Programming Languages F28PL2, Lecture 2

Jamie Gabbay

February 2, 2014

Languages and formal grammars

Recall that a language is a set of symbols/tokens and a set of (possibly empty) strings of tokens.

We will let α, β, γ range over strings.

This is a computing course, so we need to think not only about what a language is, but also about how a language may be generated.

We generate languages using formal grammar. Using a formal grammar we can:

- Verify whether a sentence is in our language.
- Synthesise legal programs.

Terminology

- Write V for the set of symbols (V for 'vocabulary').
- We may partition (split) the set V into two subsets: of terminal and nonterminal symbols. (Why we do this will become clear later.)
- Write V^* for the set of all strings of elements of V (including the empty string). Call this the closure of V.
- Write V^+ for the set of non-empty strings of elements of V.
- Write ε for the empty string — often written informally as ' "" '.

If $V = \{a, b, r, c, d, r\}$ (the set containing a, b, r, c, d, and r), then is $abracadabra \in V^*$?

Is ε always in V^*? How about V^+?

Example

Suppose a vocabulary $V = \{0, 1, +, -, *, (,), \langle \text{exp} \rangle\}$.

Suppose $\langle \text{exp} \rangle$ is nonterminal and all the other symbols are terminal.

Example sentences in V are (just elements of V^*):

- ε, the empty string.
- $1 + 1$.
- $(1 + 1)$ and $(1 + (1))$.
- ((((and $(()) * --$.
- Is $(1 + 2 + \langle \text{exp} \rangle)$ in V^*?

Terminology

Recall that α, β, γ range over strings.

A production rule is a pair $\alpha ::= \beta$.

Suppose a vocabulary $V = \{0, 1, +, -, *, (,), \langle \mathsf{exp} \rangle\}$.

Example production rules are:

$$
\begin{aligned}
\langle \mathsf{exp} \rangle &::= 0 \\
\langle \mathsf{exp} \rangle &::= 1 \\
\langle \mathsf{exp} \rangle &::= -\langle \mathsf{exp} \rangle \\
\langle \mathsf{exp} \rangle &::= (\langle \mathsf{exp} \rangle) \\
\langle \mathsf{exp} \rangle &::= \langle \mathsf{exp} \rangle + \langle \mathsf{exp} \rangle \\
\langle \mathsf{exp} \rangle &::= \langle \mathsf{exp} \rangle * \langle \mathsf{exp} \rangle
\end{aligned}
$$

Production rules

We write a sequence

$$\alpha ::= \beta_1, \ldots, \alpha ::= \beta_n \quad \text{as just}$$
$$\alpha ::= \beta_1 \mid \ldots \mid \beta_n.$$

For example:

$$\begin{aligned}
\langle\text{exp}\rangle &::= 0 \\
\langle\text{exp}\rangle &::= 1 \\
\langle\text{exp}\rangle &::= -\langle\text{exp}\rangle \\
\langle\text{exp}\rangle &::= (\langle\text{exp}\rangle) \\
\langle\text{exp}\rangle &::= \langle\text{exp}\rangle+\langle\text{exp}\rangle \\
\langle\text{exp}\rangle &::= \langle\text{exp}\rangle*\langle\text{exp}\rangle
\end{aligned}
\quad \text{becomes} \quad
\begin{aligned}
\langle\text{exp}\rangle &::= 0 \mid 1 \mid \langle\text{exp}\rangle+\langle\text{exp}\rangle \mid \\
&\qquad -\langle\text{exp}\rangle \mid \langle\text{exp}\rangle*\langle\text{exp}\rangle \mid (\langle\text{exp}\rangle)
\end{aligned}$$

Production rules

We can use production rules to produce sentences. For example:

$$
\begin{aligned}
\langle \mathsf{exp} \rangle &\Rightarrow -\langle \mathsf{exp} \rangle \\
&\Rightarrow -(\langle \mathsf{exp} \rangle) \\
&\Rightarrow -(\langle \mathsf{exp} \rangle + \langle \mathsf{exp} \rangle) \\
&\Rightarrow -(1 + \langle \mathsf{exp} \rangle) \\
&\Rightarrow -(1+1)
\end{aligned}
\qquad\qquad
\begin{aligned}
\langle \mathsf{exp} \rangle &\Rightarrow \langle \mathsf{exp} \rangle + \langle \mathsf{exp} \rangle \\
&\Rightarrow 1 + \langle \mathsf{exp} \rangle \\
&\Rightarrow 1 + \langle \mathsf{exp} \rangle * \langle \mathsf{exp} \rangle \\
&\Rightarrow 1 + 0 * \langle \mathsf{exp} \rangle \\
&\Rightarrow 1 + 0 * 1
\end{aligned}
$$

So, starting from the nonterminal $\langle \mathsf{exp} \rangle$, we can generate many different sentences.

Grammars

Formally, a grammar is a 4-tuple of:

- $\mathbb{N}$ a set of nonterminal symbols.
- $\mathbb{T}$ a set of terminal symbols, disjoint from $\mathbb{N}$.
- A start symbol, in $\mathbb{N}$.
- A set of productions $\alpha ::= \beta$.

Notational conventions

Some important notational conventions which you are required to just know:

$A, B, C, S, T, \langle\text{exp}\rangle, \ldots$ range over nonterminals ($\mathbb{N}$).

$a, b, c, \ldots$ range over terminals ($\mathbb{T}$).

We call $\mathbb{N} \cup \mathbb{T}$ a vocabulary. X, Y, Z range over $\mathbb{N} \cup \mathbb{T}$.

Strings of terminals: x, y, z

Strings of terminals and/or nonterminals: $\alpha, \beta, \gamma, \ldots$

Terminology

The object-language is a language, defined as the set of strings of terminals that we can produce using the production rules, starting from the start symbol.

The meta-language is the language, defined as the set of all strings of terminals or nonterminals that we can produce using the production rules, starting from the start symbol.

The meta-language contains sentences of the object-language, but it may also contain extra sentences.

Production rules

What were the terminals and non-terminals implicit in the example production rules considered previously?

What was the start symbol?

Example grammars

$$\begin{aligned}\langle \text{exp} \rangle \ &::= 0 \mid 1 \mid \langle \text{exp} \rangle + \langle \text{exp} \rangle \mid \\ &\quad - \langle \text{exp} \rangle \mid \langle \text{exp} \rangle * \langle \text{exp} \rangle \mid (\langle \text{exp} \rangle) \end{aligned} \qquad \text{Start symbol: } \langle \text{exp} \rangle$$

$$S ::= ab \mid aSb \qquad \text{Start symbol: } S$$

$$S ::= aS \mid aT \qquad \text{Start symbol: } S$$

$$T ::= b \mid bT \qquad \text{Start symbol: } T$$

This is generative grammar. Let's generate a sentence using the second example:

$$\begin{aligned} S &\Rightarrow aSb \\ &\Rightarrow aaSbb \\ &\Rightarrow aaabbb \end{aligned}$$

Chomsky classification of grammars

Type 0 grammars contain productions of the form

$$\alpha ::= \beta.$$

α is a non-empty string of terminal and/or nonterminal symbols.

Type 0 grammars include pretty much anything.

Type 1 or context-sensitive grammars contain productions of the form

$$\alpha A \gamma ::= \alpha \beta \gamma.$$

Here A denotes a single nonterminal and β denotes an arbitrary string of terminal and/or nonterminal symbols. You can 'expand' A — subject to it occurring in the context described by α and γ.

Production rules

Things get more restrictive:

Type 2 or context-free grammars contain productions of the form

$$A ::= \gamma.$$

A denotes a single nonterminal. BNF is a language for describing Type 2 languages.

Type 3 or regular grammars contain productions of the form

$$\begin{aligned} A &::= aB \\ A &::= b \\ A &::= \epsilon. \end{aligned}$$

See also regular expressions.

Production rules

Type 3 grammars are good for identifying lexical units such as words; for instance “alphanumeric strings” or “numbers, possibly with underscores”.

Type 2 grammars are good for languages like “the language of arithmetic” or “Mary loves John”.

Most of the computer languages you know are determined by type 2 grammars (if $\langle$bool$\rangle$ then $\langle$exp$\rangle$ else $\langle$exp$\rangle$); the keywords of those languages are determined by type 3 grammars (if, then, and else).

Derivations

A little notation is useful:

$\alpha \Rightarrow \beta$ means 'β derived from α by some production'.

$\alpha \overset{p}{\Rightarrow} \beta$ means 'β derived from α by production p'.

$\alpha \overset{*}{\Rightarrow} \beta$ means 'β derived from α by zero or more productions'.

$\alpha \overset{+}{\Rightarrow} \beta$ means 'β derived from α by one or more productions'.

A type 2 (context-free) language

The language is

$$\mathbb{L} = \{a^n b^n \mid n \geq 1\}.$$

A grammar for it is

$$S ::= ab \mid aSb,$$

the start symbol is S.

Let's derive a sentence:

$$\begin{aligned} S &\Rightarrow aSb \\ &\Rightarrow aaSbb \\ &\Rightarrow aaabbb \end{aligned}$$

Note: supports balanced bracketing!

A type 3 (regular) grammar

The language is

$$\mathbb{L} = \{a^p b^q \mid p{\geq}1, q{\geq}1\}.$$

A grammar for it is

$$S ::= aS \mid aT \qquad T ::= b \mid bT.$$

The start symbol is S.

Let's derive a sentence:

$$S \Rightarrow aS \Rightarrow aaT \Rightarrow aabT \Rightarrow aabbT \Rightarrow aabbb.$$

Note: does not support balanced bracketing.

Production rules

Suppose we want to know whether a sentence α in language $\mathbb{L}$?

One algorithm to decide this is to try to generate it by applying all possible production rules in all possible orders.

For example is $-(\mathsf{id} + \mathsf{id})$ in the language determined by this grammar:

$$\begin{array}{rcl} \langle \mathsf{exp} \rangle & ::= & \langle \mathsf{exp} \rangle + \langle \mathsf{exp} \rangle \mid \langle \mathsf{exp} \rangle * \langle \mathsf{exp} \rangle \mid \\ & & (\langle \mathsf{exp} \rangle) \mid -\langle \mathsf{exp} \rangle \mid \mathsf{id} \end{array}$$

Production rules

Yes:

$$\begin{aligned}
\langle \text{exp} \rangle &\Rightarrow -\langle \text{exp} \rangle \\
&\Rightarrow -(\langle \text{exp} \rangle) \\
&\Rightarrow -(\langle \text{exp} \rangle + \langle \text{exp} \rangle) \\
&\Rightarrow -(\text{id} + \langle \text{exp} \rangle) \\
&\Rightarrow -(\text{id} + \text{id})
\end{aligned}$$

This is immensely inefficient! I am only claiming that this algorithm works in principle.

More on efficiency later.

More terminology you need to know

Phrase: a string derived from a nonterminal other than the start symbol.

Sentential form: a string derived from the start symbol.

Sentence: a sentential form without nonterminals.

How do we apply productions to form phrases, sentential forms, or sentences?

Leftmost derivation: a derivation where always the leftmost nonterminal is replaced. Gives rise to leftmost sentential form.

Rightmost derivation: a derivation where always the rightmost nonterminal is replaced. Gives rise to rightmost sentential form.

Leftmost derivation of $-(\text{id} + \text{id})$

$$
\begin{array}{rcl}
\langle \text{exp} \rangle & \Rightarrow & -\langle \text{exp} \rangle \\
& \Rightarrow & -(\langle \text{exp} \rangle) \\
& \Rightarrow & -(\langle \text{exp} \rangle + \langle \text{exp} \rangle) \\
& \Rightarrow & -(\text{id} + \langle \text{exp} \rangle) \\
& \Rightarrow & -(\text{id} + \text{id})
\end{array}
$$

$-(\text{id}+\text{id})$ is a sentential form, a sentence, and a leftmost sentential form.

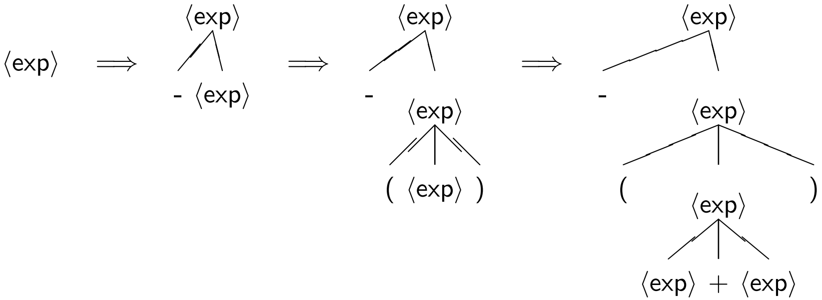
Rightmost derivation of $-(\mathsf{id} + \mathsf{id})$

$$
\begin{array}{rcl}
\langle \mathsf{exp} \rangle & \Rightarrow & -\langle \mathsf{exp} \rangle \\
& \Rightarrow & -(\langle \mathsf{exp} \rangle) \\
& \Rightarrow & -(\langle \mathsf{exp} \rangle + \langle \mathsf{exp} \rangle) \\
& \Rightarrow & -(\langle \mathsf{exp} \rangle + \mathsf{id}) \\
& \Rightarrow & -(\mathsf{id} + \mathsf{id})
\end{array}
$$

As it happens, $-(\mathsf{id}+\mathsf{id})$ is also a rightmost sentential form.

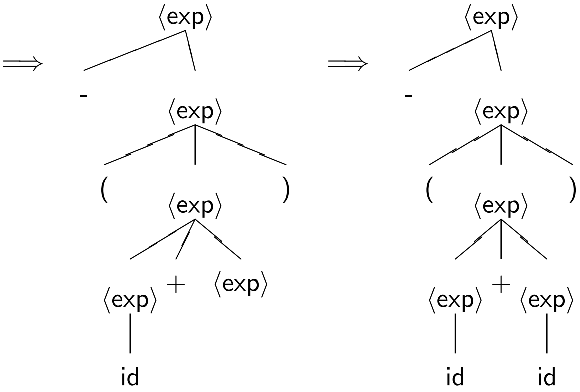
Parse trees and derivations. . .

Parse trees remember how a sentence was produced.



. . . just a bit more

The parse tree on the far right represents both leftmost and rightmost derivations given previously.



Different grammars

Two different grammars can define the same language $\mathbb{L}$.

Call two grammars equivalent when they describe the same language.

However, equivalent grammars can define different parse trees.

Two grammars

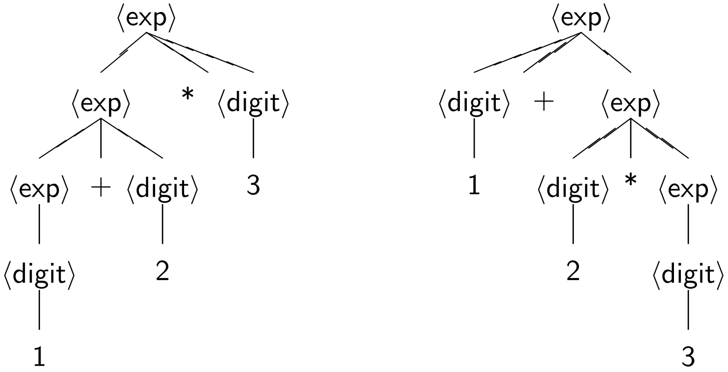
Grammar 1:

$$\begin{array}{rcl} \langle\mathsf{exp}\rangle & ::= & \langle\mathsf{digit}\rangle \mid \langle\mathsf{exp}\rangle{+}\langle\mathsf{digit}\rangle \mid \langle\mathsf{exp}\rangle{*}\langle\mathsf{digit}\rangle \\ \langle\mathsf{digit}\rangle & ::= & 1 \mid 2 \mid 3 \end{array}$$

Grammar 2:

$$\begin{array}{rcl} \langle\mathsf{exp}\rangle & ::= & \langle\mathsf{digit}\rangle \mid \langle\mathsf{digit}\rangle{+}\langle\mathsf{exp}\rangle \mid \langle\mathsf{digit}\rangle{*}\langle\mathsf{exp}\rangle \\ \langle\mathsf{digit}\rangle & ::= & 1 \mid 2 \mid 3 \end{array}$$

Different parse trees



Different parse trees

This is important, because different parse trees may induce different intuitive meanings.

Programs are not just syntax: we write a program because we give it meaning.

That meaning can be influenced by how we parse the string.

- The tree on the left intuitively means 9.
- The tree on the right intuitively means 7.