MA/CSSE 474
Theory of Computation

Algorithms and Decision Procedures for Regular Languages

Intro to Context-free Grammars

Decision Procedures

A decision procedure is an algorithm whose result is a Boolean value. It must:

- Halt
- Be correct

Usually a decision procedure has one or more parameters
Zero-parameter decision procedures are not very interesting!

Decision Procedures for Regular Languages

- Given an FSM *M* and a string *w*, does *M* accept *w*?
- Given a regular expression α and a string *w*, does α generate *w*?
- Given an FSM *M*, is *L*(*M*) empty**?**
- Given an FSM *M*, is $L(M) = \Sigma_M^*$?
- Given an FSM *M*, is *L*(*M*) finite?
- Given an FSM *M*, is *L*(*M*) infinite?
- Given two FSMs M_1 and M_2, are they equivalent?
- Given an FSM *M*, is M minimal?

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?

Summary of Algorithms

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

- Operate on FSMs without altering the language that is accepted:
  - *Ndfsmtodfs*
  - *MinDFSM*

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

Algorithms: Decision Procedures

- Decision procedures that answer questions about languages defined by FSMs:
  - Given an FSM M and a string s, decide whether s is accepted by M.
  - Given an FSM M, decide whether $L(M)$ is empty.
  - Given an FSM M, decide whether $L(M)$ is finite.
  - Given two FSMs, M_1 and M_2, decide whether $L(M_1) = L(M_2)$.
  - Given an FSM M, is M minimal?

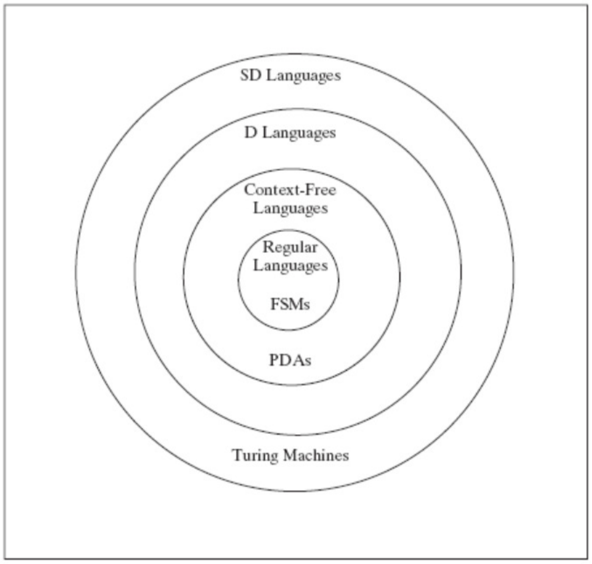
- Decision procedures that answer questions about languages defined by regular expressions: Again, convert the regular expressions to FSMs and apply the FSM algorithms.

Context-Free Grammars

CFG ≡ BNF (mostly)

Chapter 11

Languages and Machines



Rewrite Systems and Grammars

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

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

Each rule has a left-hand side and a right hand side.

Example rules:

$S \rightarrow \mathrm{a}S\mathrm{b}$
$\mathrm{a}S \rightarrow \varepsilon$
$\mathrm{a}S\mathrm{b} \rightarrow \mathrm{b}S\mathrm{ab}S\mathrm{a}$

Simple-rewrite

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*.

A Rewrite System Formalism

A rewrite system formalism specifies:

- The form of the rules

- How *simple-rewrite* works:
  - How to choose rules?
  - When to quit?

An Example

$w = S\mathrm{a}S$

Rules:

[1] $S \rightarrow \mathrm{a}S\mathrm{b}$

[2] $\mathrm{a}S \rightarrow \varepsilon$

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

Rule Based Systems

- Expert systems
- Cognitive modeling
- Business practice modeling
- General models of computation
- **Grammars**

Grammars Define Languages

A grammar, G, is a set of rules that are stated in terms of two alphabets:

• a ***terminal alphabet***, Σ, that contains the symbols that make up the strings in *L*(*G*), and

•a ***nonterminal alphabet***,N, the elements of which will function as working symbols that will be used while the grammar is operating. These symbols will disappear by the time the grammar finishes its job and generates a string. **Note:** $\Sigma \cap N = \varnothing$

A grammar has a unique start symbol, often called *S*.

Using a Grammar to Derive a String

Simple-rewrite (*G*, *S*) will generate the strings in *L*(*G*).

We will use the symbol ⇒ to indicate steps in a derivation.

In our example:

[1] $S \rightarrow \mathrm{a}S\mathrm{b}$

[2] $\mathrm{a}S \rightarrow \varepsilon$

A derivation could begin with:

$S \Rightarrow \mathrm{a}S\mathrm{b} \Rightarrow \mathrm{aa}S\mathrm{bb} \Rightarrow \ldots$

Generating Many Strings

• Multiple rules may match.

Given: $S \rightarrow \mathtt{a}S\mathtt{b}$, $S \rightarrow \mathtt{b}S\mathtt{a}$, and $S \rightarrow \varepsilon$

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

Three choices at the next step:

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

Generating Many Strings

• One rule may match in more than one way.

Given: $S \rightarrow \mathtt{a}TT\mathtt{b}$, $T \rightarrow \mathtt{b}T\mathtt{a}$, and $T \rightarrow \varepsilon$

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

Two choices at the next step:

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

When to Stop

May stop when:

1. The working string no longer contains any nonterminal symbols (including, when it 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 in a substring that 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 $S \rightarrow \varepsilon$

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

When to Stop

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

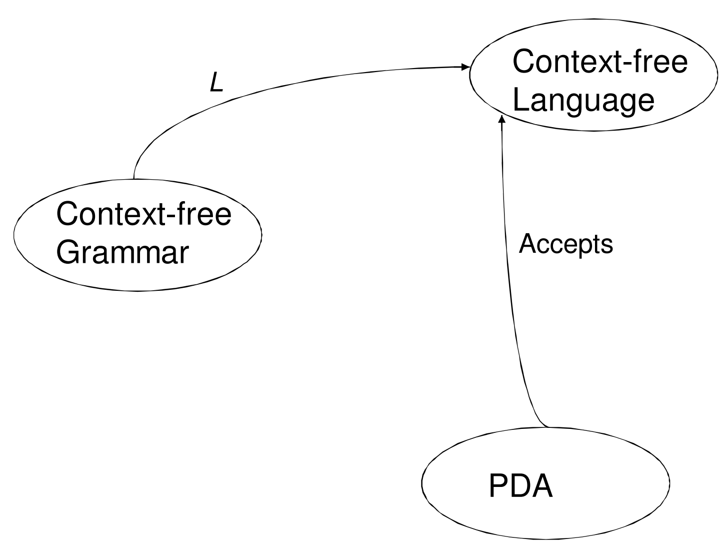
Example:

G contains only the rules $S \rightarrow B\mathtt{a}$ and $B \rightarrow \mathtt{b}B$, with S the start symbol.

Then all derivations proceed as:

$$S \Rightarrow B\mathtt{a} \Rightarrow \mathtt{b}B\mathtt{a} \Rightarrow \mathtt{bb}B\mathtt{a} \Rightarrow \mathtt{bbb}B\mathtt{a} \Rightarrow \mathtt{bbbb}B\mathtt{a} \Rightarrow \ldots$$

Context-free Grammars, Languages, and PDAs



Context-Free Grammars

No restrictions on the form of the right hand sides.

$S \rightarrow \mathtt{abD}\mathtt{e}FG\mathtt{ab}$

But require single non-terminal on left hand side.

$S \rightarrow$

but not $ASB \rightarrow$

$a^n b^n$

Balanced Parentheses language

$a^m b^n : m >= n$

Context-Free Grammars

A context-free grammar G is a quadruple,
(V, Σ, R, S), where:

- V is the rule alphabet, which contains nonterminals and terminals.
- Σ (the set of terminals) is a subset of V,
- R (the set of rules) is a finite subset of $(V - \Sigma) \times V^*$,
- S (the start symbol) is an element of $V - \Sigma$.

Example:

$(\{S, \mathtt{a}, \mathtt{b}\}, \{\mathtt{a}, \mathtt{b}\}, \{S \rightarrow \mathtt{a}\ S\ \mathtt{b}, S \rightarrow \varepsilon\}, S)$

Rules are also known as **productions**.