The Big Picture

Chapter 3
Examining Computational Problems

• We want to examine a given computational problem and see how difficult it is.
• Then we need to compare problems
• Problems appear different

• We want to cast them into the same kind of problem
  • decision problems
  • in particular, language recognition problem
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
The Power of Encoding

• For problem already stated as decision problems:
  • define the language to be decided: encode the inputs as strings and then define a language that contains exactly the set of inputs for which the desired answer is yes.

• For other problems: first reformulate the problem as a decision problem, then encode it as a language recognition task (as described above)
Even Length Testing

Problem: Given \( w \in \{a, b\}^* \), is \( w \) even-length?

- The language to be decided: \( \{w \in \{a, b\}^*: w \) is even-length\}

- The original problem and its language formulation are Equivalent
  - By equivalent we mean that either problem can be reduced to the other.
  - 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.
Primality Testing

Problem: Given a nonnegative integer $x$, is it prime?

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

Anything can be encoded as a string.

\(<X>\) is the string encoding of \(X\).
\(<X, Y>\) is the string encoding of the pair \(X, Y\).
Everything is a String

• Problem: Given an undirected graph $G$, is it connected?
• Instance of the problem:

```
1       2       3
\    \    \\
4     5
```

• Encoding of the problem: Let $V$ be a set of binary numbers, one for each vertex in $G$. Then we construct $\langle G \rangle$ as follows:
  • Write $|V|$ as a binary number,
  • Write a list of edges,
  • Separate all such binary numbers by “/”.

```
101/1/10/10/11/1/100/10/101
```

• The language to be decided: CONNECTED = {$w \in \{0, 1, /\}^* : w = n_1/n_2/\ldots n_i$, where each $n_i$ is a binary string and $w$ encodes a connected graph, as described above}.
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: $\{<w, d> : d \text{ is a candidate match for the query } w\}$
Does a Program Always Halt?

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

• The language to be decided:

\[ HP_{\text{ALL}} = \{ p : p \text{ halts on all inputs} \} \]
Turning Problems Into Decision Problems

Transform the original problem into a verification problem that verifies the correctness of candidate solutions.

- a decision problem
- equivalence in terms of solvability/computability

How can we use verification for solving?
Problem: Given two nonnegative integers, compute their product.

• Reformulation: Transform computing into verification.
  – Is 2x3=5?

• The language to be decided:

\[ L = \{ w \text{ of the form: } \langle \text{integer}_1 \rangle \times \langle \text{integer}_2 \rangle = \langle \text{integer}_3 \rangle, \text{ where: } \langle \text{integer}_n \rangle \text{ is any well formed integer, and } \langle \text{integer}_3 \rangle = \langle \text{integer}_1 \rangle \times \langle \text{integer}_2 \rangle \} \]

\[ 12 \times 9 = 108 \]
\[ 12 = 12 \]
\[ 12 \times 8 = 108 \]
Problem: Given a list of integers, sort it.

- Reformulation: Transform sorting into verification
  - Given two lists, is the 2^{nd} in sorted order of the 1^{st}?
- The language to be decided:
  \[ L = \{ w_1 \neq w_2 : \exists n \geq 1 (w_1 \text{ is of the form } <\text{int}_1, \text{int}_2, \ldots \text{int}_n>, \\
  w_2 \text{ is of the form } <\text{int}_1, \text{int}_2, \ldots \text{int}_n>, \text{ and} \\
  w_2 \text{ contains the same objects as } w_1 \text{ and } w_2 \text{ is sorted}) \} \]

Examples:
\[
\begin{align*}
1, 5, 3, 9, 6 & \# 1, 3, 5, 6, 9 \\
1, 5, 3, 9, 6 & \# 1, 2, 3, 4, 5, 6, 7
\end{align*}
\]
Problem: Given a database and a query, execute the query.

• Reformulation: Transform the query execution problem into evaluating a reply for correctness.

• The language to be decided:

$L = \{d \# q \# a: \]
  d \text{ is an encoding of a database,} \\
  q \text{ is a string representing a query, and} \\
  a \text{ is the correct result of applying } q \text{ to } d\}$

Example:

(name, age, phone), (John, 23, 567-1234) # (select name age=23) # (Mary, 24, 234-9876) # (John)
Another Example Showing Equivalence

Consider the multiplication example:

\[ L = \{ w \text{ of the form:} \]
\[ <\text{integer}_1> \times <\text{integer}_2> = <\text{integer}_3>, \text{ where:} \]
\[ \text{integer}_n \text{ is any well formed integer, and} \]
\[ \text{integer}_3 = \text{integer}_1 \times \text{integer}_2 \} \]

Given a multiplication machine, we can build the language recognition machine:

Given the language recognition machine, we can build a multiplication machine:
Languages and Machines

SD Languages
(Recursively enumerable)

D Languages
(Recursive)

Context-Free Languages

Regular Languages

FSMs

PDAs

Turing Machines
Finite State Machines

An FSM to accept $a^*b^*$:

- We call the class of languages acceptable by some FSM regular.
- There are simple useful languages that are not regular:
  - An FSM to accept $A^nB^n = \{a^n b^n : n \geq 0\}$
  - How can we compare numbers of a’s and b’s?
  - The only memory in an FSM is in the states and we must choose a fixed number of states in building it. But no bound on number of a’s
Pushdown Automata

Build a PDA (roughly, FSM + a single stack) to accept $A^nB^n = \{a^n b^n : n \geq 0\}$

Example: $aaabb$

Stack:
Another Example

• Bal, the language of balanced parentheses
  • contains strings like (()) or ()(), but not ())))
  • important, almost all programming languages allow parentheses, need checking
  • PDA can do the trick, not FSM

• We call the class of languages acceptable by some PDA context-free.

• There are useful languages not context free.
  • $A^n B^n C^n = \{a^n b^n c^n : n \geq 0\}$
  • a stack wouldn’t work. All popped out and get empty after counting b
Turing Machines

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

| ... | □ | □ | □ | a | a | b | b | b | □ | □ | ... |

Finite State Controller $s, q_1, q_2, ... h_1, h_2$
Turing Machines

• FSM and PDA (exists some equivalent PDA) are guaranteed to halt.
• But not TM. Now use TM to define new classes of languages, \( D \) and \( SD \).
• A language \( L \) is in \( D \) iff there exists a TM \( M \) that halts on all inputs, accepts all strings in \( L \), and rejects all strings not in \( L \).
  • in other words, \( M \) can always say yes or no properly
• A language \( L \) is in \( SD \) iff there exists a TM \( M \) that accepts all strings in \( L \) and fails to accept every string not in \( L \). Given a string not in \( L \), \( M \) may reject or it may loop forever (no answer).
  • in other words, \( M \) can always say yes properly, but not no.
    • give up looking? say no?
• \( D \subset SD \)

• \( Bal, A^nB^n, A^nB^nC^n \ldots \) are all in \( D \)
  • how about regular and context-free languages?
• In \( SD \) but \( D \): \( H = \{<M, w> : TM \ M \ halts \ on \ input \ string \ w\} \)
• Not even in \( SD \): \( H_{all} = \{<M> : TM \ M \ halts \ on \ all \ inputs\} \)
Languages and Machines

Hierarchy of language classes

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

- Applies far more broadly.
- Expressiveness generally comes at a price
  - computational efficiency, decidability, clarity
Languages, Machines, and Grammars

- SD (recursively enumerable)
- TMs
- Unrestricted grammar

- Context-free languages
- NDPDAs
- Context-sensitive grammar

- Context-sensitive languages
- LBAs

- D (recursive)

- Regular languages
- FSMs
- Regular grammar / regular expression

- DCF
- DPDAs

- Context-sensitive languages
- LBAs
- Context-sensitive grammar

- Context-free languages
- NDPDAs

- Unrestricted grammar
A Tractability Hierarchy

- **P**: contains languages that can be decided by a TM in polynomial time

- **NP**: contains languages that can be decided by a nondeterministic TM (one can conduct a search by guessing which move to make) in polynomial time

- **PSPACE**: contains languages that can be decided by a machine with polynomial space

\[ P \subseteq NP \subseteq PSPACE \]

- **P = NP**? Biggest open question for theorists
Decision Procedures

Chapter 4
Decidability Issues

Goal of the book: be able to make useful claims about problems and the programs that solve them.

• cast problems as language recognition tasks
• define programs as state machines whose input is a string and output is *Accept* or *Reject*
Decision Procedures

An *algorithm* is a detailed procedure that accomplishes some clearly specified task.

A *decision procedure* is an algorithm to solve a decision problem.

Decision procedures are programs and must possess two correctness properties:
- must halt on all inputs
- when it halts and returns an answer, it must be the correct answer for the given input
Decidability

• A decision problem is **decidable** iff there exists a decision procedure for it.

• A decision problem is **undecidable** iff there exists no a decision procedure for it.

• A decision problem is **semidecidable** iff there exists a semidecision procedure for it.
  • a semidecision procedure is one that halts and returns *True* whenever *True* is the correct answer. When *False* is the answer, it may either halt and return *False* or it may loop (no answer).

• Three kinds of problems:
  • decidable (recursive)
  • not decidable but semidecidable (recursively enumerable)
  • not decidable and not even semidecidable
Checking for even numbers: Is the integer $x$ even?

Let / perform truncating integer division, then consider the following program:

$$\text{even}(x: \text{integer}) =$$

$$\text{If}(x/2) \times 2 = x \text{ then return } True \text{ else return } False$$

Is the program a decision procedure?
Undecidable but Semidecidable

Halting Problem: For any Turing machine $M$ and input $w$, decide whether $M$ halts on $w$.
- $w$ is finite
- $H = \{<M, w> : \text{TM } M \text{ halts on input string } w\}$
- asks whether $M$ enters an infinite loop for a particular input $w$

Java version: Given an arbitrary Java program $p$ that takes a string $w$ as an input parameter. Does $p$ halt on some particular value of $w$?

```
haltsOnw(p:program, w:string) =
    1. simulate the execution of $p$ on $w$.
    2. if the simulation halts return $True$ else return $False$.
```

Is the program a decision procedure?
Halting-on-all (totality) Problem: For any Turing machine $M$, decide whether $M$ halts on all inputs.

- $H_{\text{ALL}} = \{<M> : \text{TM } M \text{ halts on all inputs}\}$
- If it does, it computes a total function
- equivalent to the problem of whether a program can ever enter an infinite loop, for any input
- differs from the halting problem, which asks whether $M$ enters an infinite loop for a particular input

Java version: Given an arbitrary Java program $p$ that takes a single string as input parameter. Does $p$ halt on all possible input values?

$$\text{haltsOnAll}(p:\text{program}) =$$

1. for $i = 1$ to infinity do:
   - simulate the execution of $p$ on all possible input strings of length $i$.
2. if all the simulations halt return $\text{True}$ else return $\text{False}$.

Is the program a decision procedure? A semidecision procedure?
Grammars, Languages, and Machines

- Grammar generates Language.
- Machine recognizes or accepts Language.
Clarification

A machine $M$ recognizes a language $L$ iff $M$ accepts all and only those strings in $L$.

A machine $M$ decides a language $L$ iff $M$ accepts all strings in $L$ and rejects all strings not in $L$.

recognize = accept = semi-decide ≠ decide

When a machine halts, it must either accepts or rejects. So for machines that always halt, accept implies decide.

A language $L$ is called semi-decidable iff some TM accepts $L$.
A language $L$ is called decidable iff some TM decides $L$.

SD: set of semi-decidable languages
D: set of decidable languages (a subset of SD by definition. actually a proper subset of SD by proof)