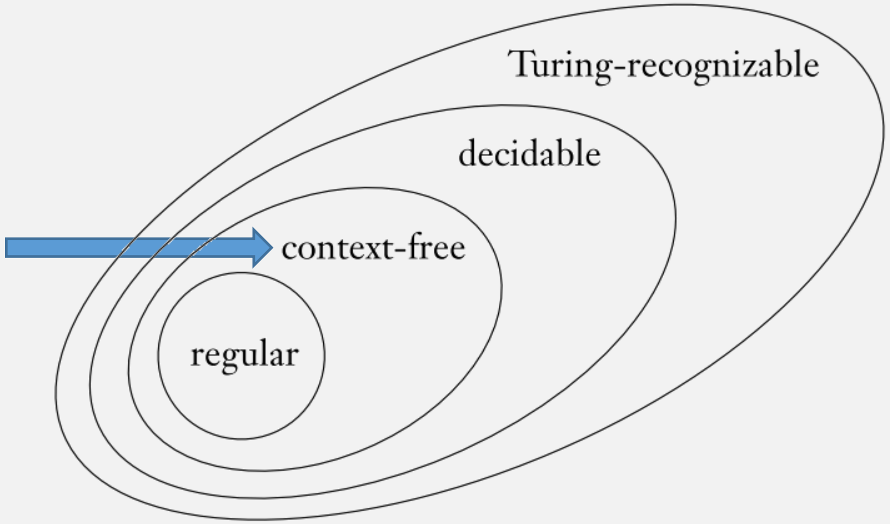
UMB CS 420

Context-Free Languages (CFLs)

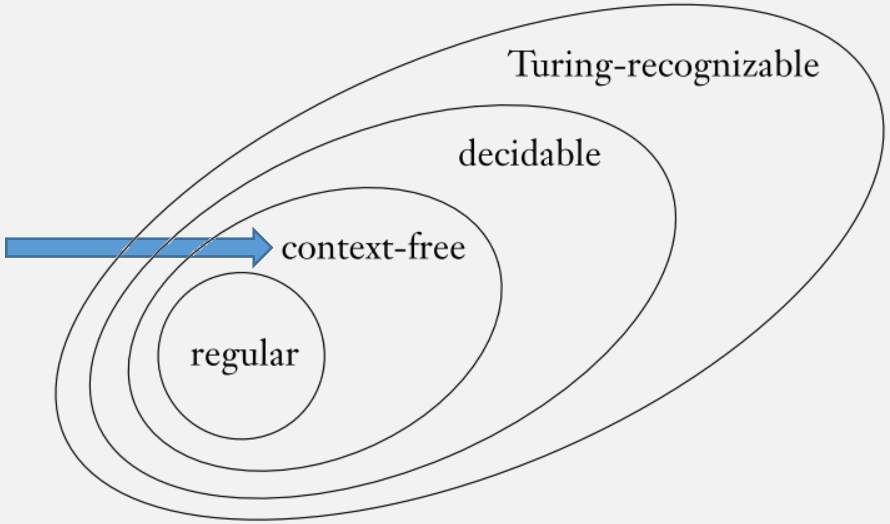
Monday, March 18, 2024



Announcements

- HW 4 in
  - ~~due Mon 3/18 12pm noon~~

- HW 5 out
  - due Mon 3/25 12pm noon



Last Time:

Non-Regular Languages

Example:

$L = \{ 0^n 1^n \mid n \geq 0 \}$

An arbitrary count

- **A DFA recognizing** L would **require infinite states!** (impossible)
  - States representing: **zero 0s seen, one 0 seen, two 0s, …**
- **This language is the same as many PLs,** e.g., **HTML!**
  - To better see this replace:
    - "**0**" with "**<tag>**" or "**(**"
    - "**1**" with "**</tag>**" or "**)**"
- The Problem: **remembering nestedness**
  - **Need to count arbitrary nesting depths**
    - E.g., `if { if { if { … } } }`
  - Thus: **most programming language syntax** is **not regular**!

We can *prove* **non-regularness** … with the **Pumping Lemma** (and **proof by contradiction**)

But … what kind of language is it then?

A Context-Free Grammar (CFG)

terminals

Top variable is:
Start variable

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

Variables
(a.k.a., **non-terminals**)
on RHS of rules

Substitution rules
(a.k.a., **productions**)

terminals (analogous to DFA's alphabet)

Context-Free Grammar (CFG): Formally

Grammar $G_1 = (V, \Sigma, R, S)$

R is this set of rules (mappings):

Top variable is:
Start variable

terminals

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \#$$

Variables
(a.k.a., **non-terminals**)

Substitution rules
(a.k.a., **productions**)

terminals (analogous to DFA's alphabet)

CFG Practical Application: Used to describe programming language syntax!

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

$V = \{A, B\}$,

$\Sigma = \{0, 1, \#\}$,

$S = A$,

Java Syntax: Described with CFGs

Definition:
A **CFG describes a context-free language!**

ORACLE

Java SE > Java SE Specifications > Java Language Specification

Chapter 2

Prev

Chapter 2. Grammars

This chapter describes the context-free grammars used in this specification to define the lexical and syntactic structure of a progran

2.1. Context-Free Grammars

"productions" = rules

"nonterminal" = variable

A *context-free grammar* consists of a number of *productions*. Each production has an abstract symbol called a *nonterminal* as its *lef hand side*, and a sequence of one or more nonterminal and *terminal* symbols as its *right-hand side*. For each grammar, the terminal symbols are drawn from a specified *alphabet*.

"goal symbol" = Start variable

Starting from a sentence consisting of a single distinguished nonterminal, called the *goal symbol*, a given context-free grammar specifies a language, namely, the set of possible sequences of terminal symbols that can result from repeatedly replacing any nonterminal in the sequence with a right-hand side of a production for which the nonterminal is the left-hand side.

A **CFG** specifies a language!

(definition of a **language:** sequence of symbols)

2.2. The Lexical Grammar

A *lexical grammar* for the Java programming language is given in §3. This grammar has as its terminal symbols the characters of th Unicode character set. It defines a set of productions, starting from the goal symbol *Input* (§3.5), that describe how sequences of Unicode characters (§3.1) are translated into a sequence of input elements (§3.5)

https://docs.oracle.com/javase/specs/jls/se7/html/jls-2.html

Analogies

Regular Language	Context-Free Language (CFL)
Regular Expression	Context-Free Grammar (CFG)
A **Reg Expr** describes a **Regular lang**	A **CFG** describes a **CFL**

thm

def

Python Syntax: Described with a CFG

(partially)

10. Full Grammar specification

This is the full Python grammar, as it is read by the parser generator and used to parse Python source files:

```
# Grammar for Python

# NOTE WELL: You should also follow all the steps listed at
# https://devguide.python.org/grammar/

# Start symbols for the grammar:
#       single_input is a single interactive statement;
#       file_input is a module or sequence of commands read from an input file;
#       eval_input is the input for the eval() functions.
#       func_type_input is a PEP 484 Python 2 function type comment
# NB: compound_stmt in single_input is followed by extra NEWLINE!
# NB: due to the way TYPE_COMMENT is tokenized it will always be followed by a NEWLINE
single_input: NEWLINE | simple_stmt | compound_stmt NEWLINE
file_input: (NEWLINE | stmt)* ENDMARKER
eval_input: testlist NEWLINE* ENDMARKER
```

(indentation checking not expressible with CFG?)

https://docs.python.org/3/reference/grammar.html

Many Other Language (partially) ~~Python~~ Syntax: Described with a CFG

10. Full Grammar specification

This is the full Python grammar, as it is read by the parser generator and used to parse Python source files:

```
# Grammar for Python

# NOTE WELL: You should also follow all the steps listed at
# https://devguide.python.org/grammar/

# Start symbols for the grammar:
#       single_input is a single interactive statement;
#       file_input is a module or sequence of commands read from an input file;
#       eval_input is the input for the eval() functions.
#       func_type_input is a PEP 484 Python 2 function type comment
# NB: compound_stmt in single_input is followed by extra NEWLINE!
# NB: due to the way TYPE_COMMENT is tokenized it will always be followed by a NEWLINE
single_input: NEWLINE | simple_stmt | compound_stmt NEWLINE
file_input: (NEWLINE | stmt)* ENDMARKER
eval_input: testlist NEWLINE* ENDMARKER
```

https://docs.python.org/3/reference/grammar.html

Java Syntax: Described with CFGs

Definition:
A **CFG** describes a **context-free language**!
but what <u>strings</u> are in the language?

ORACLE

Java SE > Java SE Specifications > Java Language Specification

Prev

Chapter 2. Grammars

This chapter describes the context-free grammars used in this specification to define the lexical and syntactic structure of a program

2.1. Context-Free Grammars

A *context-free grammar* consists of a number of *productions*. Each production has an abstract symbol called a *nonterminal* as its *left-hand side*, and a sequence of one or more nonterminal and *terminal* symbols as its *right-hand side*. For each grammar, the terminal symbols are drawn from a specified *alphabet*.

Starting from a sentence consisting of a single distinguished nonterminal, called the *goal symbol*, a given context-free grammar specifies a language, namely, the set of possible sequences of terminal symbols that can result from repeatedly replacing any nonterminal in the sequence with a right-hand side of a production for which the nonterminal is the left-hand side.

2.2. The Lexical Grammar

A *lexical grammar* for the Java programming language is given in §3. This grammar has as its terminal symbols the characters of the Unicode character set. It defines a set of productions, starting from the goal symbol *Input* (§3.5), that describe how sequences of Unicode characters (§3.1) are translated into a sequence of input elements (§3.5)

https://docs.oracle.com/javase/specs/jls/se7/html/jls-2.html

Generating Strings with a CFG

In-class exercise:

Write 3 more strings that can be generated by this grammar

Definition:
A **CFG** describes a **context-free language**!
but what <u>strings</u> are in the language?

1st rule → $A \rightarrow 0A1$

2nd rule → $A \rightarrow B$

Last rule → $B \rightarrow \#$

"Applying a rule" = replace LHS variable with RHS sequence

At each step, ***arbitrarily*** choose <u>any</u> variable to replace, and <u>any</u> rule to apply

Stop when: **string is all terminals**

A CFG **generates** a string, by repeatedly <u>applying</u> substitution rules:

Example: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

- Start with: Start variable
- Apply 1st rule
- 1st rule again
- 1st rule again
- Apply 2nd rule
- Apply last rule

Generating Strings with a CFG

Definition:
A **CFG** describes a **context-free language**!
but **what strings** are in the language?

$$G_1 = \begin{aligned} A &\rightarrow 0A1 \\ A &\rightarrow B \\ B &\rightarrow \# \end{aligned}$$

Strings in CFG's language
= all possible **generated** / **derived** strings

$L(G_1)$ is $\{0^n\#1^n \mid n \geq 0\}$

A CFG **generates** a string, by repeatedly applying substitution rules:

Example: $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$

Derivations: Formally

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

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

Single-step

$$\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$$

Where:

$\alpha, \beta \in (V \cup \Sigma)^*$ ← sequence of terminals or variables

$A \in V$ ← Variable

$A \to \gamma \in R$ ← Rule

Derivations: Formally

A *context-free grammar* is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

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

Single-step

$$\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$$

Where:

$\alpha, \beta \in (V \cup \Sigma)^*$ ← sequence of terminals or variables

$A \in V$ ← Variable

$A \rightarrow \gamma \in R$ ← Rule

Multi-step (recursively defined)

Base case: $\alpha \overset{*}{\underset{G}{\Rightarrow}} \alpha$ (0 steps)

Recursive case: (> 0 steps)

Single step

- If: $\alpha \underset{G}{\Rightarrow} \beta$ and $\beta \overset{*}{\underset{G}{\Rightarrow}} \gamma$
- Then: $\alpha \overset{*}{\underset{G}{\Rightarrow}} \gamma$

(smaller) Recursive "call"

Formal Definition of a CFL

A ***context-free grammar*** is a 4-tuple (V, Σ, R, S), where

1. V is a finite set called the ***variables***,
2. Σ is a finite set, disjoint from V, called the ***terminals***,
3. R is a finite set of ***rules***, with each rule being a variable and a string of variables and terminals, and
4. $S \in V$ is the start variable.

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

"all possible sequences of terminal symbols …"

"the language of a grammar *G* is …"

$$L(G) = \left\{ w \in \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w \right\}$$

… "that can be **generated** with rules of grammar *G*"

Any language that can be generated by some context-free grammar is called a ***context-free language***

Flashback: $\{0^n 1^n | \, n \geq 0\}$

- Pumping Lemma says: not a regular language
- It's a context-free language!
  - Proof?
  - Key step: Come up with CFG describing it …
  - Hint: It's similar to:

$$A \rightarrow 0A1$$
$$A \rightarrow B$$
$$B \rightarrow \sout{\#}\ \varepsilon$$

$L(G_1)$ is $\{0^n \sout{\#} 1^n | \, n \geq 0\}$

Statements and Justifications?

Proof: $L = \{0^n 1^n \mid n \geq 0\}$ is a CFL

Statements	Justifications
1. If a CFG describes a language, then it is a CFL	1. Definition of CFL
2. CFG G_1 describes L $\quad A \rightarrow 0A1$, $A \rightarrow B$, $B \rightarrow \varepsilon$	2. (Did you come up with examples???)
3. $L = \{0^n 1^n \mid n \geq 0\}$ is a CFL	3. By Statements #1 and #2

"|" symbol = Shorthand for multiple rules with same LHS variable

A String Can Have Multiple Derivations

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{TERM} \rangle \mid \langle \text{TERM} \rangle$$
$$\langle \text{TERM} \rangle \rightarrow \langle \text{TERM} \rangle \times \langle \text{FACTOR} \rangle \mid \langle \text{FACTOR} \rangle$$
$$\langle \text{FACTOR} \rangle \rightarrow (\langle \text{EXPR} \rangle) \mid \mathtt{a}$$

Want to generate this string: **a + a × a**

- EXPR ⇒
- EXPR + TERM ⇒
- EXPR + TERM × FACTOR ⇒
- EXPR + TERM × a ⇒

…

RIGHTMOST DERIVATION

- EXPR ⇒
- EXPR + TERM ⇒
- TERM + TERM ⇒
- FACTOR + TERM ⇒
- **a** + TERM

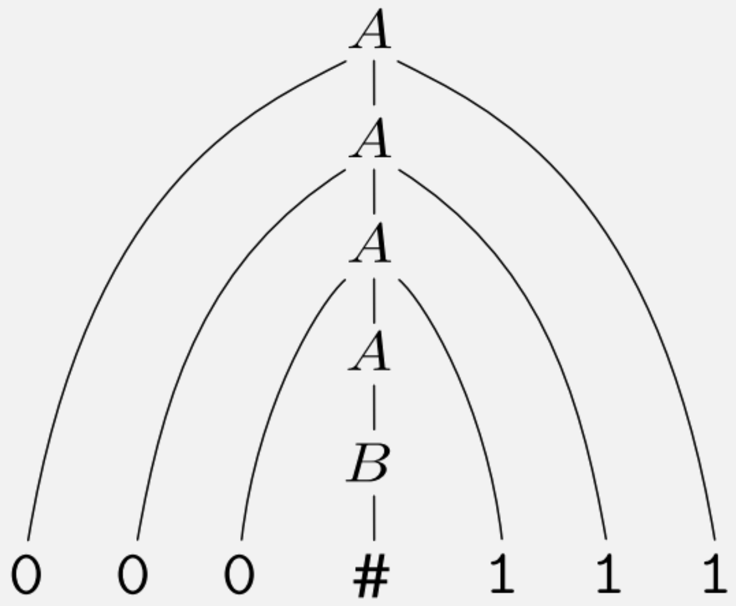
…

LEFTMOST DERIVATION

Derivations and Parse Trees

$$A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 000B111 \Rightarrow 000\#111$$

A derivation may also be represented as a **parse tree**



Multiple Derivations, Single Parse Tree

Leftmost deriviation

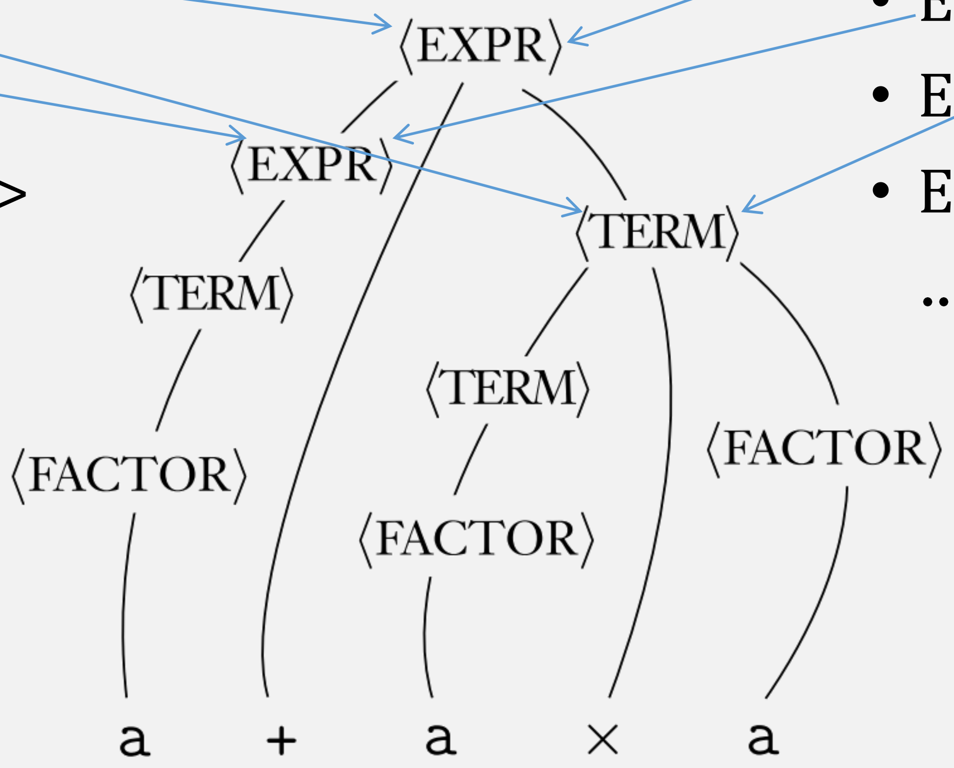
- <u>EXPR</u> =>
- <u>EXPR</u> + TERM =>
- <u>TERM</u> + TERM =>
- <u>FACTOR</u> + TERM =>
- a + TERM

...

Rightmost deriviation

- <u>EXPR</u> =>
- EXPR + <u>TERM</u> =>
- EXPR + TERM x <u>FACTOR</u> =>
- EXPR + TERM x a=>

...



<u>Same</u> parse tree

A **parse tree** represents
a CFG <u>computation</u> … like
a **sequence of states** represents
a DFA <u>computation</u>

A **Parse Tree**
gives
“meaning”
to a string

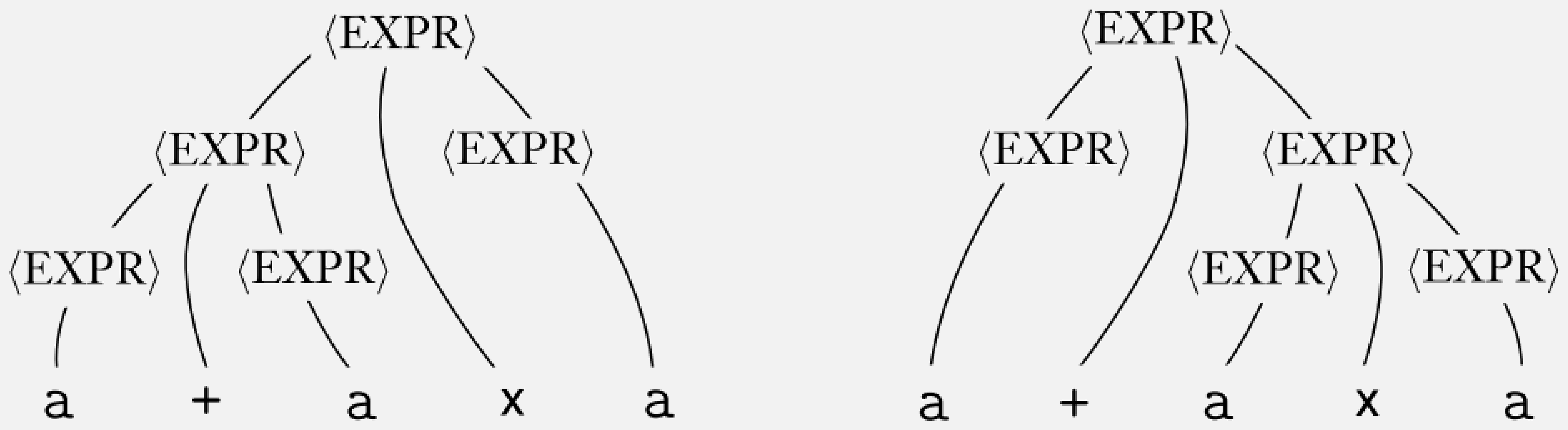
Ambiguity

grammar G_5:

$$\langle \text{EXPR} \rangle \to \langle \text{EXPR} \rangle \texttt{+} \langle \text{EXPR} \rangle \mid \langle \text{EXPR} \rangle \texttt{x} \langle \text{EXPR} \rangle \mid \texttt{(} \langle \text{EXPR} \rangle \texttt{)} \mid \texttt{a}$$

Same **string,**
different **derivation,**
and different **parse tree!**

So this string has
two meanings!



Ambiguity

A string w is derived ***ambiguously*** in context-free grammar G if it has two or more different leftmost derivations. Grammar G is ***ambiguous*** if it generates some string ambiguously.

An ambiguous grammar can give a string <u>multiple meanings</u>, ie represent <u>two different computations</u>!
(why is this <u>bad</u>?)

Real-life Ambiguity ("Dangling" `else`)

- What is the result of this `C` program?

```
if (1) if (0) printf("a"); else printf("2");
```

```
if (1)
  if (0)
    printf("a");
  else
    printf("2");
```

VS

```
if (1)
  if (0)
    printf("a");
else
  printf("2");
```

This string has 2 parsings, and thus 2 meanings!

Ambiguous grammars are confusing. A computation on a string should ideally have only one result.

Thus in practice, we typically focus on the **unambiguous** subset of CFGs (CFLs) (more on this later)

Problem is, there's no easy way to create an **unambiguous** grammar (it's up to language designers to "be careful")

Designing Grammars : Basics

1. Think about what you want to "link" together

- E.g., $0^n 1^n$
  - $A \rightarrow 0A1$
  - # 0s and # 1s are "linked"

- E.g., XML
  - $\mathrm{ELEMENT} \rightarrow <\mathrm{TAG}>\mathrm{CONTENT}</\mathrm{TAG}>$
  - Start and end tags are "linked"

2. Start with small grammars and then combine (just like FSMs)

Designing Grammars: Building Up

- Start with small grammars and then combine (just like FSMs)
  - To create a grammar for the language $\{0^n1^n | n \geq 0\} \cup \{1^n0^n | n \geq 0\}$
  - First create grammar for lang $\{0^n1^n | n \geq 0\}$:

    $$S_1 \rightarrow 0S_11 \mid \varepsilon$$

  - Then create grammar for lang $\{1^n0^n | n \geq 0\}$:

    $$S_2 \rightarrow 1S_20 \mid \varepsilon$$

  - Then combine: $S \rightarrow S_1 \mid S_2$

    $$S_1 \rightarrow 0S_11 \mid \varepsilon$$

    $$S_2 \rightarrow 1S_20 \mid \varepsilon$$

New start variable and rule combines two smaller grammars

"|" = "or" = union (combines 2 rules with same left side)

(Closed) Operations for CFLs?

- Start with small grammars and then combine (just like FSMs)
- "Or": $S \rightarrow S_1 \mid S_2$
- "Concatenate": $S \longrightarrow S_1 S_2$
- "Repetition": $S' \longrightarrow S' S_1 \mid \varepsilon$

Could you write out the full proof?

In-class Example: Designing grammars

alphabet Σ is $\{0,1\}$

$\{w \mid w \text{ starts and ends with the same symbol}\}$

1) come up with examples: In the language: 010, 101, 11011 1, 0 ?

Not in the language: 10, 01, 110 ε ?

2) Create CFG:

$S \rightarrow 0C'0 \mid 1C'1 \mid 0 \mid 1$ "string starts/ends with same symbol, middle can be anything"

$C' \rightarrow C'C \mid \varepsilon$ "middle: all possible terminals, repeated (ie, all possible strings)"

$C \rightarrow 0 \mid 1$ "all possible terminals"

3) Check CFG: **generates examples** in the language; **does not generate examples** not in language

Next Time:

Regular Languages	Context-Free Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
A Reg Expr describes a Regular Lang	A CFG describes a CFL
Finite Automaton (FSM)	???
An FSM recognizes a Regular Lang	A ??? recognizes a CFL

Next Time:

Regular Languages	**Context-Free Languages (CFLs)**
Regular Expression	Context-Free Grammar (CFG)
A Reg Expr <u>describes</u> a Regular Lang	A CFG <u>describes</u> a CFL
Finite Automaton (FSM)	**Push-down Automaton** (PDA)
An FSM <u>recognizes</u> a Regular Lang	A **PDA** <u>recognizes</u> a CFL

Next Time:

Regular Languages	Context-Free Languages (CFLs)
Regular Expression	Context-Free Grammar (CFG)
A Reg Expr describes a Regular Lang	A CFG describes a CFL
Finite Automaton (FSM)	**Push-down Automaton** (PDA)
An FSM recognizes a Regular Lang	A **PDA** recognizes a CFL
DIFFERENCE:	DIFFERENCE:
A Regular Lang is defined with a FSM	A CFL is defined with a CFG
Proved: Reg Expr ⇔ Reg Lang	*Must prove*: **PDA** ⇔ CFL

thm

def

def

thm

Submit in-class work 3/18

On gradescope