

## Chapter 4: Context-free Languages

In this chapter, we study context-free grammars and languages. Context-free grammars are used to describe the syntax of programming languages, i.e., to specify parsers of programming languages.

A language is called context-free iff it is generated by a context-free grammar. It will turn out that the set of all context-free languages is a proper superset of the set of all regular languages. On the other hand, the context-free languages have weaker closure properties than the regular languages, and we won't be able to give algorithms for checking grammar equivalence or minimizing the size of grammars.

---

Copyright © 2003–4 Alley Stoughton

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

The  $\text{\LaTeX}$  source of these slides, the associated book, and the distribution of the Forlan toolset are available on the WWW at <http://people.cis.ksu.edu/~stough/forlan/>.

1

### Section 4.1: Grammars, Parse Trees and Context-free Languages

In this section, we:

- say what (context-free) grammars are;
- use the notion of a parse tree to say what grammars mean;
- say what it means for a language to be context-free.

2

## (4.1) (Context-free) Grammars

A *context-free grammar* (CFG, or just grammar)  $G$  consists of:

- a finite set  $Q_G$  of symbols (we call the elements of  $Q_G$  the *variables* of  $G$ );
- an element  $s_G$  of  $Q_G$  (we call  $s_G$  the *start variable* of  $G$ );
- a finite subset  $P_G$  of  $\{(q, x) \mid q \in Q_G \text{ and } x \in \mathbf{Str}\}$  (we call the elements of  $P_G$  the *productions* of  $G$ ).

In a context where we are only referring to a single CFG,  $G$ , we sometimes abbreviate  $Q_G$ ,  $s_G$  and  $P_G$  to  $Q$ ,  $s$  and  $P$ , respectively. Whenever possible, we will use the mathematical variables  $p$ ,  $q$  and  $r$  to name variables.

We write **Gram** for the set of all grammars. Since every grammar can be described by a finite sequence of ASCII characters, we have that **Gram** is countably infinite.

3

## (4.1) Example Grammar

As an example, we can define a CFG  $G$  (of arithmetic expressions) as follows:

- $Q_G = \{E\}$ ;
- $s_G = E$ ;
- $P_G = \{(E, E\langle\text{plus}\rangle E), (E, E\langle\text{times}\rangle E), (E, \langle\text{openPar}\rangle E \langle\text{closPar}\rangle), (E, \langle\text{id}\rangle)\}$ .

E.g., we can read the production  $(E, E\langle\text{plus}\rangle E)$  as “an expression can consist of an expression, followed by a  $\langle\text{plus}\rangle$  symbol, followed by an expression”.

4

## (4.1) Notation for Grammars

We typically describe a grammar by listing its productions, writing a production  $(q, x)$  as  $q \rightarrow x$ , and grouping productions with identical left-sides into production families. Unless we say otherwise, the grammar's variables are the left-sides of all of its productions, and its start variable is the left-side of its first production.

Thus, our grammar  $G$  is

$$\begin{aligned} E &\rightarrow E\langle\text{plus}\rangle E, \\ E &\rightarrow E\langle\text{times}\rangle E, \\ E &\rightarrow \langle\text{openPar}\rangle E\langle\text{closPar}\rangle, \\ E &\rightarrow \langle\text{id}\rangle, \end{aligned}$$

or

$$E \rightarrow E\langle\text{plus}\rangle E \mid E\langle\text{times}\rangle E \mid \langle\text{openPar}\rangle E\langle\text{closPar}\rangle \mid \langle\text{id}\rangle.$$

5

## (4.1) Forlan Syntax for Grammars

The Forlan syntax for grammars is very similar to the notation of the preceding slide. E.g., here is how our example grammar can be described in Forlan's syntax:

```
{variables}
E
{start variable}
E
{productions}
E -> E<plus>E | E<times>E | <openPar>E<closPar> | <id>
```

Production families are separated by semicolons.

6

## (4.1) Processing CFGs in Forlan

The Forlan module `Gram` defines an abstract type `gram` (in the top-level environment) of grammars as well as a number of functions and constants for processing grammars, including:

```
val input      : string -> gram
val output     : string * gram -> unit
val numVariables : gram -> int
val numProductions : gram -> int
val equal      : gram * gram -> bool
```

7

## (4.1) More on Grammars

The *alphabet* of a grammar  $G$  ( $\mathbf{alphabet}(G)$ ) is

$$\{a \in \mathbf{Sym} \mid \text{there are } q, x \text{ such that } (q, x) \in P_G \text{ and } a \in \mathbf{alphabet}(x)\}$$

–  $Q_G$ .

I.e.,  $\mathbf{alphabet}(G)$  is all of the symbols appearing in the strings of  $G$ 's productions that aren't variables.

For example, the alphabet of our example grammar  $G$  is  $\{\langle \text{plus} \rangle, \langle \text{times} \rangle, \langle \text{openPar} \rangle, \langle \text{closPar} \rangle, \langle \text{id} \rangle\}$ .

8

## (4.1) Grammar Alphabets in Forlan

The Forlan module `Gram` defines a function

```
val alphabet : gram -> sym set
```

for calculating the alphabet of a grammar.

E.g., if `gram` of type `gram` is bound to our example grammar  $G$ , then Forlan will behave as follows:

```
- val bs = Gram.alphabet gram;
val bs = - : sym set
- SymSet.output("", bs);
<id>, <plus>, <times>, <closPar>, <openPar>
val it = () : unit
```

9

## (4.1) Parse Trees

We will explain when strings are generated by grammars using the notion of a parse tree. The set **PT** of *parse trees* is the least subset of  $\mathbf{Trees}_{\mathbf{Sym} \cup \{\%\}}$  (the set of all  $(\mathbf{Sym} \cup \{\%\})$ -trees; see Section 1.3) such that:

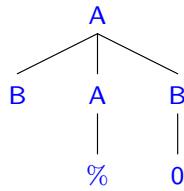
- (1) for all  $a \in \mathbf{Sym}$ ,  $n \in \mathbb{N}$  and  $pt_1, \dots, pt_n \in \mathbf{PT}$ ,  
 $a(pt_1, \dots, pt_n) \in \mathbf{PT}$ ;
- (2) for all  $q \in \mathbf{Sym}$ ,  $q(\%) \in \mathbf{PT}$ .

Since  $n$  is allowed to be 0 in rule (1), for every symbol  $a$ , we have that  $a()$  is a parse tree, which we normally abbreviate to  $a$ . On the other hand,  $\% = \%() \notin \mathbf{PT}$ . In rule (2),  $q(\%)$  abbreviates  $q(\%())$ .

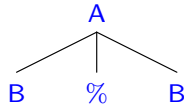
It is easy to see that **PT** is countably infinite.

## (4.1) Parse Tree Examples

For example,  $A(B, A(\%), B(0))$ , i.e.,



is a parse tree. On the other hand, although  $A(B, \%, B)$ , i.e.,



is a  $(\mathbf{Sym} \cup \{\%\})$ -tree, it's not a parse tree, since it can't be formed using rules (1) and (2).

11

## (4.1) Induction Principle for **PT**

Since the set **PT** of parse trees is defined inductively, it gives rise to an induction principle. The *principle of induction on **PT*** says that

for all  $pt \in \mathbf{PT}$ ,  $P(pt)$

follows from showing

(1) for all  $a \in \mathbf{Sym}$ ,  $n \in \mathbb{N}$  and  $pt_1, \dots, pt_n \in \mathbf{PT}$ , if

$P(pt_1), \dots, P(pt_n)$ ,

then

$P(a(pt_1, \dots, pt_n))$ ;

(2) for all  $q \in \mathbf{Sym}$ ,

$P(q(\%))$ .

12

## (4.1) The Yield of a Parse Tree

We define the yield of a parse tree, as follows. The function  $\mathbf{yield} \in \mathbf{PT} \rightarrow \mathbf{Str}$  is defined by recursion:

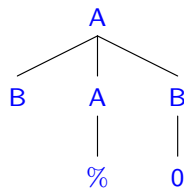
- for all  $a \in \mathbf{Sym}$ ,  $\mathbf{yield}(a) = a$ ;
- for all  $q \in \mathbf{Sym}$ ,  $n \in \mathbb{N} - \{0\}$  and  $pt_1, \dots, pt_n \in \mathbf{PT}$ ,  
$$\mathbf{yield}(q(pt_1, \dots, pt_n)) = \mathbf{yield}(pt_1) \cdots \mathbf{yield}(pt_n)$$
;
- for all  $q \in \mathbf{Sym}$ ,  $\mathbf{yield}(q(\%)) = \%$ .

We say that  $w$  is the *yield* of  $pt$  iff  $w = \mathbf{yield}(pt)$ .

13

## (4.1) Yield Example

For example, the yield of



is

$$\mathbf{yield}(B) \mathbf{yield}(A(\%)) \mathbf{yield}(B(0)) = B\% \mathbf{yield}(0) = B\%0 = B0.$$

14

## (4.1) Validity of Parse Trees for Grammars

We say when a parse tree is valid for a grammar  $G$  as follows. Define a function  $\text{valid}_G \in \mathbf{PT} \rightarrow \{\text{true}, \text{false}\}$  by recursion:

- for all  $a \in \mathbf{Sym}$ ,  $\text{valid}_G(a) = a \in \text{alphabet}(G)$  or  $a \in Q_G$ ;
- for all  $q \in \mathbf{Sym}$ ,  $n \in \mathbb{N} - \{0\}$  and  $pt_1, \dots, pt_n \in \mathbf{PT}$ ,

$$\begin{aligned} & \text{valid}_G(q(pt_1, \dots, pt_n)) \\ &= (q, \text{rootLabel}(pt_1) \cdots \text{rootLabel}(pt_n)) \in P_G \text{ and} \\ & \text{valid}_G(pt_1) \text{ and } \cdots \text{ and } \text{valid}_G(pt_n); \end{aligned}$$

- for all  $q \in \mathbf{Sym}$ ,  $\text{valid}_G(q(\%)) = (q, \%) \in P_G$ .

We say that  $pt$  is *valid for  $G$*  iff  $\text{valid}_G(pt) = \text{true}$ . We sometimes abbreviate  $\text{valid}_G$  to **valid**.

15

## (4.1) Parse Tree Validity Examples

Suppose  $G$  is the grammar

$$A \rightarrow BAB \mid \%,$$

$$B \rightarrow 0$$

(by convention, its variables are  $A$  and  $B$  and its start variable is  $A$ ).

Let's see why the parse tree  $A(B, A(\%), B(0))$  is valid for  $G$ .

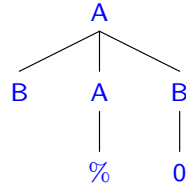
- Since  $A \rightarrow BAB \in P_G$  and the concatenation of the root labels of the sub-trees  $B$ ,  $A(\%)$  and  $B(0)$  is  $BAB$ , the overall tree will be valid for  $G$  if these sub-trees are valid for  $G$ .
- The parse tree  $B$  is valid for  $G$  since  $B \in Q_G$ .
- Since  $A \rightarrow \% \in P_G$ , the parse tree  $A(\%)$  is valid for  $G$ .

16

## (4.1) Validity Examples (Cont.)

- Since  $B \rightarrow 0 \in P_G$  and the root label of the sub-tree  $0$  is  $0$ , the parse tree  $B(0)$  will be valid for  $G$  if the sub-tree  $0$  is valid for  $G$ .
- The sub-tree  $0$  is valid for  $G$  since  $0 \in \text{alphabet}(G)$ .

Thus, we have that



is valid for  $G$ .

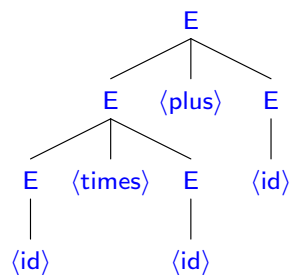
17

## (4.1) Validity Examples (Cont.)

Suppose  $G$  is our grammar of arithmetic expressions

$$E \rightarrow E(\text{plus})E \mid E(\text{times})E \mid \langle \text{openPar} \rangle E \langle \text{closPar} \rangle \mid \langle \text{id} \rangle.$$

Then the parse tree



is valid for  $G$ .

18

## (4.1) The Meaning of Grammars

A string  $w$  is *generated* by a grammar  $G$  iff  $w \in \mathbf{alphabet}(G)^*$  and there is a parse tree  $pt$  such that

- $pt$  is valid for  $G$ ;
- $\mathbf{rootLabel}(pt) = s_G$ ;
- $\mathbf{yield}(pt) = w$ .

The *language generated* by a grammar  $G$  ( $L(G)$ ) is

$$\{w \in \mathbf{Str} \mid w \text{ is generated by } G\}.$$

### Proposition 4.1.1

For all grammars  $G$ ,  $\mathbf{alphabet}(L(G)) \subseteq \mathbf{alphabet}(G)$ .

19

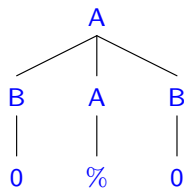
## (4.1) Grammar Meaning Examples

Let  $G$  be the example grammar

$$A \rightarrow BAB \mid \%,$$

$$B \rightarrow 0.$$

Then  $00$  is generated by  $G$  since  $00 \in \{0\}^* = \mathbf{alphabet}(G)^*$  and the parse tree



is valid for  $G$ , has  $s_G = A$  as its root label, and has  $00$  as its yield.

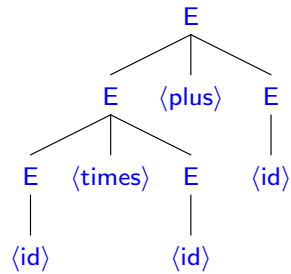
20

## (4.1) Grammar Meaning Examples (Cont.)

Suppose  $G$  is our grammar of arithmetic expressions

$$E \rightarrow E(\text{plus})E \mid E(\text{times})E \mid \langle \text{openPar} \rangle E \langle \text{closPar} \rangle \mid \langle \text{id} \rangle.$$

Then  $\langle \text{id} \rangle \langle \text{times} \rangle \langle \text{id} \rangle \langle \text{plus} \rangle \langle \text{id} \rangle$  is generated by  $G$  since  $\langle \text{id} \rangle \langle \text{times} \rangle \langle \text{id} \rangle \langle \text{plus} \rangle \langle \text{id} \rangle \in \mathbf{alphabet}(G)^*$  and the parse tree



is valid for  $G$ , has  $s_G = E$  as its root label, and has  $\langle \text{id} \rangle \langle \text{times} \rangle \langle \text{id} \rangle \langle \text{plus} \rangle \langle \text{id} \rangle$  as its yield.

21

## (4.1) Context-free Languages

A language  $L$  is *context-free* iff  $L = L(G)$  for some  $G \in \mathbf{Gram}$ . We define

$$\begin{aligned} \mathbf{CFLan} &= \{ L(G) \mid G \in \mathbf{Gram} \} \\ &= \{ L \in \mathbf{Lan} \mid L \text{ is context-free} \}. \end{aligned}$$

Since  $\{0^0\}$ ,  $\{0^1\}$ ,  $\{0^2\}$ ,  $\dots$ , are all context-free languages, we have that  $\mathbf{CFLan}$  is infinite. But, since  $\mathbf{Gram}$  is countably infinite, it follows that  $\mathbf{CFLan}$  is also countably infinite.

Since  $\mathbf{Lan}$  is uncountable, it follows that  $\mathbf{CFLan} \subsetneq \mathbf{Lan}$ , i.e., there are non-context-free languages. Later, we will see that  $\mathbf{RegLan} \subsetneq \mathbf{CFLan}$ .

22

## (4.1) Equivalence of Grammars

We say that grammars  $G$  and  $H$  are *equivalent* iff  $L(G) = L(H)$ . In other words,  $G$  and  $H$  are equivalent iff  $G$  and  $H$  generate the same language. We define a relation  $\approx$  on **Gram** by:  $G \approx H$  iff  $G$  and  $H$  are equivalent. It is easy to see that  $\approx$  is reflexive on **Gram**, symmetric and transitive.

23

## (4.1) Processing Parse Trees in Forlan

The Forlan module **PT** defines an abstract type **pt** of parse trees (in the top-level environment) along with some functions for processing parse trees:

```
val input      : string -> pt
val output     : string * pt -> unit
val height    : pt -> int
val size      : pt -> int
val equal     : pt * pt -> bool
val rootLabel : pt -> sym
val yield     : pt -> str
```

The Forlan syntax for parse trees is simply the linear syntax that we've been using in this section.

24

## (4.1) Using JTR to View and Edit Parse Trees

The Java program JTR (for “Java TRee”) can be used to display and edit parse trees. It can be invoked via Forlan, or run standalone. See the Forlan WWW site for more information.

25

## (4.1) More Parse Tree Processing

The Forlan module `Gram` also defines the functions

```
val checkPT : gram -> pt -> unit
val validPT : gram -> pt -> bool
```

The function `checkPT` is used to check whether a parse tree is valid for a grammar; if the answer is “no”, it explains why not and raises an exception; otherwise it simply returns `()`. The function `validPT` checks whether a parse tree is valid for a grammar, silently returning `true` if it is, and silently returning `false` if it isn't.

26

## (4.1) Forlan Examples

Suppose the identifier `gram` of type `gram` is bound to the grammar

$$\begin{aligned} A &\rightarrow BAB \mid \%, \\ B &\rightarrow 0. \end{aligned}$$

And, suppose that the identifier `gram'` of type `gram` is bound to our grammar of arithmetic expressions

$$E \rightarrow E(\text{plus})E \mid E(\text{times})E \mid \langle \text{openPar} \rangle E \langle \text{closPar} \rangle \mid \langle \text{id} \rangle.$$

27

## (4.1) Forlan Examples (Cont.)

Here are some examples of how we can process parse trees using Forlan:

```
- val pt = PT.input "";
@ A(B, A(%), B(0))
@ .
val pt = - : pt
- Sym.output("", PT.rootLabel pt);
A
val it = () : unit
- Str.output("", PT.yield pt);
B0
val it = () : unit
- Gram.validPT gram pt;
val it = true : bool
```

28

## (4.1) Forlan Examples (Cont.)

```
- val pt' = PT.input "";
@ E(E(E(<id>), <times>, E(<id>)), <plus>, E(<id>))
@ .
val pt' = - : pt
- Sym.output("", PT.rootLabel pt');
E
val it = () : unit
- Str.output("", PT.yield pt');
<id><times><id><plus><id>
val it = () : unit
- Gram.validPT gram' pt';
val it = true : bool
```

29

## (4.1) Forlan Examples (Cont.)

```
- Gram.checkPT gram pt';
invalid production : "E -> E<plus>E"

uncaught exception Error
- Gram.checkPT gram' pt';
invalid production : "A -> BAB"

uncaught exception Error
- PT.input "";
@ A(B,%,B)
@ .
% labels inappropriate node

uncaught exception Error
```

30

## (4.1) Grammar Synthesis Example

We conclude this section with a grammar synthesis example. Suppose  $X = \{0^n 1^m 2^m 3^n \mid n, m \in \mathbb{N}\}$ . How can we find a grammar  $G$  such that  $L(G) = X$ ?

The key is to think of generating the strings of  $X$  from the outside in, in two phases. In the first phase, one generates pairs of 0's and 3's, and, in the second phase, one generates pairs of 1's and 2's. E.g., a string could be formed in the following stages:

0 3,  
00 33,  
001233.

31

## (4.1) Grammar Synthesis Example (Cont.)

This analysis leads us to the grammar

$A \rightarrow 0A3,$   
 $A \rightarrow B,$   
 $B \rightarrow 1B2,$   
 $B \rightarrow \epsilon,$

where  $A$  corresponds to the first phase, and  $B$  to the second phase.

32

## (4.1) Grammar Synthesis Example (Cont.)

For example, here is how the string 001233 may be parsed using  $G$ :

