Formal Languages and Automata, 2024 Fall semester

Lec 14. Variants of TM

Eunjung Kim

A BIT OF HISTORY

- In 1923, Hilbert proposed 10 open problems in the International Congress of Mathematicians ($\rightsquigarrow$ part of "Hilbert's 23 problems")
- 10th problem:

  "Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers."

A bit of history

- In order to prove that such a process ('algorithm') is impossible, we need to formalize the notion of algorithm, or computability.
- Alonzo Church, Alan Turing, Gödele-Herbrand independently came up with their own notions of computability (λ-calculus, TM, general recursive), all of which were shown to be equivalent.

A bit of history

- In order to prove that such a process ('algorithm') is impossible, we need to formalize the notion of algorithm, or computability.
- Alonzo Church, Alan Turing, Gödele-Herbrand independently came up with their own notions of computability (λ-calculus, TM, general recursive), all of which were shown to be equivalent.
- This led to Church-Turing thesis: a thesis stating that an intuitive notion of algorithms is equivalent to TM.

A bit of history

- In order to prove that such a process ('algorithm') is impossible, we need to formalize the notion of algorithm, or computability.
- Alonzo Church, Alan Turing, Gödele-Herbrand independently came up with their own notions of computability (λ-calculus, TM, general recursive), all of which were shown to be equivalent.
- This led to Church-Turing thesis: a thesis stating that an intuitive notion of algorithms is equivalent to TM.
- According to this, the notion of TM we learnt, rather anecdotal, must be robust.

A BIT OF HISTORY

- In order to prove that such a process ('algorithm') is impossible, we need to formalize the notion of algorithm, or computability.
- Alonzo Church, Alan Turing, Gödele-Herbrand independently came up with their own notions of computability (λ-calculus, TM, general recursive), all of which were shown to be equivalent.
- This led to Church-Turing thesis: a thesis stating that an intuitive notion of algorithms is equivalent to TM.
- According to this, the notion of TM we learnt, rather anecdotal, must be robust.
- All <u>reasonable</u> variations are equivalent. We'll see some examples in the class.

TM WITH 'STAY PUT' OPTION

- Now, TM has an additional option of not moving its head (stay put).
- That is, δ a function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R, S\}$.
- The new TM variant has the same (not more) power as the original TM.

TM with 'stay put' option

TM with 'stay put' option

For every TM with stay put option, there is an equivalent TM without this option (i.e. recognizing the same language).

Multitape Turing machine

- Now, TM has multiple tapes with a head on each tape, and read/write/move its heads simultaneously.
- That is, δ is a function from $Q \times \Gamma^k$ to $Q \times \Gamma^k \times \{L, R, S\}^k$.
- The multitape TM variant has the same (not more) power as the original TM.

Multitape Turing machine

Multitape TM

For every multitape TM, there is an equivalent single-tape TM.

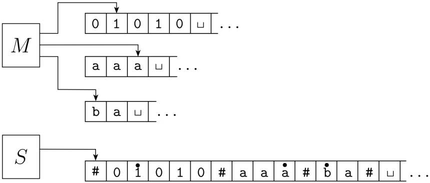

Figure 3.14, Sipser 2012.

Multitape Turing machine

Simulating Multitape TM with single-tape TM

- M has two tapes (generalizes to k tapes straightforwardly).
- S is the new single-tape TM we want to construct.
- Introduce extra symbols; $\mathring{a}$ per symbol $a \in \Sigma$ and a delimiter $\#$.
- S shall maintain the following property $(\star)$ while simulating M.
  1. The tape contents of S is of the form $\#w\#z\#$ where w and z are the strings of 1st and 2nd tape of M.
  2. The symbols of $\#w\#z\#$ corresponding to the head locations in the 1st and 2nd tapes of M are dotted, and no other symbols are dotted.

Multitape Turing machine

Simulating one transition of M with S

Consider the transition $\delta(q, a, b) = (a', b', L, R)$ of M. S simulates this transition as follows.

1. Move the head of S to the left end.
2. By scanning the tape left-to-right, decide which symbols are dotted (a and b in this case) and enters the state (q, a, b). Move the head to the left end.
3. By scanning the tape left-to-right, in each 'track' of the single tape, rewrite $\mathring{a}$ as a, and add dot on its left (or right, if the corresponding head move is 'R') symbol.
4. If the simulation at i-th track makes a move to the right, which is $\#$, we add dot to $\#$, then replace add $\sqcup$ in front of $\mathring{\#}$, restore $\mathring{\#}$ to $\#$, and shift all symbols starting after $\mathring{\#}$ by one to the right.
5. When the 3-4th steps are done for each track, simulating one transition of M is complete. Clearly the invariant $(\star)$ is maintained.

Nondetermistic Turing machine

- Now, TM can make multiple transition per state $\times$ symbol, or no transition may be defined.
- That is, δ is a function from $Q \times \Gamma$ to $2^{Q \times \Gamma \times \{L,R,S\}}$.
- Nondeterministic TM accepts an input string $w \in \Sigma_0^*$ if there exists an accepting computation history starting from $q_0 w$ (amongst *all* possible computation histories starting from $q_0 w$).
- The new TM variant has the same (not more) power as the original TM.

Nondetermistic Turing machine

Nondeterministic TM

For every nondeterministic TM, there is an equivalent multitape TM.

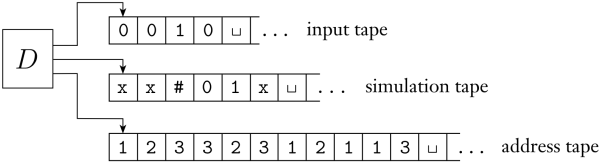

Figure 3.17, Sipser 2012.

NONDETERMISTIC TURING MACHINE

The idea.

- D keeps track of the branching computation history of M in a BFS manner.
- Address tape remembers the location of the node t in the computation tree as a p-ary tree.
- Simulation tape is used for a single-tape TM simulation of N from the root (the starting configuration) to node t.

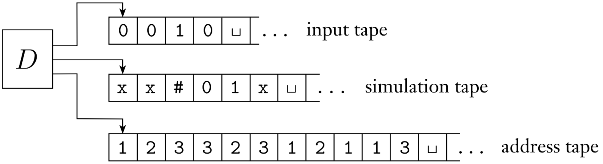

Figure 3.17, Sipser 2012.

Nondetermistic Turing machine

Simulating nondeterministic TM with 3-tape TM

- D is the 3-tape TM we want to construct.
- Introduce extra symbols; $\{0, 1, \ldots, p\}$ where $p = |2^{Q \times \Sigma \times \{L,R,S\}}| - 1$.
- A computation history of length ℓ can be represented as a sequence $s = s_1, \cdots, s_\ell$ with $s_i \in \{1, \ldots, p\}$.
- Each s_i interprets as an instruction: "choose s_i-th element out of $|2^{Q \times \Sigma \times \{L,R,S\}}|$ as the i-th move".

Nondetermistic Turing machine

D shall simulate the nondeterministic TM N as follows.

- For each sequence $s = (s_1, \ldots, s_\ell)$, simulate the (unique) computation history obtained by applying the transition sequence s in order.
  1. Initialization: D initialize the simulation tape by erasing its contents and writing the initial input string to M by copying the contents in the input tape onto the simulation tape.
  2. Each single move of M in the branch of s: D reads s_i in the address tape, update the contents in the simulation tape accordingly.
- If the simulation following the instructions of s ends in an accept state of N, then D accepts.
- Otherwise, increase s by one (in p-ary representation) and repeat the above.

Hardwired TM to programmable TM!

- TM is defined by its transition function.
- This means that one TM can compute (recognize or decide) a single function (language).
- One TM, useful for a single purpose only.
  $\rightsquigarrow$ hardwired as produced in the factory.
- But computer as we know is an all-round player with programs.
  $\rightsquigarrow$ stored-program computer, universal.
- Universal TM, the mathematical model that embodies this historic transition.

Hardwired TM to programmable TM!

Key insight

TM not only 'receives' an input string, but TM itself can be an input string (once <u>appropriately</u> encoded as a string).

Hardwired TM to programmable TM!

Key insight

TM not only ‘receives’ an input string, but TM itself can be an input string (once appropriately encoded as a string).

- Let’s build a super TM U which reads an arbitrary TM M and an input w to M, and does what M would do on the input w.

Hardwired TM to programmable TM!

Key insight

TM not only 'receives' an input string, but TM itself can be an input string (once appropriately encoded as a string).

- Let's build a super TM U which reads an arbitrary TM M and an input w to M, and does what M would do on the input w.
- If U can simulate any other TM, with U we can do any computation that any TM M can do by loading (reading) M and an input to M; instead of using all sorts of TM's, we use a single TM U - a universal TM.

Hardwired TM to programmable TM!

Key insight

TM not only 'receives' an input string, but TM itself can be an input string (once appropriately encoded as a string).

- Let's build a super TM U which reads an arbitrary TM M and an input w to M, and does what M would do on the input w.
- If U can simulate any other TM, with U we can do any computation that any TM M can do by loading (reading) M and an input to M; instead of using all sorts of TM's, we use a single TM U - a universal TM.
- Turing proved that a universal TM exists. A couple of legendary scientists and mathematicians including Turing himself realized this concept in the 40's, the earliest versions of modern-day computers.