MA/CSSE 474

Theory of Computation

Summary of regular Language Algorithms

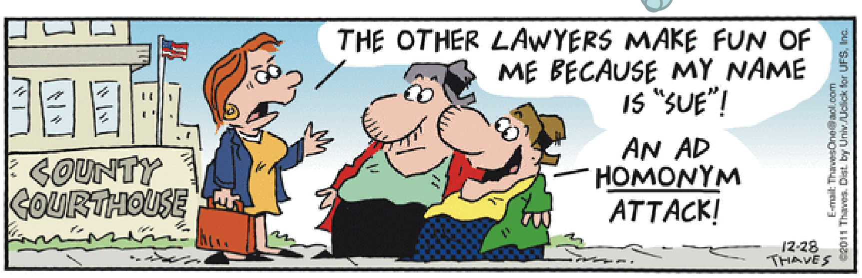
Intro to Grammars

Context-free Grammars (CFG)

Your Questions?

- Previous class days' material
- Reading Assignments
- HW 8 or 9 problems
- Exam 2
- Anything else

This one has so many levels …



Summary of algorithms we have so far

The next few slides are here for reference.
I do not expect to spend class time on them.

You should know how to do all of them, but during class exercises and homework you may simply "call" any of them as part of your decision procedures.

- Operate on FSMs without altering the language that is accepted:
  - *ndfsmtodfsm(M: NDFSM)*
  - *minDFSM (M:DFSM)*
  - *buildFSMcanonicalform(M:FSM)*

Summary of Algorithms

- Compute functions of languages defined as FSMs:
  - Given FSMs M_1 and M_2, construct a FSM M_3 such that $L(M_3) = L(M_2) \cup L(M_1)$.
  - Given FSMs M_1 and M_2, construct a new FSM M_3 such that $L(M_3) = L(M_2)\, L(M_1)$.
  - Given FSM M, construct an FSM M^* such that $L(M^*) = (L(M))^*$.
  - Given a DFSM M, construct an FSM M^* such that $L(M^*) = \neg L(M)$.
  - Given two FSMs M_1 and M_2, construct an FSM M_3 such that $L(M_3) = L(M_2) \cap L(M_1)$.
  - Given two FSMs M_1 and M_2, construct an FSM M_3 such that $L(M_3) = L(M_2) - L(M_1)$.
  - Given an FSM M, construct an FSM M^* such that $L(M^*) = (L(M))^R$.
  - Given an FSM M, construct an FSM M^* that accepts *letsub*($L(M)$).

Algorithms, Continued

- Converting between FSMs and regular expressions:
  - Given a regular expression α, construct an FSM M such that:

    $$L(\alpha) = L(M)$$

  - Given an FSM M, construct a regular expression α such that:

    $$L(\alpha) = L(M)$$

- **Algorithms that implement operations on languages defined by regular expressions**: any operation that can be performed on languages defined by FSMs can be implemented by converting all regular expressions to equivalent FSMs and then executing the appropriate FSM algorithm.

Algorithms, Continued

- Converting between FSMs and regular grammars:
  - Given a regular grammar G, construct an FSM M such that:

    $$L(G) = L(M)$$

  - Given an FSM M, construct a regular grammar G such that:

    $$L(G) = L(M).$$

We have not yet discussed regular grammars.
They are in the reading assignment for tomorrow.
This is here for completeness.

Answering Specific Questions

Given two regular expressions α_1 and α_2, is:

$$(L(\alpha_1) \cap L(\alpha_2)) - \{\varepsilon\} \neq \varnothing?$$

1. From α_1, construct an FSM M_1 such that $L(\alpha_1) = L(M_1)$.
2. From α_2, construct an FSM M_2 such that $L(\alpha_2) = L(M_2)$.
3. Construct M' such that $L(M') = L(M_1) \cap L(M_2)$.
4. Construct M_ε such that $L(M_\varepsilon) = \{\varepsilon\}$.
5. Construct M'' such that $L(M'') = L(M') - L(M_\varepsilon)$.
6. If $L(M'')$ is empty return *False*; else return *True*.

For practice later: Given two regular expressions α_1 and α_2, are there at least 3 strings that are generated by both of them?

Context-Free Grammars

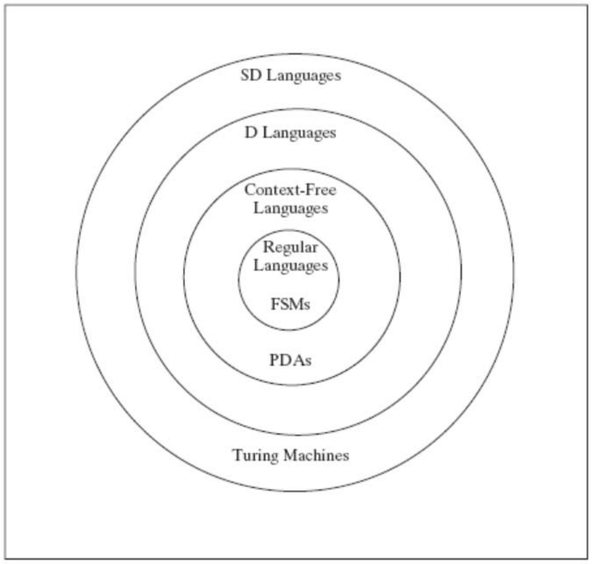
CFG ≡ BNF (mostly)

Chapter 11

For regular languages, we first discussed language recognition (FSM), then language description (reg. exp.)

For context-free languages, we first discuss language generation (CFG), then language recognition (PDA)

Languages and Machines



Rewrite Systems and Grammars

A ***rewrite system*** (a.k.a. ***production system*** or ***rule-based system***) is:

- a list of rules, and
- an algorithm for applying them.

Each rule has a left-hand side (lhs) and a right hand side (rhs)

Example rules:

$S \rightarrow aSb$
$aS \rightarrow \varepsilon$
$aSb \rightarrow bSabSa$

Simple-rewrite algorithm

simple-rewrite(*R*: rewrite system, *w*: initial string) =

1. Set *working-string* to *w*.

2. Until told by *R* to halt do:

   Match the *lhs* of some rule against some part of *working-string*.

   Replace the matched part of *working-string* with the *rhs* of the rule that was matched.

3. Return *working-string*.

***lhs* means "left-hand side"** ***rhs* means "right-hand side"**

An Example

$w = SaS$

Rules:

[1] $S \to aSb$
[2] $aS \to \varepsilon$

- What order to apply the rules?
- When to quit?

String Generation from a Grammar

- Multiple rules may match.

Grammar: $S \rightarrow \mathrm{a}S\mathrm{b}$, $S \rightarrow \mathrm{b}S\mathrm{a}$, and $S \rightarrow \varepsilon$

Derivation so far: $S \Rightarrow \mathrm{a}S\mathrm{b} \Rightarrow \mathrm{aa}S\mathrm{bb} \Rightarrow$

Three choices for the next step:

$S \Rightarrow \mathrm{a}S\mathrm{b} \Rightarrow \mathrm{aa}S\mathrm{bb} \Rightarrow \mathrm{aaa}S\mathrm{bbb}$ (using rule 1),
$S \Rightarrow \mathrm{a}S\mathrm{b} \Rightarrow \mathrm{aa}S\mathrm{bb} \Rightarrow \mathrm{aab}S\mathrm{abb}$ (using rule 2),
$S \Rightarrow \mathrm{a}S\mathrm{b} \Rightarrow \mathrm{aa}S\mathrm{bb} \Rightarrow \mathrm{aabb}$ (using rule 3).

Generating Many Strings

- One rule may match in more than one position in the string.

Grammar: $S \rightarrow \mathrm{a}TT\mathrm{b}$, $T \rightarrow \mathrm{b}T\mathrm{a}$, and $T \rightarrow \varepsilon$

Derivation so far: $S \Rightarrow \mathrm{a}TT\mathrm{b} \Rightarrow$

Two choices for which nonterminal to replace in the next step:

$S \Rightarrow \mathrm{a}\underline{T}T\mathrm{b} \Rightarrow \mathrm{ab}T\mathrm{a}T\mathrm{b} \Rightarrow$
$S \Rightarrow \mathrm{a}\underline{T}T\mathrm{b} \Rightarrow \mathrm{a}T\mathrm{b} \Rightarrow$

Replace the first T

$S \Rightarrow \mathrm{a}T\underline{T}\mathrm{b} \Rightarrow \mathrm{a}T\mathrm{b}T\mathrm{ab} \Rightarrow$
$S \Rightarrow \mathrm{a}T\underline{T}\mathrm{b} \Rightarrow \mathrm{a}T\mathrm{b} \Rightarrow$

Replace the second T

When to Stop

We may stop when:

1. The working string no longer contains any nonterminal symbols (including, when the working string is ε).

In this case, we say that the working string is ***generated*** by the grammar.

Example:

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

When to Stop

May stop when:

2. There are nonterminal symbols in the working string but none of them is the left-hand side of any rule in the grammar.

In this case, we have a blocked or non-terminated derivation but no generated string.

Example:

Rules: $S \rightarrow aSb$, $S \rightarrow bTa$, and $S \rightarrow \varepsilon$

Derivation: $S \Rightarrow aSb \Rightarrow abTab \Rightarrow$ [blocked]

When to Stop

It is possible that neither (1) nor (2) is achieved.

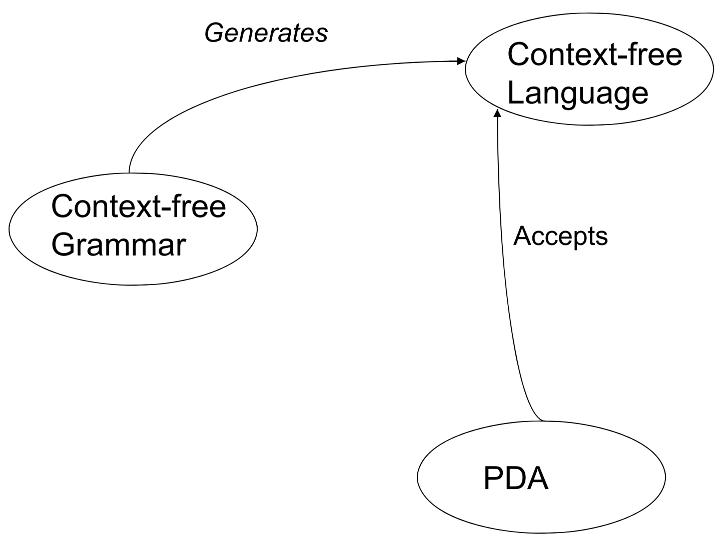
Example:

G contains only the rules $S \rightarrow Ba$ and $B \rightarrow bB$, with *S* the start symbol.

Then all derivations proceed as:

$$S \Rightarrow Ba \Rightarrow bBa \Rightarrow bbBa \Rightarrow bbbBa \Rightarrow bbbbBa \Rightarrow \ldots$$

Context-free Grammars, Languages, and PDAs



Context-free Grammar Formal Definition

A CFG G=(V, Σ, R, S) (Each part is finite)

Σ is the ***terminal alphabet***; it contains the set of symbols that make up the strings in $L(G)$, and

N *(our textbook does not use this name, but I will) is the* ***nonterminal alphabet*****:** a set of working symbols that G uses to structure the language. These symbols disappear by the time the grammar finishes its job and generates a string. **Note:** $\Sigma \cap N = \varnothing$.

Rule alphabet (vocabulary): **V** = $\Sigma \cup N$

- **R**: A finite set of productions of the form $A \rightarrow \beta$, where $A \in N$ and $\beta \in V^*$ **Rules** are also known as **productions**.

G has a unique **start symbol**, **S** $\in N$

Context-Free Rules

No restrictions on the form of the right-hand side.

$S \rightarrow \mathrm{ab}DeFG\mathrm{ab}$

But we require single non-terminal on left-hand side.

$S \rightarrow$

but not $ASB \rightarrow$ or $aS \rightarrow$

Write CFGs that Generate These Languages

A^nB^n

BAL (Balanced Parentheses language)

$\{a^m b^n : m >= n\}$

Formal Definitions: Derivations, Context-free Languages

$x \Rightarrow_G y$ iff $x = \alpha A \beta$

and $A \rightarrow \gamma$ is in R

$y = \alpha \gamma \beta$

$w_0 \Rightarrow_G w_1 \Rightarrow_G w_2 \Rightarrow_G \ldots \Rightarrow_G w_n$ is a **derivation** in G.

Let $\Rightarrow_G{}^*$ be the reflexive, transitive closure of $\Rightarrow_G$.

Then the **language generated by G**, denoted $L(G)$, is:

$\{w \in \Sigma^* : S \Rightarrow_G{}^* w\}$.

A language L is ***context-free*** if there is some context-free grammar G such that $L = L(G)$.

Regular Grammars

A brief side-trip into Chapter 7

Regular Grammars

In a regular grammar, every rule (production) in *R* must have a right-hand side that is:

- ε, or
- a single terminal, or
- a single terminal followed by a single nonterminal.

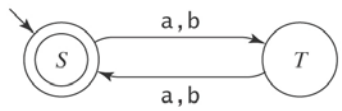
Regular: $S \rightarrow \mathrm{a}$, $S \rightarrow \varepsilon$, and $T \rightarrow \mathrm{a}S$

Not regular: $S \rightarrow \mathrm{a}S\mathrm{a}$ and $S \rightarrow T$

Regular Grammar Example

$L = \{w \in \{a, b\}^* : |w| \text{ is even}\}$

$((aa) \cup (ab) \cup (ba) \cup (bb))^*$



$S \rightarrow \varepsilon$
$S \rightarrow aT$
$S \rightarrow bT$
$T \rightarrow a$
$T \rightarrow b$
$T \rightarrow aS$
$T \rightarrow bS$

Derive **abbb** from this grammar

Regular Languages and Regular Grammars

Theorem: A language is regular iff it can be defined by a regular grammar.

Proof: By two constructions, one for each direction.

Regular Languages and Regular Grammars

Regular grammar* $\rightarrow$ *FSM:

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

1. In M, Create a separate state for each nonterminal in V.
2. Start state is the state corresponding to S .
3. If there are any rules in R of the form $X \rightarrow a$, for some $a \in \Sigma$, create a new state labeled #.
4. For each rule of the form $X \rightarrow a\ Y$, add a transition from X to Y labeled a.
5. For each rule of the form $X \rightarrow a$, add a transition from X to # labeled a.
6. For each rule of the form $X \rightarrow \varepsilon$, mark state X as accepting.
7. Mark state # as accepting.

$S \rightarrow bS$, $S \rightarrow aT$
$T \rightarrow aS$, $T \rightarrow b$, $T \rightarrow \varepsilon$

FSM* $\rightarrow$ *Regular grammar: Similar.
Essentially reverses this procedure.