Turing Machine 04

Thumrongsak Kosiyatrakul
tkosiyat@cs.pitt.edu

Recognizable and Decidable

- Given a language R, if some Turing machines accept every strings $s \in R$ and **does not accept** (either reject or loop indefinitely) every string $s \notin R$, we say that **"R is recognizable"**
  - Note that these machines must accept on all input $s \in R$
  - However, if $s \notin R$, these machines either reject or loop infinitely
- Given a language D, if some Turing machine accept every strings $s \in D$ and rejects every string $s \notin D$, we say that **"D is decidable"**
  - Note that these Turing machines must be deciders
  - These machine either accept or reject on all input strings
    - These machine will not loop indefinitely on any strings
  - If D is decidable, D is also recognizable

Decidable Language

- Following languages are examples of decidable languages:
  - $A = \{0^{2^n} \mid n \geq 0\}$
  - $B = \{w\#w \mid w \in \Sigma^*\}$

  We already demonstrated that there exists Turing machines (deciders) that decide above languages
- There are some languages that are recognizable but not decidable
  - Suppose R is recognizable but not decidable
    - There are TMs that **accept** all strings in R and **does not accept** all strings not in R
    - No TM can **accept** all strings in R and **reject** all strings not in R

Undecidable Language

- Consider a polynomial:

$$6x^3yz^2 + 3xy^2 - x^3 - 10$$

  - A **root** of a polynomial is an assignment to its variables which results in that value of polynomial is 0
  - A polynomial has an integral root if all variables are assigned integer values
  - The above polynomial has an integral root $x = 5$, $y = 3$, and $z = 0$
- Given a polynomial **with an integral root**, can you find out its root?
  - Yes, brute force
- Given a polynomial, can you find out whether it has an integral root?
  - Not always
  - Hilbert's tenth problem stated that there is no algorithm that tests whether a polynomial has an integral root.

Undecidable Language

- Let $\langle x \rangle$ be a string representation of the object x
- Let D be the set of all string representations of polynomials that have integral root
- Formally

$$D = \{\langle p \rangle \mid p \text{ is a polynomial with an integral root}\}$$

- Given $\langle p \rangle$ (a string representation of a polynomial p), if a Turing machine can **decide** whether
  - $\langle p \rangle \in D$ (polynomial p has an integral root) or
  - $\langle p \rangle \notin D$ (polynomial p does not have an integral root)

  D is decidable
- Hilbert's tenth problem simply stated that D is not decidable.

Undecidable Language

- Consider polynomials with one variable (e.g., $2x^2 + x - 7$)
- Let

  $D_1 = \{\langle p \rangle \mid \langle p \rangle$ is a polynomial over x with an integral root$\}$
- Is D_1 recognizable?
  - Yes, if there exists a Turing machine that accepts every $\langle p \rangle \in D_1$ and does not accept every $\langle p \rangle \notin D_1$
- Example: M_1 that recognizes D_1 using a brute force algorithm in high-level definition

  $M_1 =$"On input $\langle p \rangle$ where p is a polynomial over the variable x:

  1. Evaluate p with x set successively to the value 0, 1, -1, 2, -2, 3, -3, ... If at any point the polynomial evaluates to 0, *accept*"
- Note that M_1 accepts all $\langle p \rangle \in D_1$ and loop indefinitely on all $\langle p \rangle \notin D_1$
- Therefore, D_1 is recognizable.

Undecidable Language

- Consider polynomials with one variable (e.g., $2x^2 + x - 7$)
- Let

  $D_1 = \{\langle p \rangle \mid \langle p \rangle$ is a polynomial over x with an integral root$\}$
- Is D_1 decidable?
  - Yes, if there exists a Turing machine that accepts every $\langle p \rangle \in D_1$ and rejects every $\langle p \rangle \notin D_1$
- Luckily there is an upper/lower bound of the value of x that a machine needs to test:

  $$\pm k \frac{c_{\mathsf{max}}}{c_1}$$

  where k is the number of terms in the polynomial, c_{max} is the coefficient with the largest absolute value, and c_1 is the coefficient of the highest order term
- Change M_1 such that it rejects after testing value goes out-of-bound
- Therefore, D_1 is decidable

Undecidable Language

- Let D be the set of all polynomials that have integral root

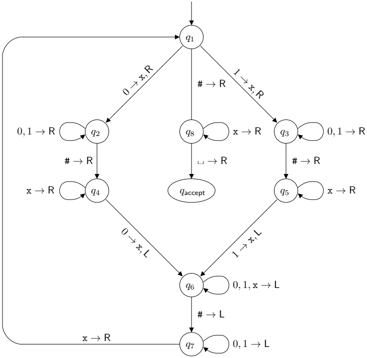
$$D = \{\langle p \rangle \mid \langle p \rangle \text{ is a polynomial with an integral root}\}$$

- We can create a machine that tries all possible assignment values starting from 0s
  - For example, in case of two variables x and y, try the following values $[x, y]$:

$$[0, 0], [0, 1], [1, 0], [1, 1], [0, -1], [-1, 0], [-1, -1], [0, 2], \dots$$

  - If a polynomial p has an integral root, eventually it will be evaluated to 0
- Therefore, D is recognizable
- Unfortunately, there is no bound that we can check and machine may loop infinitely
  - If the polynomial p does not have an integral root, we will keep trying new values of $[x, y]$ forever (loop indefinitely)
- D is not decidable

- A description of a Turing machine can be huge even for a very simple algorithm
- Example, compare two strings $\{w\#w \mid w \in \{0,1\}^*\}$



- The above state diagram represents the **formal description** in a form of state diagram of a Turing machine

Describing Turing Machines

- An **implementation description** of the previous Turing machine that decides $\{w\#w \mid w \in \{0,1\}^*\}$ is shown below
- On input string w:
  1. Zig-zag across the tape to corresponding positions on either side of the # symbol to check whether those positions contain the same symbol. If they do not, or if no # is found, *reject*. Cross off symbols as they are checked to keep track of which symbols correspond.
  2. When all symbols to the left of the # have been crossed off, check for any remaining symbols to the right of the #. If any symbols remain, *reject*; otherwise, *accept*.
- Note that the above description describes the way the Turing machine moves its head and store data (cross off symbols)

Describing Turing Machines

- An **high-level description** of the previous Turing machine that decides $\{w\#w \mid w \in \{0,1\}^*\}$ is shown below:

  $M =$ "On input s where $s = x\#y$ for some string x and y:

  1. Compare whether the string x is identical to the string y.
  2. If they are identical, $accept$; otherwise, $reject$."

- Note that the **where** clause behaves like a filter
  - Any string that does not satisfy the **where** clause will be rejected immediately
- What a TM can do?
  - From the Church-Turing thesis, if there is an algorithm to do something, a TM can do the same thing
  - Examples:
    - Compare two strings
    - Check whether the length of a string is a power of 2
    - Addition, subtraction, multiplication, division, modulo
    - Any algorithms discussed in Chapter 1

Describing Turing Machines

- High-level description of a Turing machine is suitable for describing universal Turing machine
- Consider the following language:

$$A = \{x_1\#x_2\#\ldots\#x_n \mid x_i = x_j \text{ for every } i \text{ and } j\}$$

- The following machine M' decides A using TM M as a subroutine:

$M' =$ "On input s where $s = x_1\#x_2\#\ldots\#x_n$:

1. For every i where $1 \leq i \leq n-1$:
2. Run M on input $x_i\#x_{i+1}$.
3. If M rejects, *reject*.
4. *accept*"

Conclusions

- Algorithm and Turing Machine are consider equivalent
  - Anything that an algorithm can do, there exists a TM that can do the same thing
    - Simply convert the algorithm to TM
  - Anything that a Turing machine can do, there exists an algorithm that can do the same thing
    - Simply convert the TM to algorithm
- Because of this, if there is a problem that a TM cannot solve, no algorithm can solve the same thing