CS420
Context-Free Languages (CFL)

Monday, October 5, 2020

HW3 Questions?

HW2: For all langs, DFA recog ⇔ NFA recog

- DFA => NFA
  - Let δ_{dfa} be DFA's transition fn
  - Then equiv NFA's transition fn is $\delta_{nfa}(q,x) = \{\delta_{dfa}(q,x)\}$
  - NOTE: cannot just say DFA = NFA (the formal defs are different)
    - Must explain how
- NFA => DFA
  - By Thm 1.39

- Alternate answer: DFA ⇔ reg lang, and reg lang ⇔NFA

Can a DFA/NFA "do computation"?

$$\Sigma_2 = \left\{ \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}$$

$$D = \{w \in \Sigma_2^* | \text{ the top row of } w \text{ is a larger number than is the bottom row}\}$$

Consider each row to be a binary number

- Is language D regular?
  - I.e., can an DFA/NFA "do" this (greater than) computation?
- Machine M recognizing D:
  - 3 states: *maybe*, *top-row-greater*, *bot-row-greater*
  - Start state: *maybe*
  - Accept states: *top-row-greater*
  - Transitions (for each pair of bits, starting in *maybe*):
    - If top bit greater, go to *top-row-greater*, and stay there
    - If bot bit greater, go to *bot-row-greater*, and stay there
    - Else stay in *maybe* state

RegEx match open tags except XHTML self-cont

Asked 10 years, 10 months ago Active 1 month ago Viewed 2.9m times

1553

6572

I need to match all of these opening tags:

```
<p>
<a href="foo">
```

But not these:

```
<br />
<hr class="foo" />
```

You can't parse [X]HTML with regex. Because HTML can't be parsed by regex. Regex is not a tool that can be used to correctly parse HTML. As I have answere HTML-and-regex questions here so many times before, the use of regex will not allow you to consume HTML. Regular expressions are a tool that is insufficiently sophisticated to understand the constructs employed by HTML. HTML is not a regular language and hence cannot be parsed by regular expressions. Regex queries are not equipped to break down HTML into its meaningful parts. so many times but it is not getting to me. Even enhanced irregular regular expressions as used by Perl are not up to the task of parsing HTML. You will never make me cra

HTML is a language of sufficient complexity that it cannot be parsed by regular expressions. Even Jon Skeet cannot parse HTML using regular expressions. Every time you attempt to parse HTML with regular expressions, the unholy child weeps the blood of virgins, and Russian hackers pwn your webapp. Parsing HTML with regex summons tainted souls into the realm of the living. HTML and regex go together like love, marriage, and ritual infanticide. The cannot hold it is too late. The force of regex and HTML together in the same conceptual space will destroy your mind like so much watery putty. If you parse HTML with regex you are giving in to Them and their blasphemous ways which doom us all to inhuman toil for the One whose Name cannot be expressed in the Basic Multilingual Plane, he comes. HTML-plus-regexp will liquify the nerves of the sentient whilst you observe, your psyche withering in the onslaught of horror. Regex-based HTML parsers are the cancer that is killing StackOverflow *it is too late it is too late we cannot be saved* the trangession of a child ensures regex will consume all living tissue (except for HTML which it cannot, as previously prophesied) *dear lord help us how can anyone survive this scourge* using regex to parse HTML has doomed humanity to an eternity of dread torture and security holes *using rege*x as a tool to process HTML establishes a bre*ach between this world* and the dread realm of corrupt entities (like SGML entities, but *more corrupt) a mere glimp*se of the world of reg**ex parsers for HTML will ins**tantly transport a p*rogrammer's consciousness i*nto a wo*r*ld of ceaseless screaming, he comes~~, the pestilent sl~~ithy regex-infection wi**ll devour your HT**ML parser, application and existence for all time like Visual Basic only worse *he comes he comes do not fi*ght h**e com~~es, hi~~s unholy radiańcé de*stroying all enlightenment, HTML tags leaking from your eyes like liq**uid p*ain, the song of regular expr~~ession parsing~~ will ext*inguish the voices of mor**tal man from the sp**here I can see it can you see it it is beautiful t*he f`inal snuf`fing *of the lie**s of Man ALL IS LOST* **ALL IS L**OST the *pony he come*s he com~~es he comes~~ **the E**ichor permeates *all* MY FAC*E MY FACE oh god no* **NO NOOOO N**Θ stop *the an*gles `are` **not real ZALGO IS TONY THE PONY HE COMES**

Have you tried using an XML parser instead?

The other side of the argument

3281

+50

While *arbitrary* HTML with only a regex is impossible, it's sometimes appropriate to use them for parsing a *limited, known* set of HTML.

If you have a small set of HTML pages that you want to scrape data from and then stuff into a database, regexes might work fine. For example, I recently wanted to get the names, parties, and districts of Australian federal Representatives, which I got off of the Parliament's web site. This was a limited, one-time job.

Regexes worked just fine for me, and were very fast to set up.

share edit follow flag

edited Sep 19 '19 at 15:30

community wiki
10 revs, 10 users 36%
Kaitlin Duck Sherwood

131 Also, scraping fairly regularly formatted data from large documents is going to be WAY faster with judicious use of scan & regex than any generic parser. And if you are comfortable with coding regexes, way faster to code than coding xpaths. And almost certainly less fragile to changes in what you are scraping. So bleh. – Michael Johnston Apr 17 '12 at 20:47

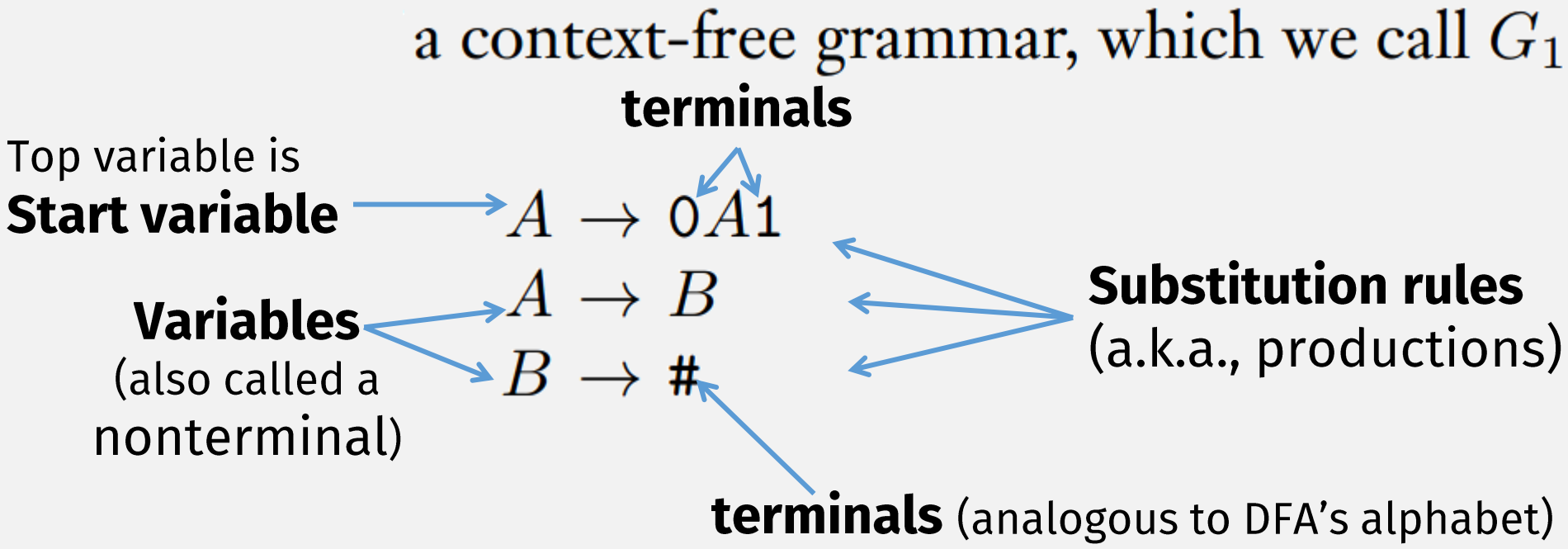
262 @MichaelJohnston "Less fragile"? Almost certainly not. Regexes care about text-formatting details than an XML parser can silently ignore. Switching between `&foo;` encodings and `CDATA` sections? Using an HTML minifier to remove all whitespace in your document that the browser doesn't render? An XML parser won't care, and neither will a well-written XPath statement. A regex-based "parser", on the other hand... – Charles Duffy Jul 11 '12 at 16:03

So XML is not regular? What is it?

- It’s a context-free language (sort of)!
- What’s a context-free language (CFL)?
- How do you parse a CFL?

Context-Free Gramamrs (CFG)

A Context-Free Grammar (CFG)



Pytyon Syntax Specified with a CFG

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

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
```

Java Syntax Specified with a CFG

https://docs.oracle.com/javase/specs/jls/se7/html/jls-18.html

Chapter 18. Syntax

This chapter presents a grammar for the Java programming language.

The grammar presented piecemeal in the preceding chapters (§2.3) is much better for exposition, but it is not well suited as a basis for a parser. The grammar presented in this chapter is the basis for the reference implementation. Note that it is not an LL(1) grammar, though in many cases it minimizes the necessary look ahead.

The grammar below uses the following BNF-style conventions:

- *[x]* denotes zero or one occurrences of *x*.
- *{x}* denotes zero or more occurrences of *x*.
- *(x | y)* means one of either *x* or *y*.

```
Identifier:
    IDENTIFIER

QualifiedIdentifier:
    Identifier { . Identifier }

QualifiedIdentifierList:
    QualifiedIdentifier { , QualifiedIdentifier }
```

A Context-Free Grammar (CFG)

a context-free grammar, which we call G_1

terminals

Start variable → $A \to 0A1$

$A \to B$

Variables (also called a nonterminal)

$B \to \#$

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

terminals (analogous to DFA's alphabet)

A CFG **generates** a string, by repeatedly substituting variables:

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

This sequence is called a **derivation**

Formal Definition of a CFG

$V = \{A, B\}$, $\Sigma = \{\mathtt{0}, \mathtt{1}, \mathtt{\#}\}$, $S = A$, and R is

$$A \to \mathtt{0}A\mathtt{1}$$
$$A \to B$$
$$B \to \mathtt{\#}$$

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.

Formal Definition of a Derivation

If u, v, and w are strings of variables and terminals, and $A \to w$ is a rule of the grammar, we say that uAv ***yields*** uwv, written $uAv \Rightarrow uwv$. Say that u ***derives*** v, written $u \overset{*}{\Rightarrow} v$, if $u = v$ or if a sequence $u_1, u_2, \ldots, u_k$ exists for $k \geq 0$ and

$$u \Rightarrow u_1 \Rightarrow u_2 \Rightarrow \ldots \Rightarrow u_k \Rightarrow v.$$

The ***language of the grammar*** is $\{w \in \Sigma^* | \; S \overset{*}{\Rightarrow} w\}$.

Definition of context-free language (CFL)

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

A Context-Free Grammar (CFG)

a context-free grammar, which we call G_1

$$A \to 0A1$$
$$A \to B$$
$$B \to \#$$

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

- Remember: in Ch1 we proved $\{0^n1^n \mid n \geq 0\}$ not regular
  - It's a CFL!
- If we replace: 0 -> "<tag>", 1 -> "</tag>", it looks like …
- XML resembles a CFL!

Analogies

Regular Language	Context-Free Language (CFL)
Regular Expression (Regexp)	Context-Free Grammar (CFG)
Regexp describes a Reg lang	CFG describes/generates a CFL

In-class exercise: derivations

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

- Come up with a derivation (a sequence of substs) for string:
  - a + a x a

A String Can Have Multiple Derivations

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

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

...

RIGHTMOST DERIVATION

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

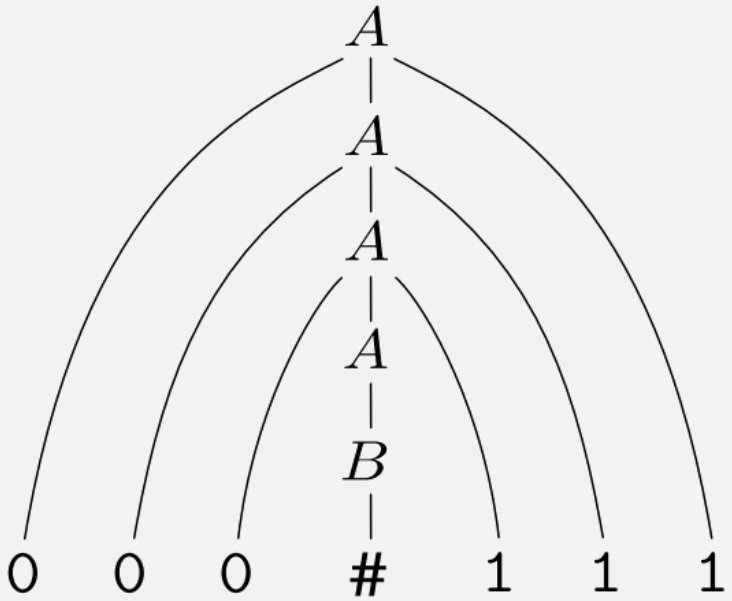
...

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**



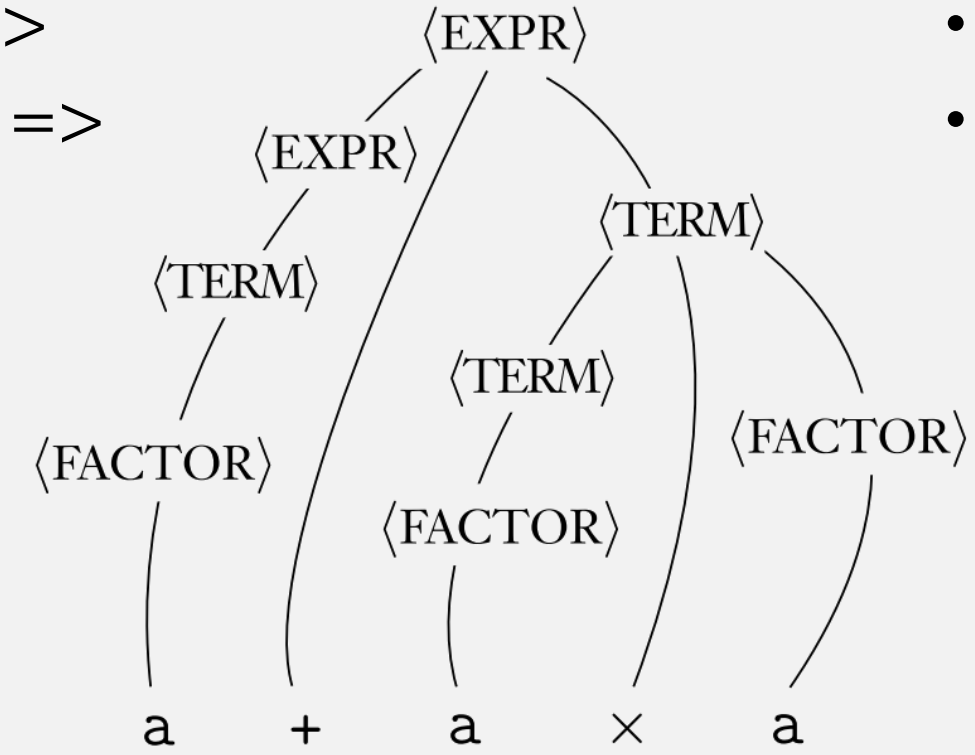
Parse Tree gives "meaning" to a string

Multiple Derivations, Single Parse Tree

- EXPR =>
- EXPR + TERM =>
- TERM + TERM =>
- FACTOR + TERM =>
- a + TERM

...

Always use **leftmost** derivation



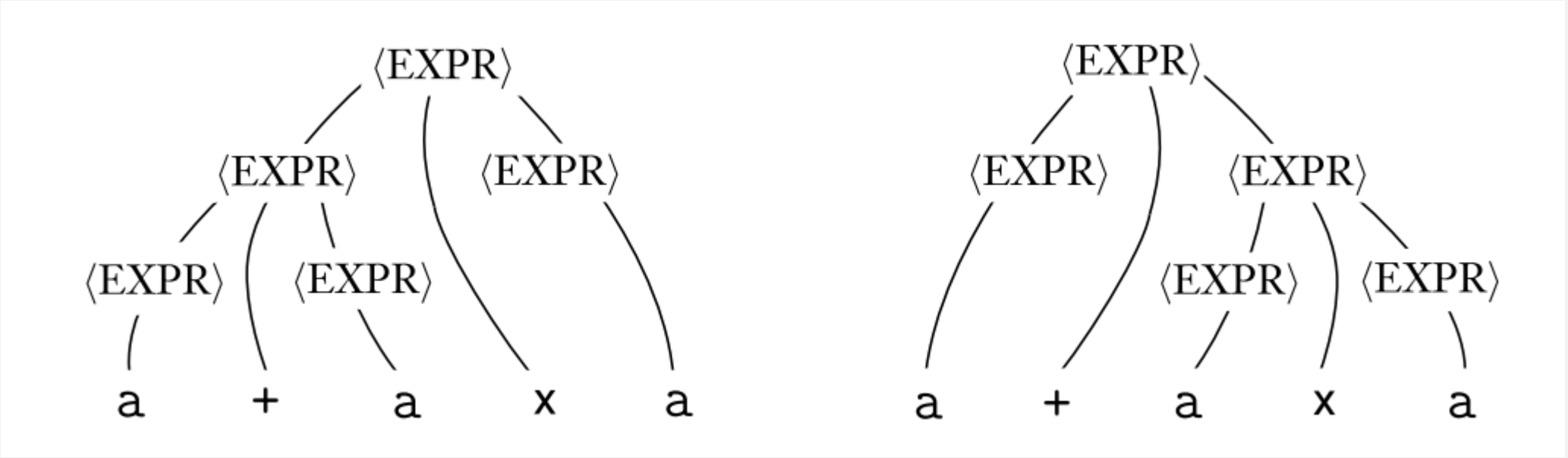
- EXPR =>
- EXPR + TERM =>
- EXPR + TERM x FACTOR =>
- EXPR + TERM x a=>

...

Parses may be ambiguous

grammar G_5:

$$\langle \text{EXPR} \rangle \rightarrow \langle \text{EXPR} \rangle + \langle \text{EXPR} \rangle \mid \langle \text{EXPR} \rangle \times \langle \text{EXPR} \rangle \mid (\langle \text{EXPR} \rangle) \mid \texttt{a}$$



Ambiguity Definition

DEFINITION 2.7

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.

I.e., a string can have multiple parse trees!

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");
```

Ambiguous grammars are bad because they give **multiple meanings** to the same string (program).

But there’s no guaranteed way to create unambiguous grammar (just have to think about it)

Designing Grammars

- Think about what you want to “link” together
- E.g., XML
  - ELEMENT → <TAG>CONTENT</TAG>
  - Start and end tags are “linked”

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

Designing Grammars 2

- Start with small grammars and then combine (just like FSMs)
- "|" = "Or"
  - To create grammar for lang $\{0^n 1^n | n \geq 0\} \cup \{1^n 0^n | n \geq 0\}$
  - First create grammar for lang $\{0^n 1^n | n \geq 0\}$:

$$S_1 \rightarrow 0S_1 1 \mid \varepsilon$$

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

$$S_2 \rightarrow 1S_2 0 \mid \varepsilon$$

  - Then combine:

$$S \rightarrow S_1 \mid S_2$$
$$S_1 \rightarrow 0S_1 1 \mid \varepsilon$$
$$S_2 \rightarrow 1S_2 0 \mid \varepsilon$$

Designing Grammars 3

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

In-class exercise: Designing grammars

alphabet Σ is $\{0,1\}$

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

- S -> 0A0 | 1A1 | ε
- A -> CA | ε
- C -> 0 | 1

Check-in Quiz 10/5

On gradescope

End of Class Survey 10/5

See course website