CS3110 Formal Language and Automata

Tingting Chen
Computer Science
Cal Poly Pomona

Topic of course

- What are the fundamental capabilities and limitations of computers?
- To answer this, we will study abstract mathematical models of computers
- These mathematical models abstract away many of the details of computers to allow us to focus on the essential aspects of computation
- It allows us to develop a mathematical theory of computation

Review of Set theory

Can specify a set in two ways:

- list of elements: $A = \{6, 12, 28\}$
- characteristic property: $B = \{x \mid x \text{ is a positive, even integer}\}$

Set membership: $12 \in A$, $9 \notin A$

Set inclusion: $A \subseteq B$ (A is a subset of B)

$A \subset B$ (A is a proper subset of B)

Set operations:

union: $A \cup \{9, 12\} = \{6, 9, 12, 28\}$

intersection: $A \cap \{9, 12\} = \{12\}$

difference: $A - \{9, 12\} = \{6, 28\}$

Set theory (continued)

Another set operation, called "taking the complement of a set", assumes a **universal set**.

Let U = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} be the universal set.
Let A = {2, 4, 6, 8}
Then $\overline{\mathrm{A}}$ = U - A = {0, 1, 3, 5, 7, 9}

The **empty set**: $\varnothing = \{\}$

Set theory (continued)

The **cardinality** of a set is the number of elements in a set.

Let $S = \{2, 4, 6\}$

Then $|S| = 3$

The **powerset** of S, represented by 2^S, is the set of all subsets of S.

$2^S = \{\{\}, \{2\}, \{4\}, \{6\},\{2,4\}, \{2,6\}, \{4,6\}, \{2,4,6\}\}$

The number of elements in a powerset is $|2^S| = 2^{|S|}$

What does the title of this course mean?

- Formal language
  - a subset of the set of all possible strings from a set of symbols
  - example: the set of all syntactically correct C programs
- Automata
  - abstract, mathematical model of computer
  - examples: finite automata, pushdown automata, Turing machine, RAM, many others

Languages

- A language is a set of strings.
- **String**: A sequence of letters.
  – Example: “cat”, “dog”, “house”
- Defined over an alphabet:
  – $\Sigma = \{a, b, c, ..., z\}$.

Alphabets and Strings

- Let's use a small alphabet: $\Sigma = \{a, b\}$.
- Strings:

  a,

  ab,

  abba,

  baba,

  aaabbbaabab …
- We often use **string variables**:

  u = ab

  v = bbbaaa

  …

String Operations

$w = a_1 a_2 \cdots a_n$

$v = b_1 b_2 \cdots b_n$

- Concatenation:

$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_n$

- Reverse:

$w^R = a_n \cdots a_2 a_1$

- String Length

$|w| = n$

$|abba| = 4$, $|a| = 1$

$|wv| = |w| + |v|$

Empty String

- The **empty string**, denoted λ , has some special properties:
- $|\lambda| = 0$
- For any string w, $\lambda w = w \lambda = w$

- If *w* is a string, then w^n stands for the string obtained by repeating *w* *n* times.
  - If w= ab, then w^3=ababab.
  - $w^0 = \lambda$

Substring

- Substring of string: a subsequence of consecutive characters

String	*Substring*
abbab	ab
abbab	abba
abbab	b
abbab	bbab

Prefix and Suffix

- For any string w, w can be written as w=uv
- u is a prefix, and v is a suffix.

- abbab

Prefix	*Suffix*
λ	abbab
a	bbab
ab	bab
a<u>bb</u>	ab
abba	b
abbab	λ

The * operation on alphabet

- Σ^* : the set of all possible strings from alphabet Σ .
  - $\Sigma = \{a, b\}$
  - $\Sigma^* = \{\lambda, a, b, aa, bb, ab, ba, aaa, aab, \ldots\}$
  - $\Sigma^+ = \Sigma^* - \{\lambda\}$
  - $\Sigma^+ = \{a, b, aa, bb, ab, ba, aaa, aab, \ldots\}$

Languages

- A language is a subset of Σ^*.
  - Example: $\Sigma = \{a, b\}$
  - $\Sigma^* = \{\lambda$, a, b, aa, bb, ab, ba, aaa, aab,…$\}$
  - Languages: $\{\lambda\}$

    {a, aa, aab}

    $\{\lambda$, abba, aab, ab,aaaaa$\}$

Another example: An infinite language $L = \{a^n b^n \ : n \geq 0\}$

Operations on languages

Set operations:

Union: $L_1 \cup L_2$ $\{a, ab, aaa\} \cup \{ab, bb\} = \{a, ab, aaa, bb\}$

Intersection: $L_1 \cap L_2$ $\{a, ab, aaa\} \cap \{ab, bb\} = \{ab\}$

Difference: $L_1 - L_2$ $\{a, ab, aaa\} - \{ab, bb\} = \{a, aaa\}$

Complement: $\overline{L} = \Sigma^* - L$

$$\overline{\{a, ba\}} = \{\lambda, b, ab, aa, bb \ldots\}$$

Operations on languages

String operations:

"Reverse of language" : $L^R = \{w^R \mid w \in L\}$

$\{a, ab, abaa\}^R = \{a, ba, aaba\}$

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

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

"concatenation of languages" $L_1 L_2 = \{xy \mid x \in L_1, y \in L_2\}$

$\{a, ab, ba\}\{b, aa\} = \{ab, aaa, abb, abaa, bab, baaa\}$

L^n

$\{a, b\}^3 = \{a, b\}\ \{a, b\}\ \{a, b\}$

$= \{aaa, aab, aba, abb, baa, bab, bba, bbb\}$

$L^0 = \{\lambda\}$

Practice problem: $L = \{a^n b^n : n \geq 0\}$, $L^2 = ?$

Star-Closure (Kleene *) and Positive Closure

Star-Closure Definition: $L^* = L^0 \cup L^1 \cup L^2 \ldots$

Example: $\{a, bb\}^*$

$= \{\lambda, a, bb, aa, bbbb, abb, bba, aaa, aabb, abba, abbbb, \ldots\}$

Positive-Closure Definition: $L^+ = L^1 \cup L^2 \ldots$

Example: $\{a, bb\}^+$

$= \{a, bb, aa, bbbb, abb, bba, aaa, aabb, abba, abbbb, \ldots\}$

Grammars

Grammars are used to generate languages.

A grammar G is defined as a quadruple:

G = (V, T, S, P)

Where V is a finite set of objects called variables

T is a finite set of objects called terminal symbols

$S \in V$ is a special symbol called the Start symbol

P is a finite set of productions or "production rules"

Sets V and T are nonempty and disjoint

Production Rules

Production rules have the form:

$$x \to y$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$

Given a string of the form

$$w = uxv$$

and a production rule

$$x \to y$$

we can apply the rule,

$$uxv \to uyv.$$

Given $z = uyv$, we can say that $w \Rightarrow z$

Read as "w derives z", or "z is derived from w"

String Derivation

If $u \Rightarrow v$, $v \Rightarrow w$, $w \Rightarrow x$, $x \Rightarrow y$, and $y \Rightarrow z$, then we say:

$$u \overset{*}{\Rightarrow} z$$

This says that u derives z in an unspecified number of steps.

Relationship between a language and a grammar

What is the relationship between a language and a grammar?

Let G = (*V, T, S, P*), the set

$$L(G) = \{w \in T^* : S \stackrel{*}{\Rightarrow} w\}$$

is the language generated by G.

Example

Consider the grammar G = (*V, T, S, P*), where:

V = {S}

T = {a, b}

S = S,

P =

$S \to aSb$
$S \to \lambda$

What are some of the strings in this language?

Example

$S \Rightarrow aSb \Rightarrow ab$

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

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$

It is easy to see that the language generated by this grammar is:

$L(G) = \{a^n b^n : n \geq 0\}$

From Languages to Grammars

Find a grammar that generates: $L = \{a^n b^{n+1} : n \geq 0\}$

So the strings of this language will be:

b (0 a's and 1 b)
abb (1 a and 2 b's)
aabbb (2 a's and 3 b's) . . .

We observe the pattern in the strings, and find that they can be considered as the concatenation of two parts: $a^n b^n$ and b.

Determine Production Rules

One solution is that we create another variable, A, to stand for the $a^n b^n$, and introduce this production rule.

$$S \rightarrow Ab$$

Since “b” is included in the language, we need to have another production rule.

$$A \rightarrow \lambda$$

Now we need to generate the other part of the string, the $a^n b^n$ part, from A.
$a^n b^n$ has equal number of a’s and b’s. So one production rule can be

$$A \rightarrow aAb$$

So, here are our rules:

$$S \rightarrow Ab$$
$$A \rightarrow aAb$$
$$A \rightarrow \lambda$$

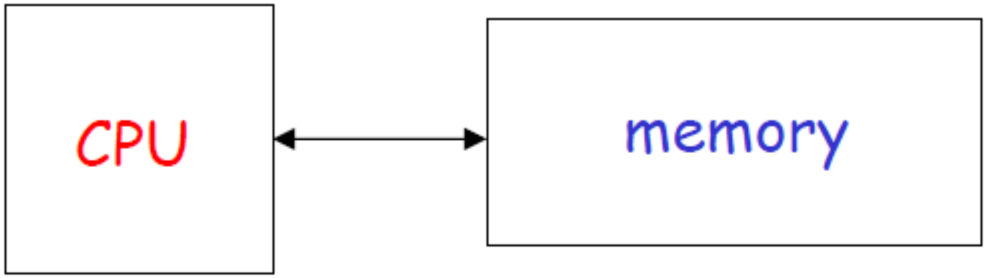
The $S \rightarrow Ab$ rule creates a single b terminal on the right, preceded by other strings (including possibly the empty string) on the left.

The $A \rightarrow \lambda$ rule allows the single b string to be generated.

The $A \rightarrow aAb$ rule and the $A \rightarrow \lambda$ rule allows ab, aabb, aaabbb, etc. to be generated on the left side of the string.

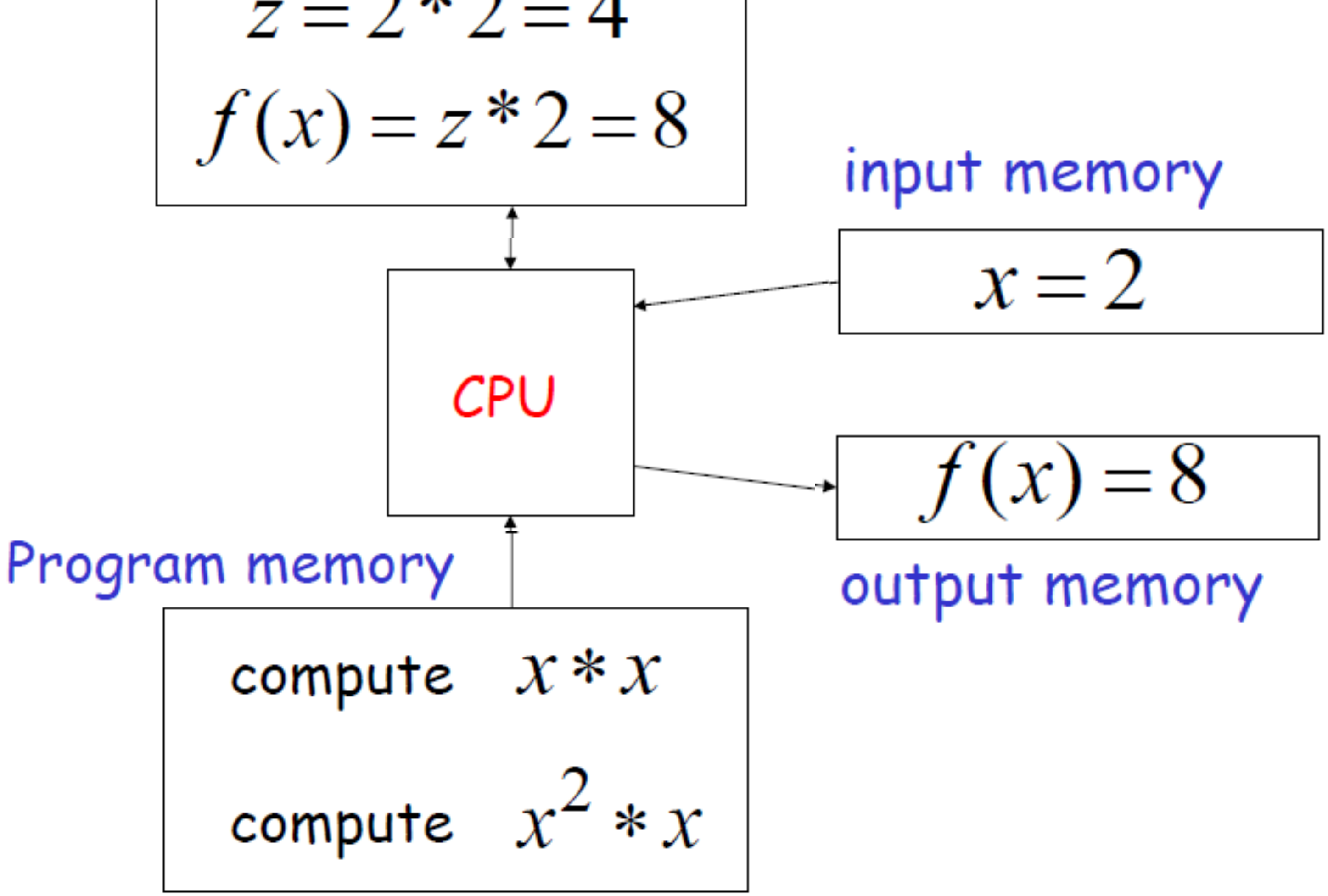
Note: it is not the only set of production rules.

Computation

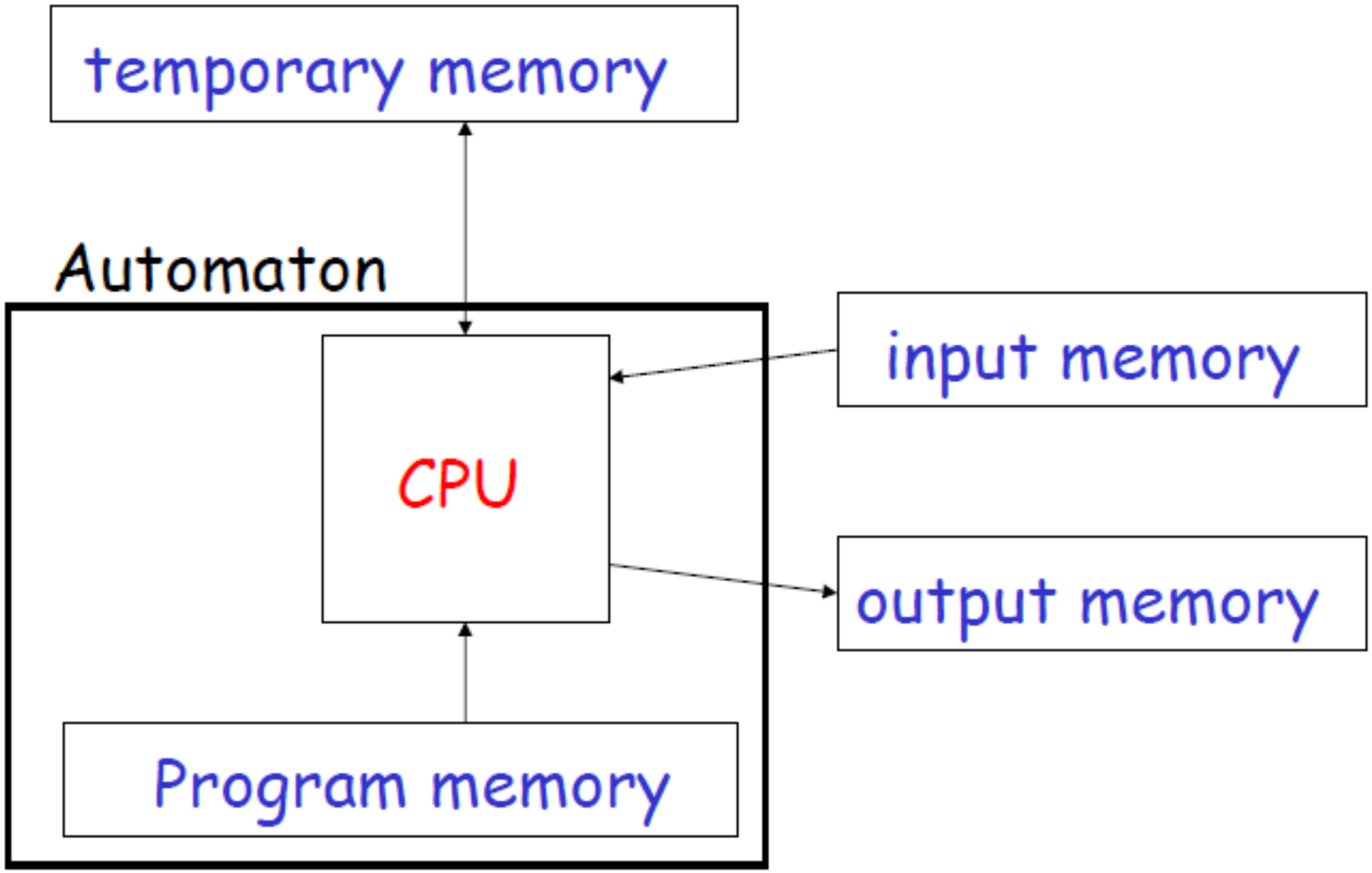


Computation -- Example

Example: $f(x) = x^3$



Automaton


temporary memory
Automaton
CPU
input memory
output memory
Program memory

Language-recognition problem

- Different Automata can be designed for different problems (with different program memory).
- There are many types of computational problem. We will focus on the simplest, called the "language-recognition problem."
- Given a string, determine whether it belongs to a language or not. (Practical application for compilers: Is this a valid C++ program?)
- We study simple models of computation, i.e., automata, and measure their computational power in terms of the class of languages they can recognize.

Automata – Related Parts

- Input File
- Control Unit (with finite states)
- Temporary Storage
- Output

Different Kinds of Automata

- Automata are distinguished by the temporary memory.

Finite Automata: no temporary memory

Applications: text-editing software: search and replace; many forms of pattern-recognition (including use in WWW search engines); sequential circuit design (including vending machines)…

Pushdown Automata: Stack (infinite memory)

Applications: compilers: parsing computer programs

Turing Machines: random access memory

Applications: any algorithm …

Automata, languages, and grammars

- In this course, we will study the relationship between automata, languages, and grammars
- Recall that a formal language is a set of strings over a finite alphabet
- Grammars are used to *generate* languages
- Automata are used to *recognize* languages