THE UNIVERSITY OF
NEWCASTLE
AUSTRALIA

FACULTY OF
ENGINEERING AND
BUILT ENVIRONMENT

www.newcastle.edu.au

Theory of Computation
Week 2

Much of the material on this slides comes from the recommended textbook by Elaine Rich

Announcement

Weekly Quiz

- Weekly quiz will be released every Monday (5PM)
- Two weeks to complete the quiz
- You can have two attempts
- All quizzes contribute to to 5% grade

Detailed content

Weekly program

- ✓ Week 1 – Background knowledge revision: logic, sets, proof techniques

Week 2 – Languages and strings. Hierarchies. Computation. Closure properties

- ❑ Week 3 – Finite State Machines: non-determinism vs. determinism
- ❑ Week 4 – Regular languages: expressions and grammars
- ❑ Week 5 – Non regular languages: pumping lemma. Closure
- ❑ Week 6 – Context-free languages: grammars and parse trees
- ❑ Week 7 – Pushdown automata
- ❑ Week 8 – Non context-free languages: pumping lemma and decidability. Closure
- ❑ Week 9 – Decidable languages: Turing Machines
- ❑ Week 10 – Church-Turing thesis and the unsolvability of the Halting Problem
- ❑ Week 11 – Decidable, semi-decidable and undecidable languages (and proofs)
- ❑ Week 12 – Revision of the hierarchy. Safety-critical systems
- ❑ Week 13 – Extra revision (if needed)

Week 02 Lecture Outline

Languages and strings, Hierarchies, Computation, Closure properties

- ❑ Alphabet, Strings,
- ❑ Function and Relations on Strings
- ❑ Languages
- ❑ Languages are sets
- ❑ Functions on Languages
- ❑ Decision Problems
- ❑ Power of Encoding
- ❑ Casting Problems as Decision Problems
- ❑ Rule of Least Power
- ❑ Decision procedures
- ❑ Nondeterminism
- ❑ Functions on languages (programs that operate on other programs)

Week 02 Videos

You already know:

- ❑ Definitions:
  - ❑ Symbols
  - ❑ Alphabet ∑
  - ❑ Strings
  - ❑ All Possible Strings ∑*
  - ❑ Languages
- ❑ String Operations:
  - ❑ Length
  - ❑ Reverse
  - ❑ Concatenation
  - ❑ Replication
- ❑ Language Operations:
  - ❑ Set operations: ∪, ∩, ¬, \ or −
  - ❑ Concatenation
  - ❑ Reversal
  - ❑ Replication
  - ❑ Kleene star and plus

- ❑ Decision Problem
- ❑ Decision Procedure
- ❑ Concept of Determinism VS Non-determinism

Videos to watch before lecture

Additional videos to watch for this week

STRINGS

- An **alphabet** (Σ) is a finite set of **symbols** (or **characters**)
  - Σ = {0, 1} (binary alphabet)
  - ASCII
- A **string** is a finite sequence of symbols chosen from some alphabet Σ
  - 01101
  - *abracadabra*

STRINGS

- ε is the empty string.
- Σ* is the set of all possible strings over an alphabet Σ.

Alphabet name	*Alphabet symbols*	*Example strings*
The English alphabet	{a, b, c, …, z}	ε, aabbcg, aaaaa
The binary alphabet	{0, 1}	ε, 0, 001100
A star alphabet	{★, ✪, ☆, ✫, ✡, ✰}	ε, ✪✪, ✪✫✫✰★✰
A music alphabet	{𝅝, 𝅗𝅥, ♩, ♪, 𝅘𝅥𝅯, 𝅘𝅥𝅰, ●}	ε, 𝅝|𝅗𝅥𝅗𝅥|𝅗𝅥♩♩|

STRINGS
Functions on Strings

- **Length**: The length of a string *s,* which we will write as $|s|$ is the number of symbols in *s*.
  - $|\varepsilon| = 0$
  - $|101101| = 6$

- **Concatenation**: The concatenation of two strings *s* and *t*, written $s||t$ or simply *st*, is the string formed by appending *t* to *s*.
  - *s* =good and *t*=bye, *st*=goodbye.
  - $|st| = |s| + |t|$

STRINGS
Functions on Strings

- **Replication**. For each string w and natural number *i*, the string w^i is defined as:
  - $w^0 = \varepsilon$
  - $w^{i+1} = w^i w$
- Examples:
  - $a^3b^2 =$ aaabb
  - $a^0b^3 =$ bbb
  - $(ab)^2 =$ abab

STRINGS
Functions on Strings

- **Reversal**: For each string w, the reverse of w, which we will write w^R is defined as:

  - If $|w|=0$ then $w=w^R= \varepsilon$
  - If $|w|\geq 1$ then $\exists a \in \Sigma$ and $\exists u \in \Sigma^*$ such that $w=ua$. Then define $w^R = au^R$

Theorem: If w and x are strings, then $(w\,x)^R = x^R\,w^R$.

Proof: By induction on $|x|$:

$|x| = 0$: Then $x = \varepsilon$, and $(wx)^R = (w\,\varepsilon)^R = (w)^R = \varepsilon\,w^R = \varepsilon^R\,w^R = x^R\,w^R$. Base case

$n \geq 0\ (((|x| = n) \rightarrow ((w\,x)^R = x^R\,w^R))$ Induction Hypothesis

$\forall n \geq 0\ (((|x| = n) \rightarrow ((w\,x)^R = x^R\,w^R)) \rightarrow$
$((|x| = n + 1) \rightarrow ((w\,x)^R = x^R\,w^R)))$: Inductive Step

Consider any string x, where $|x| = n + 1$. Then $x = u\,a$ for some character a and $|u| = n$. So:

$(w\,x)^R$	$= (w\,(u\,a))^R$	rewrite x as ua
	$= ((w\,u)\,a)^R$	associativity of concatenation
	$= a\,(w\,u)^R$	definition of reversal
	$= a\,(u^R\,w^R)$	induction hypothesis
	$= (a\,u^R)\,w^R$	associativity of concatenation
	$= (ua)^R\,w^R$	definition of reversal
	$= x^R\,w^R$	rewrite ua as x

STRINGS
Relations on Strings. Substrings

`aaa`	is a ***substring*** of	`aaabbbaaa`
`aaaaaa`	is not a substring of	`aaabbbaaa`
`aaa`	is a ***proper substring*** of	`aaabbbaaa`

- Every string is a substring of itself.
- ε is a substring of every string.

STRINGS
Relations on Strings. Prefixes

s is a ***prefix*** of t iff: $\exists x \in \Sigma^* \, (t = sx)$.

s is a ***proper prefix*** of t iff: s is a prefix of t and $s \neq t$.

Examples:

The ***prefixes*** of `abba` are: ε, `a`, `ab`, `abb`, `abba`.
The ***proper prefixes*** of `abba` are: ε, `a`, `ab`, `abb`.

Every string is a prefix of itself.

ε is a prefix of every string.

STRINGS
Relations on Strings. Suffixes

s is a ***suffix*** of t iff: $\exists x \in \Sigma^* (t = xs)$.

s is a ***proper suffix*** of t iff: s is a suffix of t and $s \neq t$.

Examples:

The ***suffixes*** of `abba` are: ε, `a`, `ba`, `bba`, `abba`.
The ***proper suffixes*** of `abba` are: ε, `a`, `ba`, `bba`.

Every string is a suffix of itself.

ε is a suffix of every string.

LANGUAGES

- A **language** is a set (finite or infinite) of strings chosen from some finite alphabet Σ
  - The set of all binary strings consisting of some number of 0's followed by an equal number of 1's; that is, ε; 01; 0011; 000111; …
  - C (the set of C programs that compiles without syntax errors)
  - English

LANGUAGES

- **Another example:** Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$.
  - Some languages over Σ:
    - $\varnothing$,
    - $\{\varepsilon\}$,
    - $\{\mathtt{a}, \mathtt{b}\}$,
    - $\{\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \mathtt{aaaa}, \mathtt{aaaaa}\}$

The language Σ^* contains an infinite number of strings, including: ε, $\mathtt{a}$, $\mathtt{b}$, $\mathtt{ab}$, $\mathtt{ababaa}$.

LANGUAGES

- **Another example:** $L = \{x \in \{\mathtt{a}, \mathtt{b}\}^* :$ all `a`'s precede all `b`'s$\}$

`ab`, `aabb` and `aabbb` are in L.

`aba`, `ba`, and `abc` are not in L.

What about: ε, `a`, `aa`, and `bb`?

LANGUAGES

- **Another example:** $L = \{x : \exists y \in \{\mathtt{a}, \mathtt{b}\}^* : x = y\mathtt{a}\}$

  Simple English description:

LANGUAGES
Two important little languages

- $L = \{\} = \varnothing$
  - The language that contains no strings
- $L = \{\varepsilon\}$
  - The language that contains the empty string

LANGUAGES
English isn't a well defined language

- L = {w: w is a sentence in English}.
- Examples, which sentences are in L?

Kerry hit the ball.

Colorless green ideas sleep furiously.

The window needs fixed.

Ball the Stacy hit blue.

LANGUAGES
The Halting Problem: an important language

- $L = \{w: w$ is a C program that halts on all inputs$\}$.
  - Well specified.
    - Unlike the English language example on previous slide
  - Can we decide what strings it contains?
    - We will see this on Week 10!

LANGUAGES
Using relations to define languages

- What are the following languages?
  - $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: no prefix of w contains $\mathtt{b}\}$
  - $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: no prefix of w starts with $\mathtt{b}\}$
  - $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: every prefix of w starts with $\mathtt{a}\}$

LANGUAGES
Using relations to define languages

- What are the following languages?
  - $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: no prefix of w contains $\mathtt{b}\}$

    $= \{\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \mathtt{aaaa}, \mathtt{aaaaa}, \mathtt{aaaaaa}, \ldots\}$
  - $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: no prefix of w starts with $\mathtt{b}\}$

    $= \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: first character of w is $\mathtt{a}\} \cup \{\varepsilon\}$
  - $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*$: every prefix of w starts with $\mathtt{a}\}$

    $= \varnothing$

LANGUAGES
Languages are Sets

- If we want to provide a Computational definition of a language we specify either
  - **Generator**, which enumerates the elements
  - **Recognizer**, which decides whether a candidate string is or not in the language
    - Returns *True* or *False*

LANGUAGES
Languages are Sets

Generators (enumerators)

- Sometimes it is important the order in which the elements are generated
- If there exists an order of the elements of Σ we can use **lexicographical order**
  - Shorter strings precede longer ones
  - If two strings have the same length, sort them in dictionary order
- The lexicographic enumeration of:
  $\{w \in \{\mathtt{a}, \mathtt{b}\}^* : |w| \text{ is even}\}$:

LANGUAGES
Languages are sets

- **What is the cardinality of a language?**
  - The smallest language over any Σ is $\varnothing$, with cardinality 0.
  - The largest is Σ^*. How big is it?

LANGUAGES
Languages are sets

Theorem: If $\Sigma \neq \emptyset$ then Σ^* is countably infinite.

Proof: The elements of Σ^* can be lexicographically enumerated by the following procedure:

- Enumerate all strings of length 0, then length 1, then length 2, and so forth.
- Within the strings of a given length, enumerate them in dictionary order.

- This enumeration is infinite since there is no longest string in Σ^*.

- Since there exists an infinite enumeration of Σ^*, it is countably infinite. **[Theorem A.1: Appendix A]**

LANGUAGES
Languages are sets

- So the smallest language has cardinality 0.
- The largest is countably infinite.
- So every language is either finite or countably infinite!

LANGUAGES
Languages are sets

- ❑ **How many languages are there?**

Theorem: If $\Sigma \neq \varnothing$ then the set of languages over Σ is uncountably infinite.

Proof:
The set of languages defined on Σ is $\mathcal{P}(\Sigma^*)$.

Σ^* is countably infinite.

If S is a countably infinite set, $\mathcal{P}(S)$ is uncountably infinite.

So $\mathcal{P}(\Sigma^*)$ is uncountably infinite. **[Theorem A.4: Appendix A]**

Functions on Languages

- **Set operations**
  - Union
  - Intersection
  - Complement
  - Difference

- **Language operations**
  - Concatenation
  - Kleene star
  - Kleene plus

LANGUAGES
Functions on languages: Concatenation

If L_1 and L_2 are languages over Σ:

$$\boldsymbol{L_1L_2 = \{w \in \Sigma^* : \exists s \in L_1\ (\exists t \in L_2\ (w = st))\}}$$

Examples:

$L_1 = \{\texttt{cat}, \texttt{dog}\}$
$L_2 = \{\texttt{apple}, \texttt{pear}\}$
$L_1\ L_2 = \{\texttt{catapple}, \texttt{catpear}, \texttt{dogapple}, \texttt{dogpear}\}$

$L_1 = \texttt{a}^*$ $\qquad$ $L_2 = \texttt{b}^*$
$L_1\ L_2 =$

LANGUAGES

Functions on languages: Kleene Star

$$L^* = \{\varepsilon\} \cup \{w \in \Sigma^* : \exists k \geq 1\ (\exists w_1, w_2, \ldots w_k \in L\ (w = w_1 w_2 \ldots w_k))\}$$

Example:

L = {`dog`, `cat`, `fish`}

L^* = {ε, `dog`, `cat`, `fish`, `dogdog`, `dogcat`, `fishcatfish`, `fishdogdogfishcat`, …}

LANGUAGES
Functions on languages: Kleene Plus

The + Operator

$L^+ = L\,L^*$

$$L^* = L^0 \cup L^1 \cup L^2 \cup L^3 \cup \ldots$$
$$L^+ = \quad L^1 \cup L^2 \cup L^3 \cup \ldots$$

$L^+ = L^* - \{\varepsilon\}$ iff $\varepsilon \notin L$

L^+ is the closure of L under concatenation.

LANGUAGES
Functions on languages: Concatenation and reverse

Theorem: $(L_1\ L_2)^R = L_2^R\ L_1^R$.

Proof:

$\forall x\ (\forall y\ ((xy)^R = y^R x^R))$ (see slide 11 of this lesson)

$(L_1\ L_2)^R = \{(xy)^R : x \in L_1 \text{ and } y \in L_2\}$ (by the definition of concatenation of languages)

$= \{y^R x^R : x \in L_1 \text{ and } y \in L_2\}$

$= L_2^R\ L_1^R$ (by the definition of concatenation of languages)

DECISION PROBLEMS

A ***decision problem*** is simply a problem for which the answer is yes or no (True or False). A ***decision procedure*** answers a decision problem.

Examples:

- Given an integer *n*, does *n* have a pair of consecutive integers as factors?

The language recognition problem: Given a language *L* and a string *w*, is *w* in *L*?

Our focus

Power of Encoding

Everything is a string.

Two categories:

- Problems that are already stated as decision problems.
- Problems that don’t look like decision problems can be recast into new problems that do look like that.

Power of Encoding – Everything is a String

What If We're Not Working with Strings?

Anything can be encoded as a string.

$<X>$ is the string encoding of some object X.

$<X, Y>$ is the string encoding of the pair of objects X, Y.

DECISION PROBLEMS

Pattern matching on the web:

- **Problem:** Given a search string *w* and a web document *d*, do they match? In other words, should a search engine, on input *w*, consider returning *d*?

- The language to be decided: {<*w*, *d*> : *d* is a candidate match for the query *w*}

DECISION PROBLEMS

Does a program always halt?

- **Problem:** Given a program *p*, written in some standard programming language, is *p* guaranteed to halt on all inputs?

- The language to be decided:

$$HP_{ALL} = \{p : p \text{ halts on all inputs}\}$$

DECISION PROBLEMS

- **Problem:** Given a nonnegative integer n, is it prime?

- An instance of the problem: Is 9 prime?
- **Encoding:** To encode the problem we need a way to encode each instance: We encode each nonnegative integer as a binary string.

- The language to be decided:

$$\text{PRIMES} = \{w : w \text{ is the binary encoding of a prime number}\}.$$

DECISION PROBLEMS

- **Problem:** Verify the correctness of the addition of two numbers.
- An instance of the problem: 2 + 3 = 5 ?
- **Encoding:** encode each of the numbers as a string of decimal digits. Each instance of the problem is a string of the form:
  $\langle integer_1 \rangle + \langle integer_2 \rangle = \langle integer_3 \rangle$
- The language to be decided:

  INTEGERSUM = {w of the from: $\langle integer_1 \rangle + \langle integer_2 \rangle = \langle integer_3 \rangle$: each of the substrings $\langle integer_1 \rangle$, $\langle integer_2 \rangle$ and $\langle integer_3 \rangle$ is an element of $\{0,1,2,3,4,5,6,7,8,9\}^+$ and $integer_3$ is sum of $integer_1$ and $integer_2$}.

DECISION PROBLEMS

- **Problem:** Protein sequence alignment:

> Given a protein fragment *f* and a complete protein molecule *p*, could *f* be a fragment from *p*?

- **Encoding**: Represent each protein molecule or fragment as a sequence of amino acid residues. Assign a letter to each of the 20 possible amino acids. So a protein fragment might be represented as `AGHTYWDNR`.

- The language to be decided:

  PRTNALIGN={<*f*, *p*> : *f* could be a fragment from *p*}.

DECISION PROBLEMS

Turning Problems into Decision Problems

- Any problem can be reformulated as a decision problem
- **IDEA:** Encode both the inputs and outputs of the original problem *P* into a single string.
  - For example if *P* takes two inputs and produces one result, then string representation could be $s=i_1; i_2; r$
- Then a string $s= x; y; z$ is in the language L that corresponds to *P* iff *z* is the result that *P* produces given the inputs *x* and *y*.

DECISION PROBLEMS

Turning Problems into Decision Problems

Casting multiplication as decision:

- **Problem:** Given two nonnegative integers, compute their product.
- **Encoding:** Transform computing into verification.
- The language to be decided:

L = {w of the form:
<$integer_1$>×<$integer_2$>=<$integer_3$>, where:
<$integer_n$> is any well formed integer, and
$integer_3 = integer_1 * integer_2$}

`12×9=108` $\in L$
`12=12` $\notin L$
`12×8=108` $\notin L$

DECISION PROBLEMS

Turning Problems into Decision Problems

Casting sorting as decision:

- **Problem:** Given a list of integers, sort it.
- **Encoding**: Transform the sorting problem into one of examining a pair of lists.
- The language to be decided:

$L = \{w_1 \# w_2 : \exists n \geq 1$
$(w_1$ is of the form $<int_1, int_2, \ldots int_n>$,
w_2 is of the form $<int_1, int_2, \ldots int_n>$, and
w_2 contains the same objects as w_1 and
w_2 is sorted$)\}$

Examples:

`1,5,3,9,6#1,3,5,6,9` $\in L$

`1,5,3,9,6#1,2,3,4,5,6,7` $\notin L$

DECISION PROBLEMS

Turning Problems into Decision Problems

The Traditional Problems and their Language Formulations are Equivalent

By equivalent we mean that either problem can be ***reduced to*** the other.

That is: if we have a ***machine*** to solve one, we can use it to build a machine to do the other using just the starting machine and other functions that can be built using a machine of equal or lesser power.

Machines accept or reject strings that we feed into them

An Example

Suppose we have a program **P** that multiplies a pair of integers. Then the following program decides the language INTEGERMUL where

INTEGERMUL={*w* of the form:<$integer_1$>$\times$<$integer_2$>= <$integer_3$>, where: <$integer_n$> is any well formed integer, and $integer_3 = integer_1 * integer_2$}

INTEGERMUL(*w*):

Given a string w of the form <$integer_1$>$\times$<$integer_2$>=<$integer_3$>

1. Let x = convert-to-integer(<$integer_1$>).
2. Let y = convert-to-integer(<$integer_2$>).
3. Let z = **P**(x,y)
4. If z = convert-to-integer(<$integer_3$>) then accept Else reject.

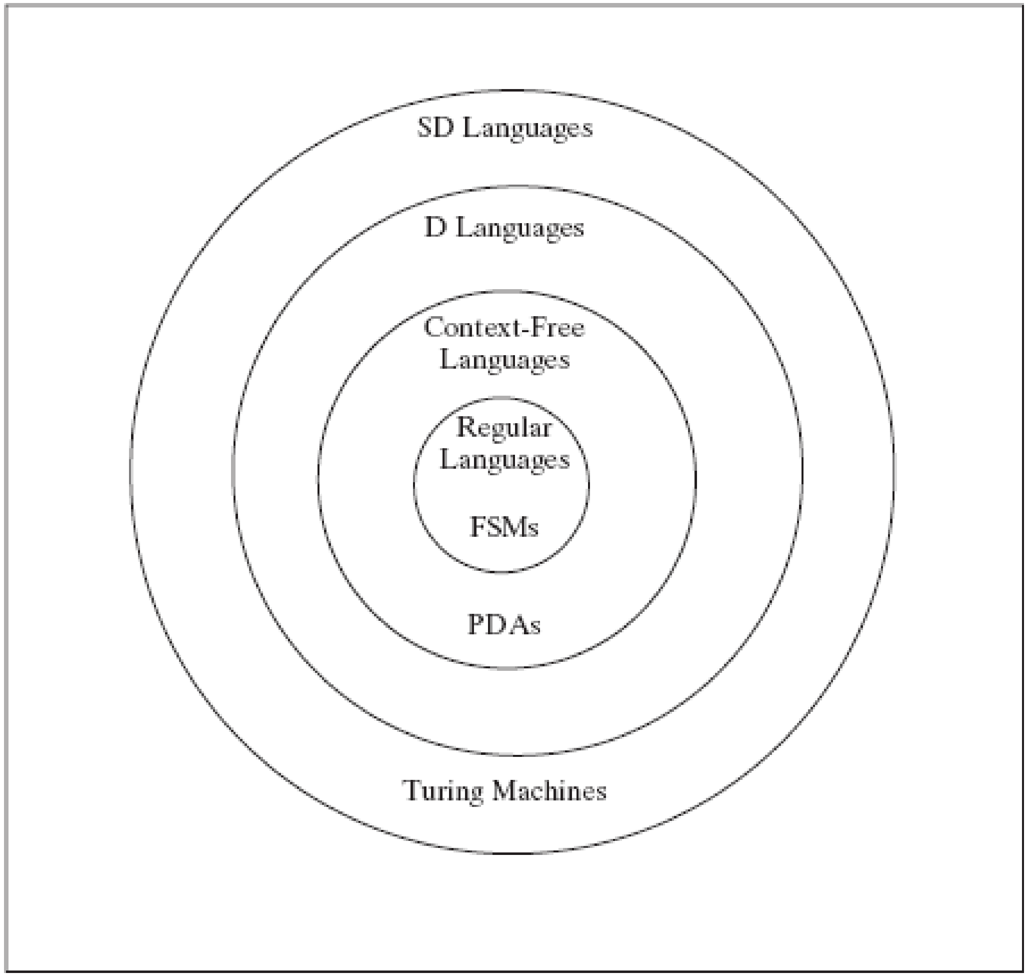
An Example

Alternatively, if we have a program T that decides the language INTEGERMUL then the following program **P** computes the multiplication of two integers x and y:

P (x,y):

1. Lexicographically enumerate the strings that represent decimal encodings of nonnegative integers.
2. Each time a string s is generated, create the new string <x> × <y> = s.
3. Feed the string to T.
4. If T accepts <x> × <y> = s, halt and return convert-to-integers(s).

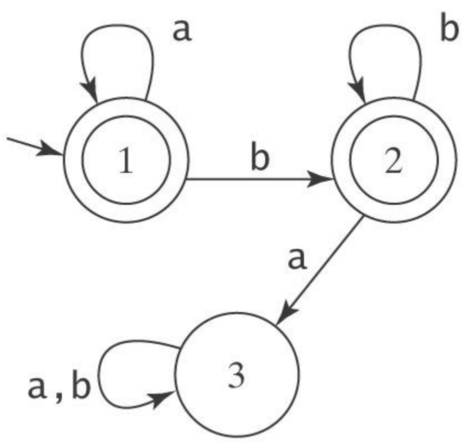
LANGUAGES AND MACHINES



LANGUAGES AND MACHINES

Finite State Machines

An FSM to accept `a*b*`:

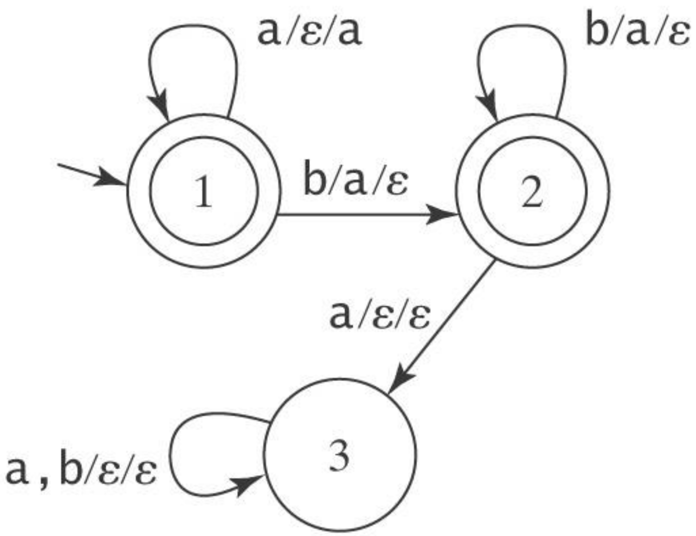


Any FSM to accept $A^nB^n = \{\mathtt{a}^n\mathtt{b}^n : n \geq 0\}$?

LANGUAGES AND MACHINES

Pushdown Automata

A PDA to accept $A^nB^n = \{a^n b^n : n \geq 0\}$

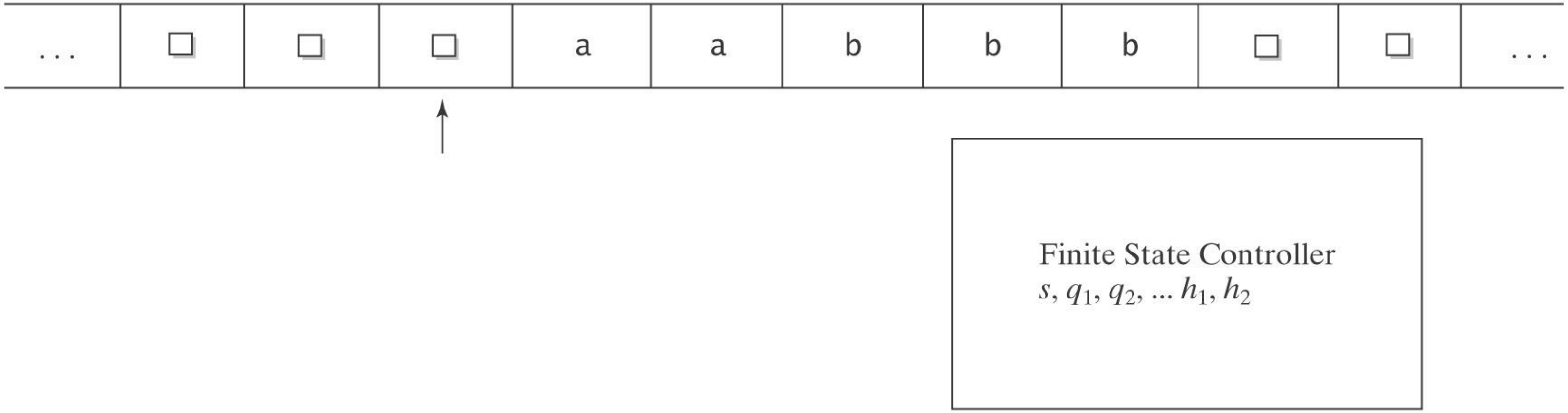


Example: aaabb

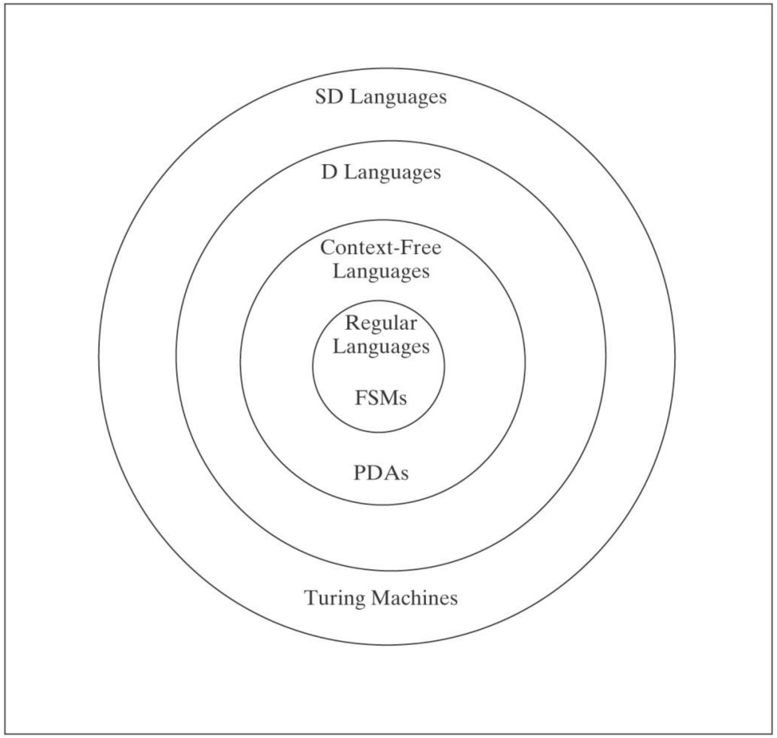
Stack:

LANGUAGES AND MACHINES
Turing Machines

A Turing Machine to accept $A^nB^nC^n$:



LANGUAGES AND MACHINES



Rule of Least Power: "Use the least powerful language suitable for expressing information, constraints or programs on the World Wide Web."

DECISION PROCEDURES

- A **decision procedure** is an algorithm to solve a decision problem
  - i.e. a program whose result is a Boolean value
  - Therefore, a decision procedure must halt.

- Examples:
  - Is string s in the language L?
  - Given two strings *s* and *t*, does *s* occur anywhere as a substring of *t*?

DECISION PROCEDURES

- A decision procedure is an algorithm that correctly answers a question and terminates. The whole idea of a decision procedure itself raises a new class of questions.
  - Is there a decision procedure for question X?
  - What is that procedure?
  - How efficient is the best such procedure?
- Clearly, if we jump immediately to an answer to question 2, we have our answer to question 1. But sometimes it makes sense to answer question 1 first. For one thing, it tells us whether to bother looking for answers to questions 2 and 3.

DECISION PROCEDURES

Fermat numbers

$F_n = 2^{2^n} + 1, n \geq 0$

$F_0 = 3$, $F_1 = 5$, $F_2 = 17$, $F_3 = 257$, $F_4 = 65{,}537$,
$F_5 = 4{,}294{,}967{,}297$, …

- Are there any prime Fermat numbers less than 1,000,000?
- Are there any prime Fermat numbers greater than 1,000,000?

DECISION PROCEDURES

- Given a Java program *P* that takes a string w as input. Does *P* halt on some particular string w?

- Given a Java program *P* that takes a single string as input parameter, does it halt on all possible input values?

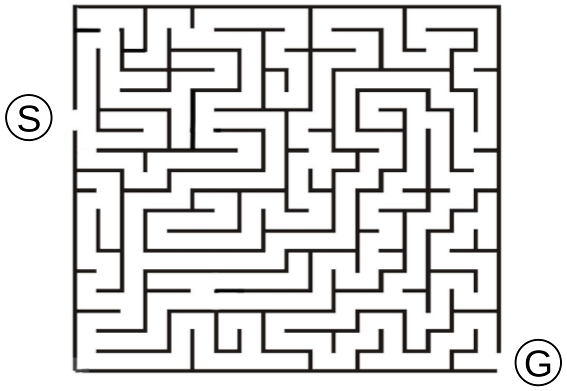
DECISION PROCEDURES

- The bottom line is that there are 3 kinds of questions:
  - Those for which a decision procedure exists
  - Those for which no decision procedure exists but a semi-decision procedure exists
  - Those for which not even a semi-decision procedure exists

DETERMINISM AND NONDETERMINISM

- Imagine you had to add to a programming language the action "***choose***" defined as:
  - ***choose*** (action 1;;
    action 2;;
    …
    action *n*)

    OR
  - ***choose*** (*x* from *S*: *P*(*x*))



- **choose** will
  - Return successful value if there is one
  - If there is no successful value, then **choose** will :
    - Halt and return False if all the actions halt and return False
    - Fail to halt if any of the actions fails to halt.

DETERMINISM AND NONDETERMINISM

- Nondeterministic trip planner

```
trip-plan(start, finish) =
   return (choose(fly-major-airline-and-rent-car(start, finish);;
                  fly-regional-airline-and-rent-car(start, finish);;
                  take-train-and-use-public-transportation
                                                     (start, finish);;
                  drive(start, finish)   ))
```

- Each of the 4 functions `trip-plan` calls returns a successful value iff it succeeds in finding a plan that meets the cost and time requirements
- Doesn't care if they run in parallel or sequentially, just needs to know if there is a value and if so what it is

The 15-Puzzle

5	2	15	9
7	8	4	12
13	1	6	11
10	14	3	

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

solve-15(*position-list*) =

/* Explore moves available from the last board configuration to have been generated. */

current = *last*(*position-list*);

if *current* = *solution* then return (*position-list*);

/* Assume that *successors*(*current*) returns the set of configurations that can be generated by one legal move from *current*. No other condition needs to be checked, so *choose* simply picks one. */

append(*position-list*, *choose x* from *successors*(*current*): *True*);

/* Recursively call *solve*-15 to continue searching from the new board configuration. */

return(*solve*-15(*position-list*));

Implementing Nondeterminism

before the first choice *choose* makes

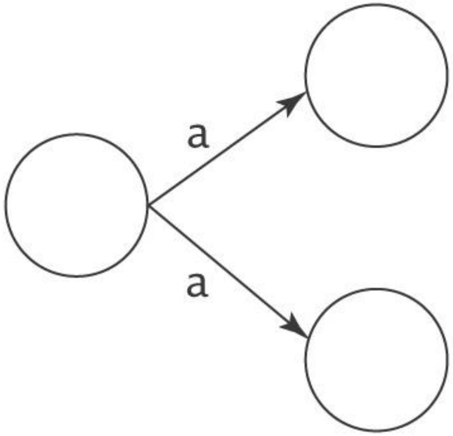
first call to *choose*
choice 1

first call to *choose*
choice 2

second call to *choose*
choice 1

second call to *choose*
choice 2

Nondeterminism in Finite State Machines



- Nondeterminism in FSM increases conveneince
- For every NDFSM there is an equivalent deterministic FSM
- So adding choose does not change the class of languages that can be accepted.
- This is also true for Turing machine but not for PDA

CLOSURE PROPERTIES ON LANGUAGES

- Now a brief reminder of our ‘framework’ slides

- A binary relation *R* on a set *A* is ***closed under*** property *P* if and only if *R* ***possesses*** *P*.

  Examples

  $<$ on the integers, *P* = transitivity

  $\leq$ on the integers, *P* = reflexive

CLOSURE PROPERTIES ON LANGUAGES

- Let INF be the set of infinite languages.
- Let FIN be the set of finite languages.

- Are languages FIN and INF closed under function...
  – *union*
  – *intersection*
  – *firstchars*
  – *chop*

CLOSURE PROPERTIES ON LANGUAGES

- Let *firstchars*(L) = $\{w : \exists y \in L\ (y = cx \wedge c \in \Sigma_L \wedge x \in \Sigma_L^* \wedge w \in c^*)\}$.
- What is *firstchars* (A^nB^n)?
- What is *firstchars* ($A^nB^nC^n$)?
- Are FIN and INF closed under *firstchars*?

CLOSURE PROPERTIES ON LANGUAGES

- Let *firstchars*(L) = $\{w : \exists y \in L\ (y = cx \wedge c \in \Sigma_L \wedge x \in \Sigma_L{}^* \wedge w \in c^*)\}$.

- What is *firstchars* (A^nB^n)?
- {a*}
- What is *firstchars* ($A^nB^nC^n$)?
- {a*}
- Are FIN and INF closed under *firstchars*?
- Think about it!

CLOSURE PROPERTIES ON LANGUAGES

- Let $chop(L) = \{w : \exists x \in L\ (x = x_1 c x_2,\ x_1 \in \Sigma_L^*,\ x_2 \in \Sigma_L^*,$
  $c \in \Sigma_L,\ |x_1| = |x_2|, \text{ and } w = x_1 x_2)\}$.

- What is $chop(A^nB^n)$?

- What is $chop(A^nB^nC^n)$?

- Are FIN and INF closed under *chop*?

CLOSURE PROPERTIES ON LANGUAGES

- Let $chop(L) = \{w : \exists x \in L\ (x = x_1 c x_2,\ x_1 \in \Sigma_L^*,\ x_2 \in \Sigma_L^*,$
  $c \in \Sigma_L,\ |x_1| = |x_2|, \text{ and } w = x_1 x_2)\}$.

- What is $chop(A^nB^n)$?
- $\varnothing$
- What is $chop(A^nB^nC^n)$?
- $\{a^{2n+1}b^{2n}c^{2n+1} : n \geq 0\}$
- Are FIN and INF closed under *chop*?
- Think about it!

CLOSURE PROPERTIES ON LANGUAGES

- Are languages FIN and INF closed under function...

Function	FIN	INF
union	Yes	Yes
intersection	Yes	No
firstchars	No	Yes
chop	Yes	No

Summary

- Alphabet, Strings, Languages
- Functions on String: Concatenation, Reversal, Replication
- Relation on String: Substring, Prefix, Suffix.
- Languages are sets: Functions on languages are functions on sets
- Operations on Languages: Concatenation, Kleene Star, Kleene Plus,
- What is Decision Problem and Decision Procedure?
- How any problem can be casted as an equivalent decision problem?
- Rule of least power
- Difference between determinism and non-determinism

References

- **Automata, Computability and Complexity. Theory and Applications**

  - By Elaine Rich

- Chapter 2, 3, 4:
  - Page : 8~52.