow \beta \bullet \gamma$$

Interpretation of $A \rightarrow \beta \bullet \gamma$: we have already parsed some input x derivable from β $(\beta \Rightarrow^*_{rm} x)$ and we MIGHT next see some input derivable from γ.

$LR(0)$ items for grammar G_2

$E' \rightarrow \bullet E$

$E \rightarrow \bullet E + T$

$E \rightarrow E \bullet + T$

$E \rightarrow E + \bullet T$

$E \rightarrow E + T \bullet$

$E \rightarrow \bullet T$

$E \rightarrow T \bullet$

$E' \rightarrow E \bullet$

$T \rightarrow \bullet T * T$

$T \rightarrow T \bullet * F$

$T \rightarrow T * \bullet F$

$T \rightarrow T * F \bullet$

$T \rightarrow \bullet F$

$T \rightarrow F \bullet$

$F \rightarrow \bullet (E)$

$F \rightarrow (\bullet E)$

$F \rightarrow (E \bullet)$

$F \rightarrow (E) \bullet$

$F \rightarrow \bullet id$

$F \rightarrow id \bullet$

Valid LR(0) items

Definition . Item $A \rightarrow \beta \bullet \gamma$ is valid for $\phi\beta$ if there exists a derivation

$$S \Rightarrow^{*}_{rm} \phi A x \Rightarrow_{rm} \phi\beta\gamma x$$

If item $A \rightarrow \beta \bullet \gamma$ is valid for $\phi\beta$ then our parser could use the item as a guide when in configurat ion

$$\$\phi\beta,\ z\$.$$

Suppose $A \rightarrow \beta B\gamma$ and $B \rightarrow \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_k$.

Consider the ways in which items for these productions might be used as parsing guides.

Derivation	Parse	Possible guides
S	$\$S, \$^*$	
$\Rightarrow^*_{rm} \phi A x$	$^*\!\leftarrow \$\phi A, x\$$	
$\Rightarrow_{rm} \phi\beta B\gamma x$	$\leftarrow \$\phi\beta B\gamma, x\$$	$A \rightarrow \beta B\gamma \bullet$
$\Rightarrow^*_{rm} \phi\beta B z x$	$^*\!\leftarrow \$\phi\beta B, zx\$$	$A \rightarrow \beta B \bullet \gamma$
$\Rightarrow_{rm} \phi\beta\alpha_i z x$	$\leftarrow \$\phi\beta\alpha_i, zx\$$	$B \rightarrow \alpha_i \bullet$
$\Rightarrow^*_{rm} \phi\beta u z x$	$^*\!\leftarrow \$\phi\beta, uzx\$$	$A \rightarrow \beta \bullet B\gamma, B \rightarrow \bullet\alpha_i$

Using items as parsing guides

Suppose our parser is in the config

$$\$\phi\beta,\ cz\$$$

and $A \to \beta \bullet c\gamma$ is valid for $\phi\beta$.

Then we MIGHT shift c onto the stack :

$$\$\phi\beta,\ cz\$ \xrightarrow{shift} \$\phi\beta c,\ z\$$$

Suppose our parser is in the config

$$\$\phi\beta,\ z\$$$

and $A \to \beta \bullet$ is valid for $\phi\beta$.

Then we MIGHT perform a reduction :

$$\$\phi\beta,\ z\$ \xrightarrow{reduce} \$\phi A,\ z\$$$

Using items as parsing guides

Suppose our parser is in the config

$$\$\phi\beta,\ z\$$$

which we will assume is valid, so $S \Rightarrow^*_{rm} \phi\beta z$.

Suppose $A \to \beta \bullet \gamma$ is valid for $\phi\beta$.

Then γ MIGHT capture the future of our parse (the past of that derivation). That is, it MIGHT be that

$$S \Rightarrow^*_{rm} \phi A x \Rightarrow_{rm} \phi\beta\gamma x \Rightarrow^*_{rm} \phi\beta y x = \phi\beta z$$

If so, our parser MIGHT proceed like so :

$$\$\phi\beta,\ z\$ = \$\phi\beta,\ yx\$ \to^* \$\phi\beta\gamma,\ x\$ \xrightarrow{reduce} \$\phi A,\ x\$.$$

That is, our parser could guess that γ will derive a prefix of the remaining input z.

The KEY idea in LR parsing

Augment our shift/reduce parser in such a way that in every configuration it can derive the set of all items valid for the the contents of the current stack.

Then at each step the parser can (non-deterministically) select an item from this set to use as a guide.

Defined a NFA with LR(0) items as states!

$$A \to \beta \bullet c\gamma \xrightarrow{\ c\ } A \to \beta c \bullet \gamma$$

$$A \to \beta \bullet B\gamma \xrightarrow{\ B\ } A \to \beta B \bullet \gamma$$

$$A \to \beta \bullet B\gamma \xrightarrow{\ \varepsilon\ } B \to \bullet \alpha_i$$

The initial state q_0 is this item constructed from the unique starting production

$E' \to \bullet E$ (for example)

and every item (state) is a final state.

Let δ_G be the transition function of this NFA.

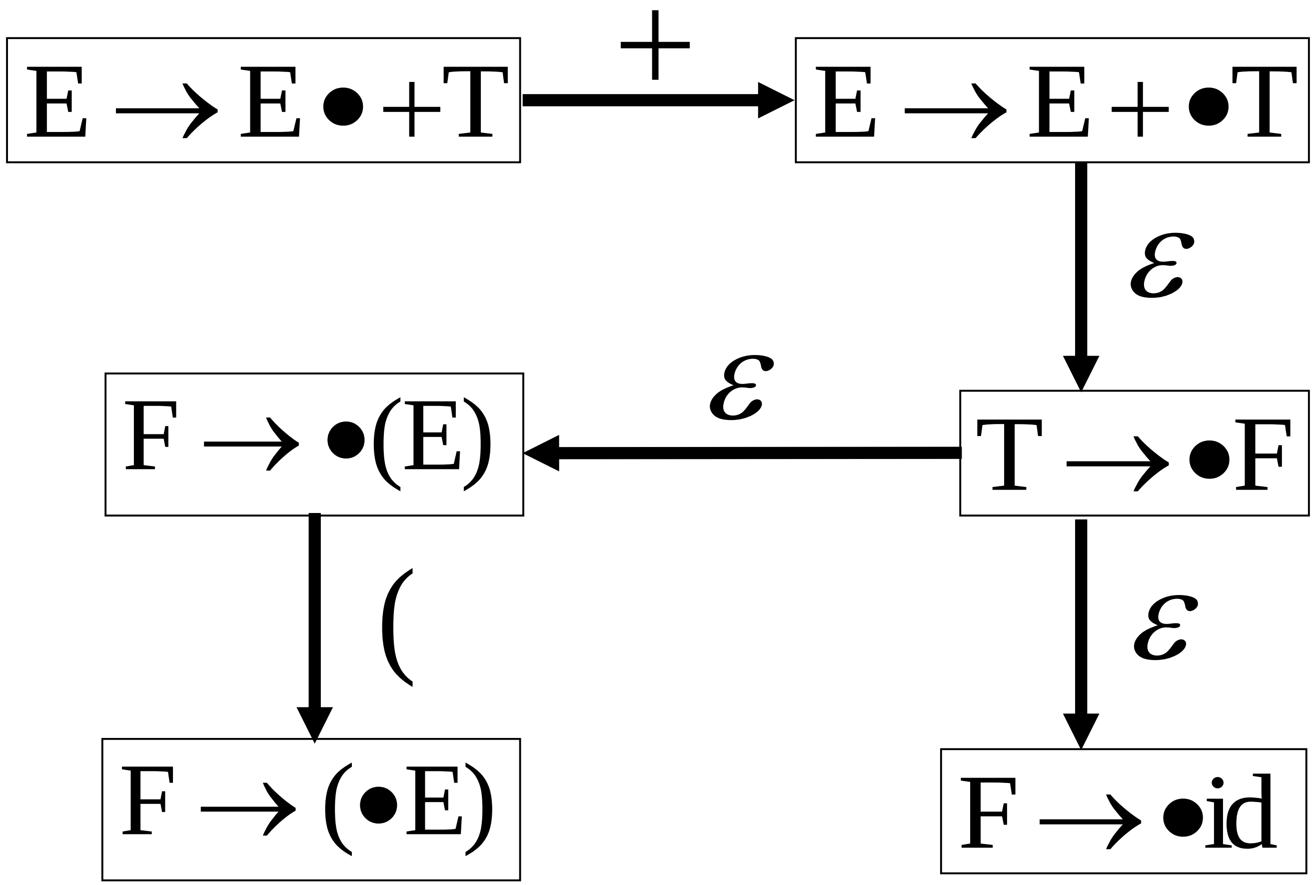
Main LR parsing theorem

Theorem. $A \to \beta \bullet \gamma \in \delta_G(q_0, \phi\beta)$ if and only if $A \to \beta \bullet \gamma$ is valid for $\phi\beta$.

Amazing fact : the language of the stack is regular!

See proof (not examinable) in Introduction to Automata Theory, Languages, and Computation. Hopcroft and Ullman.

A few NFA transitions for grammar G_2



A non-deterministic LR parsing algorithm

c := first symbol of input w$

while(true)

$\alpha :=$ the stack

if $A \rightarrow \beta \bullet c\gamma \in \delta_G(q_0, \alpha)$

then shift c onto the stack

c := next input token;

if $\mathrm{A} \rightarrow \beta \bullet \in \delta_G(q_0, \alpha)$

then reduce : pop β off the stack

and then push A onto the stack;

if $\mathrm{S} \rightarrow \beta \bullet \in \delta_G(q_0, \alpha)$

then accept and exit if no more input;

if none of the above then ERROR

This is non-deterministic since multiple conditions can be true and multiple items can match any condition.

How can we make the algorithm deterministic?

1. **The easy part: convert the NFA to a DFA**
2. **When there are shift/reduce or reduce/reduce conflicts, find some way of making a deterministic choice.**
3. **For (2), peek into the input buffer.**
4. **For (3), use FIRST and/or FOLLOW!**

Note : no matter how we do this there will be non-ambiguous grammars for which our deterministic parser will fail.

Next lecture : we will look at two popular approaches, SLR(1) and LR(1).