# Top-Down Syntax Analysis

Kristoffer H. Rose Eva Rose
*krisrose@cs.nyu.edu*

Compiler Construction
CSCI-GA.2130-001/Spring 2014
NYU Courant Institute

February 10, 2014

# Outline


1. Parsers
2. Context-Free Grammars
3. Grammar Transforms & HACS
4. Top-Down Parsing
5. HACS Parsing

# Second compilation phase

source program

Lexical Analysis

Tokens

Syntax Analysis

Tree

Semantic Analysis

Tree

Intermediate Representation Generator

IR

Optimizer

IR

Code Generator

IR

Machine-Dependent Code Optimizer

target machine code

Symbol Table

## Example

`position = initial + rate * 60`

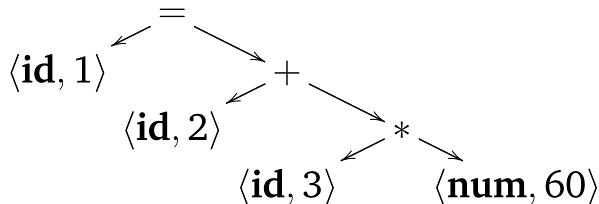
*scanned* into list of *tokens*:

$\langle \mathbf{id}, 1\rangle \ \langle = \rangle \ \langle \mathbf{id}, 2\rangle \ \langle + \rangle \ \langle \mathbf{id}, 3\rangle \ \langle * \rangle \ \langle \mathbf{num}, 60\rangle$

| | |
|---|---|
| 1 | `position` |
| 2 | `initial` |
| 3 | `rate` |

## Example

*parsed* into *parse/abstract syntax tree*:




according to precedence rules...

# Some syntax rules.

Some tree-structuring syntax rules:

- block indentation rules (Python)
- block delimiters like {...} (Java, C)
- grouping rules like ( ... ) (most languages)
- built-in algebraic precedence rules (most languages)
- statements vs expressions ...

# Outline


1 Parsers

2 Context-Free Grammars

3 Grammar Transforms & HACS

4 Top-Down Parsing

5 HACS Parsing

# A Left Recursive Grammar

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned} \tag{4.1}
$$

Bottom-Up Construction

# A Left Recursive Grammar

$$
\begin{aligned}
E &\rightarrow E + T \mid T \\
T &\rightarrow T * F \mid F \\
F &\rightarrow (\ E\ ) \mid \mathbf{id}
\end{aligned}
\tag{4.1}
$$

Bottom-Up Construction

# A Non-Left-Recursive Grammar

$$
\begin{aligned}
E &\to T\ E' \\
E' &\to \quad +\ T\ E' \mid \epsilon \\
T &\to F\ T' \\
T' &\to \quad *\ F\ T' \mid \epsilon \\
F &\to (\ E\ ) \mid \mathbf{id}
\end{aligned}
\tag{4.2}
$$

Top-Down Construction

# A Non-Left-Recursive Grammar

$$
\begin{aligned}
E &\to T\ E' \\
E' &\to \quad + \ T\ E' \mid \epsilon \\
T &\to F\ T' \\
T' &\to \quad * \ F\ T' \mid \epsilon \\
F &\to (\ E\ ) \mid \mathbf{id}
\end{aligned}
\tag{4.2}
$$

Top-Down Construction

# An Abstract Grammar

$$E \to E + E \mid E * E \mid (\ E\ ) \mid \textbf{id} \tag{4.3}$$

Good for *already constructed* trees!

# An Abstract Grammar

$$E \to E + E \mid E * E \mid ( E ) \mid \mathbf{id} \tag{4.3}$$

Good for *already constructed* trees!

# Errors

- Lexical.
- Syntactic.
- Semantic.
- Logical.

Detect early? Recover? Avoid repetition?

- Panic! (for a while, "skip to ;")
- Phrase-level.
- Error Productions.
- Global Correction.

# Errors

- Lexical.
- Syntactic.
- Semantic.
- Logical.

Detect early? Recover? Avoid repetition?

- Panic! (for a while, “skip to ;”)
- Phrase-level.
- Error Productions.
- Global Correction.

# Errors

- Lexical.
- Syntactic.
- Semantic.
- Logical.

Detect early? Recover? Avoid repetition?

- Panic! (for a while, "skip to ;")
- Phrase-level.
- Error Productions.
- Global Correction.

# Errors

- Lexical.
- Syntactic.
- Semantic.
- Logical.

Detect early? Recover? Avoid repetition?

- Panic! (for a while, "skip to ;")
- Phrase-level.
- Error Productions.
- Global Correction.

# Errors

- Lexical.
- Syntactic.
- Semantic.
- Logical.

Detect early? Recover? Avoid repetition?

- Panic! (for a while, “skip to ;”)
- Phrase-level.
- Error Productions.
- Global Correction.

# Outline


1. Parsers
2. Context-Free Grammars
3. Grammar Transforms & HACS
4. Top-Down Parsing
5. HACS Parsing

# Definition

## Context-free Grammar

$$G = (V, \Sigma, R, S)$$

where

- 'v $\in$ V' – finite set of *non-terminals*
- 'Σ' – finite set of *terminals* (or *alphabet*, where $\Sigma \cap V = \emptyset$)
- 'R' – finite relation of *productions, or rewrite rules* ($R \subseteq V \times (V \cup \Sigma)^*$)
- 'S' – *start symbol* ($S \in V$)

## Fig. 4.2

$$
\begin{aligned}
\mathit{expression} &\to \mathit{expression} + \mathit{term} \\
\mathit{expression} &\to \mathit{expression} - \mathit{term} \\
\mathit{expression} &\to \mathit{term} \\
\mathit{term} &\to \mathit{term} * \mathit{factor} \\
\mathit{term} &\to \mathit{term} \;/\; \mathit{factor} \\
\mathit{term} &\to \mathit{factor} \\
\mathit{factor} &\to (\ \mathit{expression}\ ) \\
\mathit{factor} &\to \mathbf{id}
\end{aligned}
$$

# Derivations

"Unfolding grammar until only tokens left."

Ambiguity = "More than one (left-/rightmost) derivation."

# Derivations

“Unfolding grammar until only tokens left.”

Ambiguity = “More than one (left-/rightmost) derivation.”

# Parse Tree

"Tracing unfolding grammar."

- Each unfolded nonterminal leaves a node.
- Each token a leaf.

# Parse Tree

"Tracing unfolding grammar."

- Each unfolded nonterminal leaves a node.
- Each token a leaf.

# CFG *vs* RE

"CFG can simulate any RE."

RE state = nonterminal.

RE transition = production $S \to aS'$.

RE accept = empty production.

RE start state = start symbol.

# CFG *vs* RE

"CFG can simulate any RE."

- RE state = nonterminal.
- RE transition = production $S \to tS'$.
- RE accept = empty production.
- RE start state = start symbol.

# CFG *vs* RE

"CFG can simulate any RE."

- RE state = nonterminal.
- RE transition = production $S \to tS'$.
- RE accept = empty production.
- RE start state = start symbol.

# CFG *vs* RE

"CFG can simulate any RE."

- RE state = nonterminal.
- RE transition = production $S \to tS'$.
- RE accept = empty production.
- RE start state = start symbol.

# CFG *vs* RE

"CFG can simulate any RE."

- RE state = nonterminal.
- RE transition = production $S \to tS'$.
- RE accept = empty production.
- RE start state = start symbol.

# Outline

1 Parsers

2 Context-Free Grammars

3 Grammar Transforms & HACS

4 Top-Down Parsing

5 HACS Parsing

## HACS Encoding

$$
\begin{array}{l}
\textbf{sort}\ E \mid [\![\langle E\rangle + \langle T\rangle]\!] \mid [\![\langle T\rangle]\!] \ ; \\
\textbf{sort}\ T \mid [\![\langle T\rangle * \langle F\rangle]\!] \mid [\![\langle F\rangle]\!] \ ; \\
\textbf{sort}\ F \mid [\![ \ (\langle E\rangle) \ ]\!] \mid [\![\langle Id\rangle]\!] \ ; \\
\textbf{token}\ Id \mid [a{-}z]+ \ ;
\end{array}
$$

- Explicit "$[\![\,]\!]$" for *concrete syntax*.
- Explicit "$\langle\rangle$" for *nonterminal and terminal references*.
- "sort" means “kind of parse tree node.”
- Immediate left recursion is alright.
- (All top level alternatives introduced with "|".)

# HACS Encoding

$$\textbf{sort}\ E\ |\ [\![\langle E\rangle + \langle T\rangle]\!]\ |\ [\![\langle T\rangle]\!]\ ;$$
$$\textbf{sort}\ T\ |\ [\![\langle T\rangle * \langle F\rangle]\!]\ |\ [\![\langle F\rangle]\!]\ ;$$
$$\textbf{sort}\ F\ |\ [\![\ (\langle E\rangle)\ ]\!]\ |\ [\![\langle Id\rangle]\!]\ ;$$
$$\textbf{token}\ Id\ |\ [a{-}z]+\ ;$$

- Explicit "⟦⟧" for *concrete syntax*.
- Explicit "⟨⟩" for *nonterminal and terminal references*.
- "sort" means “kind of parse tree node.”
- Immediate left recursion is alright.
- (All top level alternatives introduced with "|".)

# Eliminating Immediate Left Recursion

$$A \to A\alpha_1 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n \quad \Longrightarrow \quad \begin{array}{l} A \to \beta_1 A' \mid \cdots \mid \beta_n A' \\ A' \to \alpha_1 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$

```
sort E | ⟦⟨T⟩ ⟨E2⟩⟧ ; sort E2 | ⟦+ ⟨T⟩ ⟨E2⟩⟧ | ⟦⟧ ;
sort T | ⟦⟨F⟩ ⟨T2⟩⟧ ; sort T2 | ⟦* ⟨F⟩ ⟨T2⟩⟧ | ⟦⟧ ;
sort F | ⟦ (⟨E#1⟩) ⟧ | ⟦⟨Id⟩⟧ ;
token Id | [a–z]+ ;
```

# Eliminating Immediate Left Recursion

$$A \to A\alpha_1 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \cdots \mid \beta_n \quad \Longrightarrow \quad \begin{array}{l} A \to \beta_1 A' \mid \cdots \mid \beta_n A' \\ A' \to \alpha_1 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$

$$\begin{array}{l}
\textbf{sort } E \mid [\![\langle T\rangle\ \langle E2\rangle]\!]\ ;\ \textbf{sort } E2 \mid [\![+\ \langle T\rangle\ \langle E2\rangle]\!] \mid [\![]\!];\\
\textbf{sort } T \mid [\![\langle F\rangle\ \langle T2\rangle]\!]\ ;\ \textbf{sort } T2 \mid [\![*\ \langle F\rangle\ \langle T2\rangle]\!] \mid [\![]\!];\\
\textbf{sort } F \mid [\![\ (\langle E\#1\rangle)\ ]\!] \mid [\![\langle Id\rangle]\!]\ ;\\
\textbf{token } Id \mid [a\text{–}z]+\ ;
\end{array}$$

# Eliminating General Left Recursion

But what about

$$S \to A\ a$$
$$A \to A\ c \mid S\ d \mid \epsilon$$

Inline, then eliminate as before:

$$\begin{aligned} S &\to A\ a \\ A &\to A\ c \mid A\ a\ d \mid \epsilon \end{aligned} \implies \begin{aligned} S &\to A\ a \\ A &\to A' \\ A' &\to c\ A' \mid a\ d\ A' \mid \epsilon \end{aligned}$$

# Eliminating General Left Recursion

But what about

$$S \to A\,a$$
$$A \to A\,c \mid S\,d \mid \epsilon$$

Inline, then eliminate as before:

$$\begin{aligned} S &\to A\,a \\ A &\to A\,c \mid A\,a\,d \mid \epsilon \end{aligned} \implies \begin{aligned} S &\to A\,a \\ A &\to A' \\ A' &\to c\,A' \mid a\,d\,A' \mid \epsilon \end{aligned}$$

# Hint—Dumbing It Down

Sometimes you have

$$S \to A\ B \mid a\ C$$
$$A \to a \mid A\ b$$
$$C \to c$$

What to do with an $a$?

$$S \to A\ B \mid A\ C$$
$$A \to a \mid A\ b$$
$$C \to c$$

Kristoffer H. Rose Eva Rose*krisrose@cs.nyu.edu*

# Hint—Dumbing It Down

Sometimes you have

$$
\begin{aligned}
S &\to A\ B \mid a\ C \\
A &\to a \mid A\ b \\
C &\to c
\end{aligned}
$$

What to do with an $a$?

$$
\begin{aligned}
S &\to A\ B \mid A\ C \\
A &\to a \mid A\ b \\
C &\to c
\end{aligned}
$$

# Hint—Dumbing It Down

Sometimes you have

$$S \to A\ B \mid a\ C$$
$$A \to a \mid A\ b$$
$$C \to c$$

What to do with an $a$?

$$S \to A\ B \mid A\ C$$
$$A \to a \mid A\ b$$
$$C \to c$$

# Hint—Dumbing It Down

Sometimes you have

$$
\begin{aligned}
S &\to A\ B \mid a\ C \\
A &\to a \mid A\ b \\
C &\to c
\end{aligned}
$$

What to do with an $a$?

$$
\begin{aligned}
S &\to A\ B \mid {\color{red}A}\ C \\
A &\to a \mid A\ b \\
C &\to c
\end{aligned}
$$

# Hint—HACS Encoding with Precedence

$$
\begin{aligned}
\textbf{sort}\ E \quad & |\ [\![\langle E\rangle + \langle E@1\rangle]\!] \\
& |\ [\![\langle E@1\rangle * \langle E@2\rangle]\!]@1 \\
& |\ \textbf{sugar}\ [\![(\langle E\#1\rangle)\ ]\!]@2 \rightarrow E\#1 \\
& |\ [\![\langle Id\rangle]\!]\ ;
\end{aligned}
$$

- Single "sort" captures *abstract syntax tree.*
- *Precence* marker "@"*n* disambiguates parse.
- "sugar" specifies concrete-only syntax.

# Hint—HACS Encoding with Precedence

$$
\begin{aligned}
\textbf{sort}\ E\ \ &|\ [\![\langle E\rangle\ +\ \langle E@1\rangle]\!]\\
&|\ [\![\langle E@1\rangle\ *\ \langle E@2\rangle]\!]@1\\
&|\ \textbf{sugar}\ [\![(\langle E\#1\rangle)\ ]\!]@2 \rightarrow E\#1\\
&|\ [\![\langle Id\rangle]\!]\ ;
\end{aligned}
$$

- Single "sort" captures *abstract syntax tree*.
- *Precence* marker "@"*n* disambiguates parse.
- "sugar" specifies concrete-only syntax.

# Outline


1. Parsers
2. Context-Free Grammars
3. Grammar Transforms & HACS
4. Top-Down Parsing
5. HACS Parsing

# A Non-Left-Recursive Grammar

$$
\begin{aligned}
E &\to T\ E' \\
E' &\to \ + T\ E' \mid \epsilon \\
T &\to F\ T' \\
T' &\to \ * F\ T' \mid \epsilon \\
F &\to (\ E\ ) \mid \mathbf{id}
\end{aligned} \tag{4.2}
$$

# FIRST

1. If $X$ is *terminal*, $\text{FIRST}(X) = \{X\}$.
2. If $X$ is *non-terminal* with $X \to Y_1 \ldots Y_k$ $(k > 0)$, then $\text{FIRST}(X)$
   - includes every $a \in \text{FIRST}(Y_i)$ where $Y_1 \ldots Y_{i-1} \Rightarrow \epsilon$, and
   - includes $\epsilon$ if $\forall i = 1 \ldots k : \epsilon \in \text{FIRST}(Y_i)$.
3. if $X$ is *non-terminal* with $X \to \epsilon$ then $\epsilon \in \text{FIRST}(X)$.

# FOLLOW

1. $\$ \in \text{FOLLOW}(S)$.
2. For every production context $A \to \alpha B \beta$, $(\text{FIRST}(\beta) \setminus \{\epsilon\}) \subseteq \text{FOLLOW}(B)$; and
3. For every production context $A \to \alpha B$ or $A \to \alpha B \beta$ where $\epsilon \in \text{FIRST}(B)$, $\text{FOLLOW}(A) \subseteq \text{FOLLOW}(B)$.

## LL(1)—Left (to right) Leftmost (derivation) of 1 (input symbol)

Three requirements for productions $A \to \alpha \mid \beta$:

1. $\textsc{first}(\alpha)$ and $\textsc{first}(\beta)$ are disjoint (including for $\epsilon$).
2. If $\epsilon \in \textsc{first}(\alpha)$ then $\textsc{first}(\beta)$ and $\textsc{follow}(A)$ are disjoint.
3. If $\epsilon \in \textsc{first}(\beta)$ then $\textsc{first}(\alpha)$ and $\textsc{follow}(A)$ are disjoint.

# Predictive Parsing Table

Input: Grammar $G$.

Output: Parsing table $M$.

Method: For each production $A \to \alpha$ in the grammar:

1. For each $a \in \text{FIRST}(\alpha)$ add $A \to \alpha$ to $M[A, a]$.
2. If $\epsilon \in \text{FIRST}(\alpha)$ then for each $b \in \text{FOLLOW}(A)$ add $A \to \alpha$ to $M[A, b]$. (If $\epsilon \in \text{FIRST}(\alpha)$ and $\$ \in \text{FOLLOW}(A)$ then add $A \to \alpha$ to $M[A, \$]$.)

Any entries $M[A, a]$ that have no content are set to **error**.

$E \to T\,E' \quad E' \to +\,T\,E' \mid \epsilon \quad T \to F\,T' \quad T' \to *\,F\,T' \mid \epsilon \quad F \to$

# Predictive Parsing Table

Input: Grammar $G$.

Output: Parsing table $M$.

Method: For each production $A \to \alpha$ in the grammar:

1. For each $a \in \text{FIRST}(\alpha)$ add $A \to \alpha$ to $M[A, a]$.
2. If $\epsilon \in \text{FIRST}(\alpha)$ then for each $b \in \text{FOLLOW}(A)$ add $A \to \alpha$ to $M[A, b]$. (If $\epsilon \in \text{FIRST}(\alpha)$ and $\$ \in \text{FOLLOW}(A)$ then add $A \to \alpha$ to $M[A, \$]$.)

Any entries $M[A, a]$ that have no content are set to **error**.

$$E \to T\,E' \qquad E' \to +\,T\,E' \mid \epsilon \qquad T \to F\,T' \qquad T' \to *\,F\,T' \mid \epsilon \qquad F \to$$

# Parser Table

| Non-terminal | *InputSymbol* id | + | $*$ | ( | ) | \$ |
|---|---|---|---|---|---|---|
| $E$ | $E \to TE'$ | | | $E \to TE'$ | | |
| $E'$ | | $E' \to +TE'$ | | | $E' \to \epsilon$ | $E' \to \epsilon$ |
| $T$ | $T \to FT'$ | | | $T \to FT'$ | | |
| $T'$ | | $T' \to \epsilon$ | $T' \to *FT'$ | | $T' \to \epsilon$ | $T' \to \epsilon$ |
| $F$ | $F \to \mathbf{id}$ | | | $F \to (E)$ | | |

# Outline


1. Parsers
2. Context-Free Grammars
3. Grammar Transforms & HACS
4. Top-Down Parsing
5. HACS Parsing

# HACS Expression Parser

```
1  /* 2. SYNTAX ANALYSIS. */

2  sort Stat   | ⟦ ⟨Name⟩ := ⟨Exp⟩ ; ⟧ | ⟦ { ⟨Stat*⟩ } ⟧ ;

3  sort Exp    | ⟦ ⟨Exp@1⟩ + ⟨Exp@2⟩ ⟧@1
4              | ⟦ ⟨Exp@2⟩ * ⟨Exp@3⟩ ⟧@2
5              | ⟦ ⟨Int⟩ ⟧@3
6              | ⟦ ⟨Float⟩ ⟧@3
7              | ⟦ ⟨Name⟩ ⟧@3
8              | sugar ⟦ (⟨Exp#⟩) ⟧@3 → Exp# ;

9  sort Name   | symbol ⟦ ⟨Id⟩ ⟧ ;
```

## HACS Parser Generation Tricks

- Automatically does immediate left recursion elimination.
- Unusual brackets ⟦⟧⟨⟩. . .
- "sort" covers multiple non-terminals.
- Precedence handled automatically (the @ markers).
- Sugar.
- (Names – that's next week.)

Oh, and it is just a front-end for JavaCC.

# HACS Parser Generation Tricks

- Automatically does immediate left recursion elimination.
- Unusual brackets $[\![\,]\!]\langle\rangle\ldots$
- “sort” covers multiple non-terminals.
- Precedence handled automatically (the @ markers).
- Sugar.
- (Names – that’s next week.)

Oh, and it is just a front-end for JavaCC.

*Questions?*

krisrose@cs.nyu.edu