Computability Theory

Turing Machines
TMs

- We introduce the most powerful of the automata we will study: Turing Machines (TMs)

- TMs can compute any function normally considered computable

- Indeed we can define computable to mean computable by a TM
A Turing machine (TM) can compute everything a usual computer program can compute
- may be less efficient

But the computation allows a mathematical analysis of questions like:
- What is computable?
- What is decidable?
- What is complexity?
A Turing Machine is a theoretical computer consisting of a tape of infinite length and a read-write head which can move left and right across the tape.
Informal definition

- TMs uses an infinite memory (tape)
- Tape initially contains the input string followed by blanks
- When started, a TM executes a series of discrete transitions, as determined by its transition function and by the initial characters on the tape
Informal definition

- For each transition, the machine checks
  - what state it is in and what character is written on the tape below the head.
  - Based on those, it then changes to a new state, writes a new character on the tape, and moves the head one space left or right.

- The machine stops after transferring to the special **HALT** (accept or reject) state.

- If TM doesn’t enter a HALT state, it will go on forever
Differences between FA and TMs

- A TM can both write on the tape and read from it
- The read-write head can move both to the left and to the right
- The tape is infinite
- The special states of rejecting and accepting take immediate effect
  - No need to wait for the end of the string
Examples

a → a, R

b → a, R

a → a, R means TM reads the symbol a, it replaces it with a and moves head to right.

When reading an a, this TM will move right one square and stay in the same start state. When it scans a b, it will change this symbol to an a and go into the other state (accept state).
Examples

- A TM that tests for memberships in the language
  - $A = \{w#w \mid w \text{ belongs to } \{0,1\}^*\}$
- Idea:
  - Zig-zag across tape, crossing off matching symbols
Examples

- Tape head starts over leftmost symbol
- Record symbol in control and overwrite $X$
- Scan right: reject if blank encountered before #
- When # encountered, move right one space
- If symbols don’t match, reject
Examples

- Overwrite X
- Scan left, past # to X
- Move one space right
- Record symbol and overwrite X
- After # encountered, skip all Xs right of #
- If symbols don’t match, reject
- …
Examples

- Finally scan left past #
- If a or b encountered, reject
- When blank encountered, accept
Examples

- http://ironphoenix.org/tril/tm/
Formal Definition

- A Turing Machine is a 7-tuple \((Q, \Sigma, T, \xi, q_0, q_{\text{accept}}, q_{\text{reject}})\), where
  - \(Q\) is a finite set of states
  - \(\Sigma\) is a finite set of symbols called the alphabet
  - \(T\) is tape alphabet, where \(\_\) belongs to \(T\), and \(\Sigma \subseteq T\)
  - \(\xi : Q \times T \rightarrow Q \times T \times \{L,R\}\) is the transition function
  - \(q_0 \in Q\) is the start state
  - \(q_{\text{accept}} \subseteq Q\) is the accept state
  - \(q_{\text{reject}} \subseteq Q\) is the reject state, where \(q_{\text{accept}} \neq q_{\text{reject}}\)
The transition function

\[ \xi : Q \times T \rightarrow Q \times T \times \{L,R\} \] is the transition function
\[ \xi(q,a) = (r,b,L) \]
means

in state \( q \) where head reads tape symbol \( a \)

the machine overwrites \( a \) with \( b \)

enters state \( r \)

move the head left
Workings of a TM

- $M = (Q, \Sigma, T, \xi, q_0, q_{\text{accept}}, q_{\text{reject}})$ computes as follows

  - Input $w=w_1w_2\ldots w_n$ is on leftmost $n$ tape squares
  - Rest of tape is blank –
  - Head is on leftmost square of tape
Workings of a TM

- \( M = (Q, \Sigma, T, \xi, q_0, q_{\text{accept}}, q_{\text{reject}}) \)
- When computation starts
  - M Proceeds according to transition function \( \xi \)
  - If M tries to move head beyond left-hand-end of tape, it doesn’t move
  - Computation continues until \( q_{\text{accept}} \) or \( q_{\text{reject}} \) is reached
- Otherwise M runs forever
Configurations

- Computation changes
  - Current state
  - Current head position
  - Tape contents
Configurations

- Configuration
  - 1011r0111

- Means
  - State is r
  - Left-Hand-Side (LHS) is 1011
  - Right-Hand-Side (RHS) is 0111
  - Head is on RHS 0
Configurations

- $uarbv$ yields $upacv$ if
  - $\xi(r,b) = (p,c,L)$

- $uarbv$ yields $uacpv$ if
  - $\xi(r,b) = (p,c,R)$

- Special cases: $rbv$ yields $pcv$ if
  - $\xi(r,b) = (p,c,L)$

- $wr$ is the same as $wr--$
More configurations

- We have
  - starting configuration $q_0w$
  - accepting configuration $w_0q_{\text{accept}}w_1$
  - rejecting configuration $w_0q_{\text{reject}}w_1$
- halting configurations
  - $w_0q_{\text{accept}}w_1$
  - $w_0q_{\text{reject}}w_1$
Accepting a language

- TM M accepts input w if a sequence of configurations $C_1, C_2, \ldots, C_k$ exist
  - $C_1$ is start configuration of M on w
  - Each $C_i$ yields $C_{i+1}$
  - $C_k$ is an accepting configuration
- The collection of strings that M accepts is the language of M, denoted by $L(M)$
Detailed example

- M1 accepting \( \{ w\#w \mid w \in \{0,1\}^* \} \)
- Page 133 of Sipser’s book.
Enumerable languages

- Definition: A language is \textit{(recursively)} enumerable if some TM accepts it
  - In some other textbooks Turing-recognizable
Enumerable languages

On an input to a TM we may
- accept
- reject
- loop (run for ever)

Not very practical: never know if TM will halt
Enumerable languages

- Definition: A TM *decides* a language if it always halts in an accept or reject state. Such a TM is called a *decider*.

- Definition: A language is *decidable* if some TM decides it.
  - Some textbooks use *recursive* instead of *decidable*.

- Therefore, every decidable language is enumerable, but not the reverse!
Example of decidable language

Here is a decidable language

- $L=\{a^ib^jc^k \mid ix j = k, \ l,j,k > 0\}$

- Because there exist a TM that decides it

- How?
Example of decidable language

How?
1. Scan from left to right to check that input is of the form $a^*b^*c^*$
2. Return to the start
3. Cross off an $a$ and
4. Scan right till you see an $a$
5. Mark each $b$ and scan right till you see an $a$
6. Cross off that $c$
7. Move left to the next $b$, and repeat previous two steps until all $b$'s are marked
8. Unmark all $b$'s and go to the start of the tape
9. Goto step 1 until all $a$'s are crossed off
10. Check if all $c$'s are crossed off; if so accept; otherwise reject
Example of decidable language

- given,
  - aabbbcccccc
Example of decidable language

given,
- aabbccccccc
- a\abbbccccccc
- abbbccccccc
Example of decidable language

- given,
  - aabbbcccccc
  - $\emptyset$abbbcccccc
  - abbbcccccc
  - abbb$\in$cccccc
Example of decidable language

- given,
  - aabbbbcccccc
  - aabbbbcccccc
  - abbbcccccc
  - abbbcccccc
  - abbbcccccc
Example of decidable language

- given,
  - aabbbcccccc
  - \(\emptyset\)abbbcccccc
  - abbbcccccc
  - abbb\text{\(\emptyset\)}cccccc
  - abbb\text{\(\emptyset\)}cccccc
  - abbb\text{\(\emptyset\)}cccccc
  - abbb\text{\(\emptyset\)}cccccc
  - aabbbccc
  - aabbbccc
Example of decidable language

given,
- aabbbccccccc
- aabbbccccccc
- abbbccccccc
- abbbccccccc
- abbbccccccc
- abbbccccccc
- abbbccccccc
- abbbccccccc
- bbbcccc bbbcc bbbc bbb
TM: Variants

- Turing Machine head stays put
  - Does not add any power
- Nondeterminism added to TMs
  - Does not add any power
Many models have been proposed for general-purpose computation.

Remarkably all “reasonable” models are equivalent to Turing Machines.

All “reasonable” programming languages like C, C++, Java, Prolog, etc are equivalent.

The notion of an *algorithm* is model-independent!
What is an algorithm?

- Informally
  - A recipe
  - A procedure
  - A computer program

- Historically
  - Notion has long history in Mathematics (Al-khwarizmi), but
  - Not precisely defined until 20th century
  - Informal notion rarely questioned, but are insufficient
Hilbert’s 10th problem

- Hilbert’s tenth problem (1900): Find a finite algorithm that decides whether a polynomial has an integer root.
What is a polynomial?

- A **polynomial** is a sum of terms, each term is a product of variables and constants (coefficients).
- A **root** of a polynomial is an assignment of values to the variables such that the value of the polynomial is 0.
- $x=2$ and $y=2$ is a root for $5x + 15y = 25$.
- There is no such algorithm (1970)!
  - Mathematics of 1900 could not have proved this, because they didn’t have a formal notion of an algorithm.
  - Formal notions are required to show that no algorithm exists.
Church-Turing Thesis

- Formal notions appeared in 1936
  - Lambda-calculus of Alonzo Church
  - Turing machines of Alan Turing
  - These definitions look very different, but are equivalent

- The Church-Turing Thesis:
  - The intuitive notion of algorithms equals Turing machines algorithms

- In 1970, it was shown that no algorithm exists for testing whether a polynomial has integral roots
Relationships among machines

- NFA/DFA
- PDA
- NPDA
- TM
Relationships among languages

- **Regular**
- **Context-free**
- **Decidable**
- **Enumerable**
Conclusions

Turing Machines

definition

examples

Enumerable/decidable languages
definition

Next: Decidability Theory