Formal Languages

Context free languages provide a convenient notation for recursive description of languages.

The original goal of formalizing the structure of natural languages is still elusive, but CFGs are now the universally accepted formalism for definition of (the syntax of) *programming* languages.

Writing parsers has become an almost fully automated process thanks to this theory.

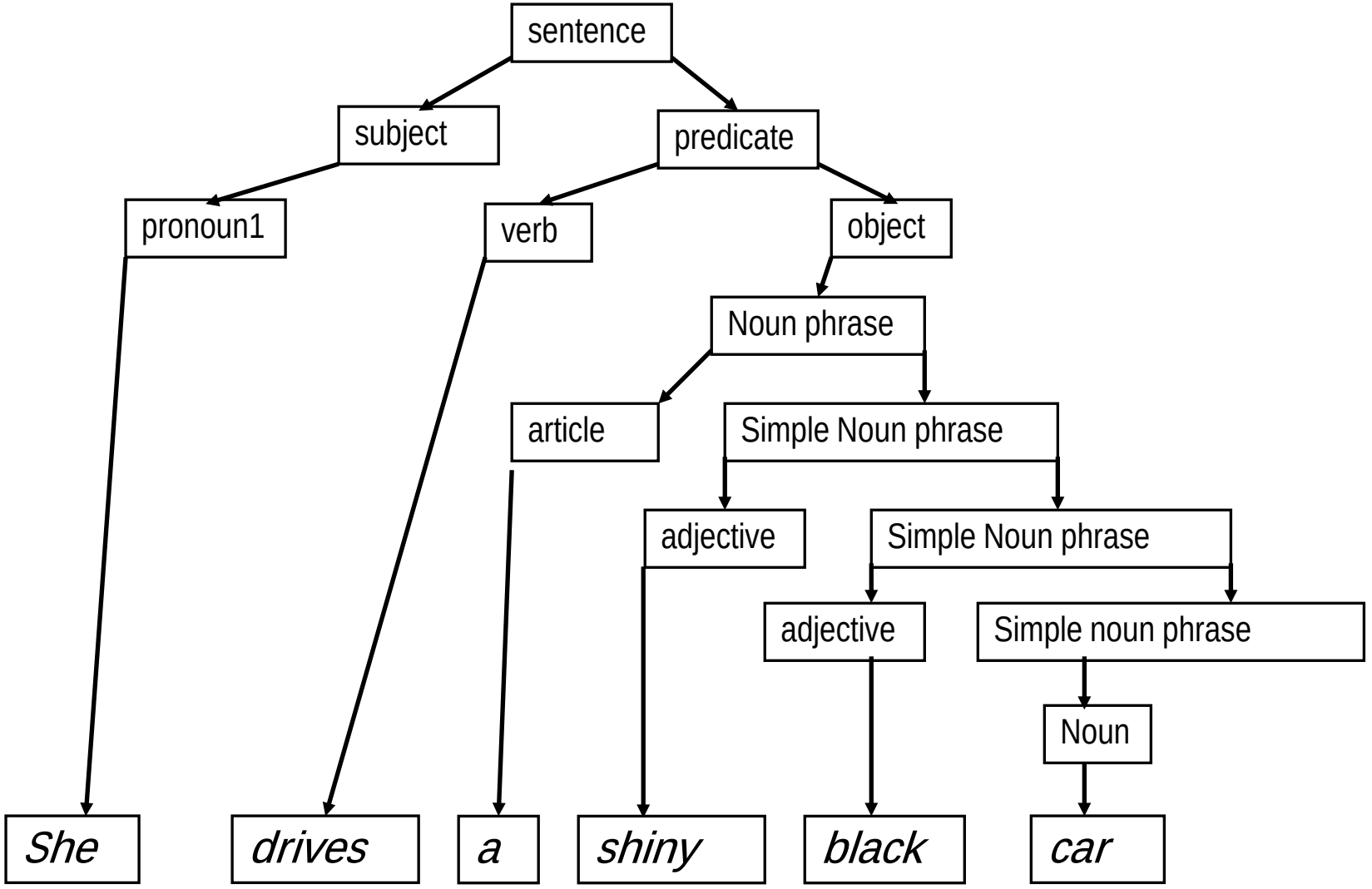
A Simple Grammar for English

Example taken from Floyd & Beigel.

<Sentence>	→	<Subject> <Predicate>
<Subject>	→	<Pronoun1> | <Pronoun2>
<Pronoun1>	→	**I** | **we** | **you** | **he** | **she** | **it** | **they**
<Noun Phrase>	→	<Simple Noun Phrase> | <Article> <Noun Phrase>
<Article>	→	**a** | **an** | **the**
<Predicate>	→	<Noun> | <Adjective> <Simple Noun Phrase>
<Simple Noun Phrase>	→	<Verb> | <Verb> <Object>
<Object>	→	<Pronoun2> | <Noun Phrase>
<Pronoun2>	→	**me** | **us** | **you** | **him** | **her** | **it** | **them**
<Noun>	→	. . .
<Verb>	→	. . .

Example

Derive the sentence "*She drives a shiny black car*" from these rules.



<sentence> ⇒
<subject> <predicate> ⇒
<pronoun> <predicate> ⇒
She <predicate> ⇒
She <verb> <object> ⇒
She drives <object> ⇒
She drives <simple noun phrase> ⇒
She drives <article> <noun phrase> ⇒
She drives a <noun phrase> ⇒
She drives a <adjective> <noun phrase> ⇒
She drives a shiny <noun phrase> ⇒
She drives a shiny <adjective> <simple noun phrase> ⇒
She drives a shiny black <simple noun phrase> ⇒
She drives a shiny black <noun> ⇒
She drives a shiny black car

A Grammar for Expressions

<Expression>	→	<Term> | <Expression> **+** <Term>
<Term>	→	<Factor> | <Term> ***** <Factor>
<Factor>	→	<Identifer> | **(** <Expression> **)**
<Identifier>	→	**x** | **y** | **z** | ...

In class exercise: Derive

- x + (y * 3)
- x + z * w + q

Definition of Context-Free-Grammars

A CFG is a quadruple G= (V,T,P,S), where

- V is a finite set of *variables* (*nonterminals*, *syntactic categories*)
- T is a finite set of *terminals*
- P is a finite set of *productions* -- rules of the form $X \longrightarrow a$, where $X \in V$ and $a \in (V \cup T)^*$
- S, the *start symbol*, is an element of V

Vertical bar (|), as used in the examples on the previous slide, is used to denote a set of several productions (with the same *lhs*).

Example

<Expression>	→	<Term> | <Expression> **+** <Term>
<Term>	→	<Factor> | <Term> ***** <Factor>
<Factor>	→	<Identifer> | **(** <Expression> **)**
<Identifier>	→	**x** | **y** | **z** | ...

V = {<Expression>, <Term>, <Factor>, <Identifier>}

T = {+, *, (,), x, y, z, ...}

P = {

<Expression> → <Term>
<Expression> → <Expression> **+** <Term>
<Term> → <Factor>
<Term> → <Term> ***** <Factor>
<Factor> → <Identifer>
<Factor> → **(** <Expression> **)**
<Identifier> → **x**
<Identifier> → **y**
<Identifier> → **z**
<Identifier> → ...

}

S = <Expression>

Notational Conventions

a, b, c, … (lower case, beginning of alphabet) are concrete terminals;

u, v, w, x, y, z (lower case, end of alphabet) are for strings of terminals

$\alpha, \beta, \gamma, \ldots$ (Greek letters) are for strings over $(T \cup V)$ (*sentential forms*)

A, B, C, … (capitals, beginning of alphabet) are for variables (for non-terminals).

X, Y, Z are for variables standing for terminals.

Short-hand

Note. We often abbreviate a context free grammar, such as:

$G_2 =$ (V={S},
T={(,)},
P={ $S \to \varepsilon$, $S \to SS$, $S \to (S)$ },
S=S}

By giving just its productions

$S \to \varepsilon$ | SS |(S)

And by using the following conventions.

1) The start symbol is the lhs of the first production.
2) Multiple production for the same lhs non-terminal can be grouped together by using vertical bar (|)
3) Non-terminals are capitalized.
4) Terminal-symbols are lower case or non-alphabetic.

Derivations

The single-step derivation relation $\Rightarrow$ on $(V \cup T)^*$ is defined by:

$\alpha \Rightarrow \beta$ iff β is obtained from α by replacing an occurrence of the lhs of a production with its rhs. That is, $\alpha'A\alpha'' \Rightarrow \alpha'\gamma\alpha''$ is true iff $A \rightarrow \gamma$ is a production.

We write $\alpha \Rightarrow^* \beta$ when β can be obtained from α through a sequence of several (possibly zero) derivation steps.

The *language of the CFG* , G, is the set

$L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$ (where S is the start symbol of G)

Context-free languages are languages of the form L(G)

Example 1

The familiar non-regular language

$$L = \{ a^k b^k \mid k \geq 0 \}$$

is context-free.

The grammar G_1 for it is given by T={a,b}, V={S}, and productions:

1. $S \rightarrow \Lambda$
2. $S \rightarrow a\ S\ b$

Here is a derivation showing $a^3b^3 \in L(G)$:

$S \Rightarrow_2 aSb \Rightarrow_2 aaSbb \Rightarrow_2 aaaSbbb \Rightarrow_1 aaabbb$

(Note: we sometimes label the arrow with a subscript which tells the production used to enable the transformation)

Example 1 continued

Note, however, that the fact $L=L(G_1)$ is not totally obvious. We need to prove set inclusion both ways.

To prove $L \subseteq L(G_1)$ we must show that there exists a derivation for every string $a^k b^k$; this is done by induction on k.

For the converse, $L(G_1) \subseteq L$, we need to show that if $S \Rightarrow^* w$ and $w \in T^*$, then $w \in L$. This is done by induction on the length of derivation of w.

Example 2

The language of balanced parentheses is context-free. It is generated by the following grammar:

$G_2 = (V=\{S\},$

$T=\{(,)\},$

$P=\{ S \rightarrow \Lambda \mid SS \mid (S)\},$

$S=S\}$

Example 3

Consider the grammar:

$S \rightarrow AS \mid \Lambda$

$A \rightarrow 0A1 \mid A1 \mid 01$

The derivation:

$S \Rightarrow AS \Rightarrow A1S \Rightarrow 011S \Rightarrow 011AS \Rightarrow$
$0110A1S \Rightarrow 0110011S \Rightarrow 0110011$

shows that $0110011 \in L(G_3)$.

Example 3 notes

The language $L(G_3)$ consists of strings $w \in \{0,1\}^*$ such that:

P(w): Either $w=\varepsilon$, or w begins with 0, and every block of 0's in w is followed by at least as many 1's

Again, the proof that G_3 generates all and only strings that satisfy P(w) is not obvious. It requires a two-part inductive proof.

Leftmost and Rightmost Derivations

The same string w usually has many possible derivations $S \equiv a_0 \Rightarrow a_1 \Rightarrow a_2 \Rightarrow \ldots \Rightarrow a_n \equiv w$

We call a derivation *leftmost* if in every step $a_i \Rightarrow a_{i+1}$, it is the first (leftmost) variable in $a_i\$$ that is being replaced with the rhs of a production. Similarly, in a *rightmost* derivation, it is always the last variable that gets replaced.

The above derivation of the string 0110011 in the grammar G_3 is leftmost. Here is a rightmost derivation of the same string:

$S \Rightarrow A\underline{S} \Rightarrow AA\underline{S} \Rightarrow A\underline{A} \Rightarrow A0\underline{A}1 \Rightarrow \underline{A}0011 \Rightarrow \underline{A}10011 \Rightarrow 0110011$

$S \rightarrow AS \mid \varepsilon$

$A \rightarrow 0A1 \mid A1 \mid 01$

Facts

Every Regular Language is also a Context Free Language

How might we prove this?

- Choose one of the many specifications for regular languages
- Show that every instance of that kind of specification has a total mapping into a Context Free Grammar

What is an appropriate choice?

In Class Exercise

Map the Regular Expressions into a Context Free language.

```
data RegExp a
  = Lambda                          -- the empty string ""
  | Empty                           -- the empty set
  | One a                           -- a singleton set {a}
  | Union (RegExp a) (RegExp a)     -- union of two RegExp
  | Cat (RegExp a) (RegExp a)       -- Concatenation
  | Star (RegExp a)                 -- Kleene closure
```

Find CFG for these languages

$\{a^n\, b\, a^n \mid n \in Nat\}$

$\{ w \mid w \in \{a,b\}^*,$ and w is a palindrome of even length$\}$

$\{a^n\, b^k \mid n,k \in Nat,\ n \leq k\}$

$\{a^n\, b^k \mid n,k \in Nat,\ n \geq k\}$

$\{ w \mid w \in \{a,b\}^*,$ w has equal number of a's and b's $\}$

Context Free Expressions

Just as the regular languages over an alphabet have a convenient shorthand (regular expressions) the context free languages also have a convenient short hand, context free expressions.

Definition. The set of *context free expressions* (with respect to Σ) is defined inductively by the following rules:

1. The symbols $\varnothing$ and Λ are context free expressions
2. Every symbol $\alpha \in \Sigma$ is a context free expression
3. If E and F are context free expressions, then (Mu @i . E), (EF) and (E+F) are regular expressions.
4. In a surrounding Mu context, @i , is a context free expression.

Definition as a datatype

```
data CfExp a
  = Lambda                          -- the empty string “”
  | Empty                           -- the empty set {}
  | One a                           -- a singleton set {a}
  | Union (CfExp a) (CfExp a)       -- union of two CfExp
  | Cat (CfExp a) (CfExp a)         -- Concatenation
  | Mu Int (CfExp a)                -- Recursion
  | V Int                           -- Variable
```

Note the two new kinds of expressions

Mu and V, which replace the Star operator of the regular expressions

Context Free Expressions as Languages

Definition. For every context free expression E, there is an associated language *L(E)*, defined inductively as follows:

1. $L(\varnothing)=\varnothing$ and $L(\Lambda)=\{\Lambda\}$
2. L(a)={'a'}
3. Inductive cases
    1. L(EF) = L(E) L(F) recall implicit use of dot L(E) • L(F)
    2. $L(E+F) = L(E) \cup L(F)$
    3. L(Mu @i . E) = L(E[Mu @i .E / @i])
        1. Where E[d/@i] denotes substitution of d for @i

Definition. A language is *context free* if it is of the form L(E) for some context free expression E.

Conventions

- We use juxtaposition to denote concatenation
- We use parentheses to denote grouping
- We use # for $\varnothing$
- We use ^ for Λ
- We denote variables as @i for all positive integers i.
- We use the (RegLang) Star operator as a shorthand

```
exp*  = Mu @i.(exp + ^)@i
```

Find Context Free Expression for these languages

$\{a^n\, b\, a^n \mid n \in Nat\}$

$\{ w \mid w \in \{a,b\}^*,$ and w is a palindrome of even length$\}$

$\{a^n\, b^k \mid n,k \in Nat,\ n \leq k\}$

$\{a^n\, b^k \mid n,k \in Nat,\ n \geq k\}$

$\{ w \mid w \in \{a,b\}^*,$ w has equal number of a's and b's $\}$

Meaning of a CFExp

```
-- extract an "nth" approximation of the set denoted

meaning:: Ord a => Int -> (CfExp a) -> Set [a]
meaning n (One x) = one x
meaning n Lambda = lam
meaning n Empty = empty
meaning n (Union x y) =
   union (meaning n x) (meaning n y)
meaning n (Cat x y) = cat (meaning n x) (meaning n y)
meaning n (V m) =
   error ("Unbound variable: v"++show m++".")
meaning n (x@(Mu v body)) =
   meaning n (unfold n v body body)
```