Mathematics for Computing Science (CS2013)

Formal Languages: Preliminaries

Formal Languages

- We have heard of or have used many languages
    - English, Polish, French, etc.
    - Java, Python, C, C#, etc.
- These are very different kinds of languages
- What is a language?
    - How can we decide if a sentence is indeed from a language?
    - Examples in English:
        - “Large red vans go fast.” and “Colourless green ideas sleep furiously.”
        - “Vans red large go fast.”

Formal Languages

- Formal language has a well-defined test if a sentence belongs to it or not
- The test is based solely on the form of the sentence
  - No “meaning” involved
  - “Colourless green ideas sleep furiously” makes no sense, but it is a grammatically correct English sentence

Symbols, Alphabet and Strings

- **Symbols**: atomic (basic) components of a formal language
  - Examples: digits 0-9, small letters a-z, special characters £, %, *, etc.
  - They cannot be split apart into smaller sub-components
  - Akin to characters of a keyboard
- **Alphabet**: a *finite set* of symbols
  - Examples: {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}, {a, b, c, d, e, …, z}
  - We denote sets by italicised capital letters *A*, *B*, etc.
- **String** over an alphabet *T*: a finite sequence of symbols from T
  - Example strings over *T* = {0, 1} are 0011, 101, 1100, etc.
- **Empty string** λ (lambda): a special string with no symbols
  - Some textbooks also use ε (epsilon) to represent the empty string

Definitions

- We refer to strings as w (for "word")
  - We also use v to refer generically to a string
  - We may use subscripts (e.g., w_1, w_i, v_2) to differentiate strings
- **Length** of a string w, denoted as $|w|$, is the number of symbols w has
  - Examples: $|010| = 3$, $|abba| = 4$
  - Note that $|\lambda| = 0$
- Strings w and v are **equal**, denoted as $w = v$, if they have exactly the same sequence of symbols
  - Examples: given $w_1 = 010$, $v = 010$, $w_1 = v$
- Strings w and v are **different**, denoted as $w \neq v$, if they are not equal
  - Examples: given $w_2 = 011$, $v = 010$, $w_2 \neq v$

Definitions (cont'd)

- **Concatenation** of two strings w and v, denoted wv, is the sequence of symbols in w followed by the sequence of symbols in v
  - If w = abb and v = bab, then wv = abbbab, vw = bababb
  - Notice: $w\lambda = \lambda w = w$ (empty string is "identity element" of concatenation)
- Concatenation is **not commutative**
  - wv not necessarily equal to vw
- Concatenation is **associative**
  - It is always the case that $w(vu) = (wv)u$

Definitions (cont'd)

- u is a **substring** of w if there are v_1 and v_2 such that $w = v_1uv_2$
  - Example: $u = 10$ is a substring of $w = 01101$ ($v_1 = 01$, $v_2 = 1$)
  - Note: empty string λ is a substring of any string
  - Important: v_1 and/or v_2 can be the empty string λ (special cases – see below)
- **Prefix**: u is a prefix of w if it is a **substring** of w and $v_1 = \lambda$
  - Example: $u = 011$ is a prefix of $w = 01101$ ($v_1 = \lambda$, $v_2 = 01$)
- **Proper prefix**: u is a proper prefix of w if $w = uv_2$ and $w \neq u$
  - Substring v_2 cannot be λ
- **Suffix**: u is a suffix of w if it is a **substring** of w and $v_2 = \lambda$
  - Example: $u = 101$ is a suffix of $w = 01101$ ($v_1 = 01$, $v_2 = \lambda$)
- **Proper suffix**: u is a proper suffix of w if $w = v_1u$ and $w \neq u$
  - Substring v_1 cannot be λ

Definitions (cont'd)

- T^* is the infinite **set of all strings** over alphabet T
  - For $T = \{0, 1\}$, $T^* = \{\lambda, 0, 1, 00, 11, 01, 10, 000, \ldots\}$
- $T^+ = T^* - \{\lambda\}$ is the **set of all strings** over alphabet T of size 1 or more
- For any symbol $a \in T$, a^n is the string of n a's concatenated
  - $0 \in \{0, 1\}$, $0^3 = 000$, $0^4 = 000$
  - We also use $a^* = \{\lambda, a, aa, aaa, \ldots\}$ (all strings of a's)
  - We also use $a^+ = \{a, aa, aaa, \ldots\}$ (all strings of a's of size 1 or more)
  - Important: $a^n a^m = a^{n+m}$
- A **language over alphabet** T is any set of strings using symbols from T
  - Also called a T-language, or simply a language (when context makes T clear)
  - For $T = \{0, 1\}$, $L = \{\lambda, 0, 1, 0011, 0110\}$ is a T-language
  - L is a T-language if, and only if, $L \subseteq T^*$

Language Operations

- Languages are sets
  - All set operations also applicable to languages
- Let A and B be languages over an alphabet T
  - $A \cup B = \{w | w \in A$ **or** $w \in B\}$ (**set union**; strings in A **or** B)
  - $A \cap B = \{w | w \in A$ **and** $w \in B\}$ (**set intersection**; strings in A **and** B)
  - $\overline{A} = \{w | w \in T^*$ **and** $w \notin A\}$ (**set complement**; strings in T^* **and not** in A)
  - $AB = \{uv | u \in A$ **and** $v \in B\}$ (**set concatenation**; part from A and part from B)
- Language operations with various properties:
  - $A(B \cup C) = AB \cup AC$
  - $A(B \cap C) = AB \cap AC$

Language Operations (Cont'd)

- $A^n = AAA\ldots A$, that is A concatenated with itself n times
  - In special, $A^0 = \{\lambda\}$
- We thus have
  - $A^* = A^0 \cup A^1 \cup A^2 \cup \ldots$
  - $A^+ = A^1 \cup A^2 \cup A^3 \cup \ldots$
  - $(A^*)^* = A^*$ (idempotence of *)
- A^* is the **Kleene closure** of A

Orderings

- $T = \{a_1, a_2, \ldots\}$ an alphabet with an ordering $\succ$ over its symbols, $a_i \succ a_j$
  - Remember: elements of a set have no order
  - We formally represent order via a relation $\succ$

Strings over T ordered in two ways:

1. Dictionary order take into account $\succ$ only
2. Lexical order take into account length and $\succ$

Orderings

- $T = \{a_1, a_2, \ldots\}$ an alphabet with an ordering $\succ$ over its symbols, $a_i \succ a_j$

1. Dictionary order:
   - Strings beginning with a_i are ordered before strings beginning a_j, if $a_i \succ a_j$.
   - Within groups of strings beginning with the same symbol, strings are ordered by their second symbol, etc.
   - λ is always the first string
   - To allow comparison, if strings have different lengths, then fill up the shorter one with λ's on the right

Orderings

- $T = \{a_1, a_2, \ldots\}$ an alphabet with an ordering $\succ$ over its symbols, $a_i \succ a_j$

2. Lexical order:
    - Strings are ordered by their length, with the shortest first
    - Within groups of strings of the same length, strings are ordered in dictionary order
    - λ is always the first string

Specifying Languages

Suppose an alphabet T and the following T-languages

- $L_1 = \{a^n | n = 1, 2, 3, \ldots\}$, for $a \in T$ – what's in L_1?
- $L_2 = \{a^n | n = 1, 4, 6, 9, 16, \ldots\}$, for $a \in T$ – what's in L_2?
- $L_3 = \{a^n | n = 1, 4, 9, 48, \ldots\}$, for $a \in T$ – what's in L_3?

Problem:

- Devise a clear and precise way to define infinite languages

Languages and “machines”

- “Machines”: rudimentary and very abstract kinds of computers
  - No hardware/software
  - Explained/formalised via mathematics
- Languages represent a problem; machines find solutions
- There are different types of questions:
  - Is 234456788 a prime number? (yes/no question or decision problem)
  - What is 386^{677}? (function problem)
  - What is the best move of a chess game? (find one from many options)
  - How to go from Aberdeen to Edinburgh? (how to achieve a result)

Languages and “machines” (cont’d)

- Some of these problems are very hard
  - Some solutions may take too long (centuries even with powerful computers)
  - Some do not have a solution!
- Formal languages are useful abstractions of actual problems
  - Compact description (no need for pages and pages of text, diagrams, etc.)
  - Unambiguous (it’s a mathematical formulation)
  - No irrelevant details
  - Capture many instances of real-life problems
- Study problems abstractly
  - We can see solutions we might otherwise miss
  - We can see equivalent/related problems

Recognising Languages

- Problem of defining languages: how we could **recognise them**
- Recognise a language:
  - Given a language description L and a string w find out if $w \in L$
- Problem statement:

"Is there a method of **recognising** infinite languages?"

- Alternative formulation:
  - Input: description of an infinite language L and a string w
  - Devise an algorithm (an effective procedure) to find out if $w \in L$
    - The algorithm should always stop (termination)
    - The algorithm should always give the correct answer (correctness)
    - The algorithm should give the answer in feasible time (complexity)

Recognising Languages (cont'd)

Our approach:

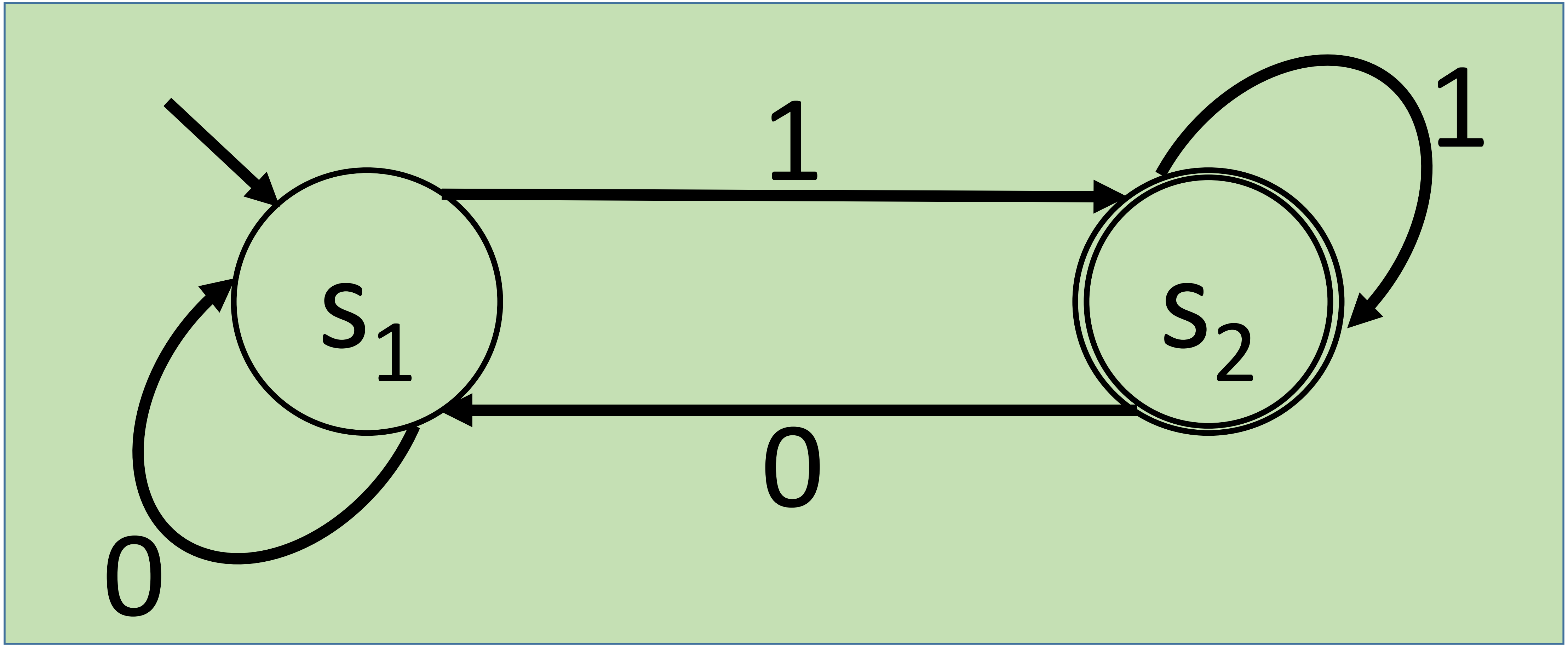
- Devise an abstract machine specific to a language description L
- The machine takes as input a candidate string w and produces an answer yes ($w \in L$) or no ($w \notin L$)
- These machines are called **finite state automata**
  - Also called finite automata or **finite state machines** (FSM)

Finite (State) Automata

- Very simple kind of computer
  - Limited memory and simple operation
  - Many controllers (door, central heating, etc.) use automata, though
- Intuitively:
  - FSA has a finite number of states (hence the “finite state”)
  - It receives individual symbols of a string as input, one at a time
  - Control of execution moves from one state to another, following transitions
  - Each transition (edge) has a label, indicating the value the current input is for that edge to be followed
  - It has a start state, and one or more final states
  - When we run out of symbols (string comes to an end) we stop
  - If we stop in a final state the string is accepted, otherwise it is rejected

Finite (State) Automata (Cont'd)

- FSA to recognise all strings from $T = \{0, 1\}$ ending with 1



Summary

- Formal languages
  - What they are
  - Why study them
  - Operations on formal languages
  - How to describe formal languages
  - Languages and machines
- Finite state automata (FSA)
  - Informal presentation of FSAs

Further reading

- Chapter 0 "Introduction to the theory of computation", by Michael Sipser (there are copies in the library)
- Chapter 1 of "An Introduction to Formal Languages and Automata", by Peter Linz (PDF available on-line)