

David Keil
Framingham State University

4. Turing machines and undecidability

1. Turing machines
2. TM computations
3. Turing decidability
4. Undecidability

Reading:
H-M-U, Ch. 8-9

Inquiry

- Can a state-transition system with infinite tape model more computations than with stack?
- What languages are decidable?
- What problems can be solved?

Objectives

- 4a. Describe the Turing-machine model of computation
- 4b. Show a problem to be decidable
- 4c. Show a problem to be undecidable
- 4d. Explain the notion of reducibility of problems

1. Turing machines

Turing's model of computation (1936):

- enabled invention of general-purpose computers
- augments DFA with a two-way read/write tape
- is roughly equivalent to the notions of an algorithm, or of a program
- enables us to reason about intractable and undecidable problems, saving us the time we might spend trying to solve them
- helps us capture essence and limits of algorithmic computing

Not all decidable sets are CF

- What class of device accepts
 $L = \{xx \mid x \in \Sigma^*\}$?
(strings that consist of the same substring repeated twice)
- Acceptor for L would require a *queue*; a one-stack machine is not sufficient
- L is not context free, is not recognized by any PDA (provable by use of Pumping Lemma for CFLs)
- Clearly L is decidable by some algorithm
- Third class of automata: *Turing machines*

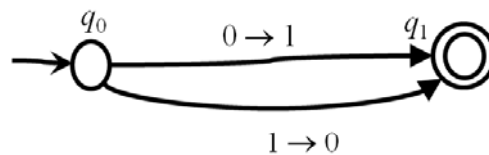
Turing's model

- Based on human “computer” with paper and pencil
- *Operations*: tape head reads a symbol at current location on paper, moves left or right, writes symbol at current location
- Next action is looked up in *transition table* based on current input and “state of mind” of computer
- Machine halts when it enters a “halting” (accept or reject) state

Definition

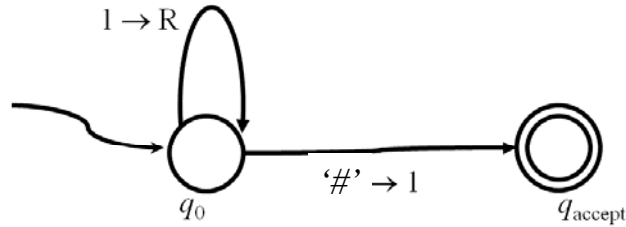
- *Turing machine:*
 $M = \langle Q, \Sigma, \Gamma, \delta, q_0, q_{\text{accept}}, q_{\text{reject}} \rangle$
 where Σ and Γ are *input* and *tape* alphabets
- *Transition function* δ :
 $Q \times (\Sigma \cup \{\#\} \cup \Gamma) \rightarrow$
 $Q \times (\Sigma \cup \Gamma \cup \{\#, L, R\})$
 where L, R denote left or right moves
- Tape is infinite in both directions
- Tape head starts at leftmost nonblank cell
- First blank to right of a symbol in Σ has infinitely many blanks to its right

Example: negater TM



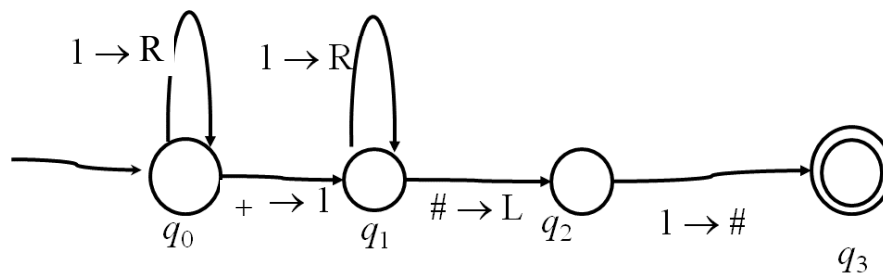
- The TM above reads one bit, writes its negation to the tape, and halts
- $\Sigma = \{ 1, 1 \}$
- $Q = \{ q_0, q_1 \}$
- $\delta = \{ \langle (q_0, 0), (q_1, 1) \rangle, \langle (q_0, 1), (q_1, 0) \rangle \}$

Example: unary incrementer



- Given input of a series of n 1's on tape, this TM will leave $(n + 1)$ 1's as output
- $\Sigma = \{ 1 \}$
- $Q = \{ q_0, q_{accept} \}$
- $\delta = \{ \langle (q_0, 1), (q_0, R) \rangle, \langle (q_0, \#), (q_{accept}, 1) \rangle \}$

Unary adder



2. TM computations

- A *configuration* is a snapshot of the arrangement of the TM's components at an instant in time
- *Configuration* $(q, t_{left}, t_{head}, t_{right})$: state; tape contents to left of head; symbol at head; contents to right of head
- The assertion that configuration $C = (q, x, a, y)$ of TM M yields configuration $C' = (q', w, b, z)$ in one step is written $C \Rightarrow_M C'$
- Intuitively, $C \Rightarrow_M C'$ means that from configuration C , application of $\delta(q, t_{head})$ yields configuration C'

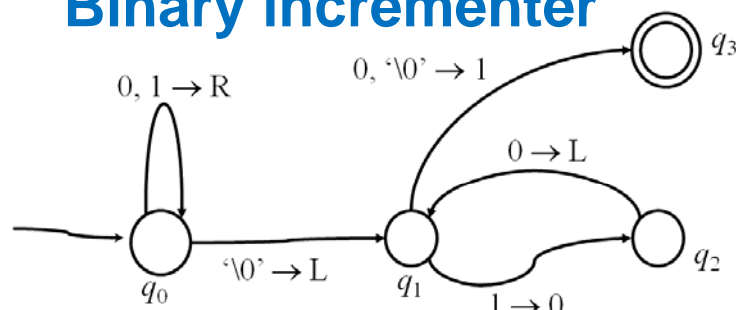
Transitions between configurations

- When $C = (q, x, a, y)$ and $C' = (q', w, b, z)$, $C \Rightarrow_M C'$, iff $\delta(q, t_{head}) =$
 - (q', b) and $w = x, z = y$, or
 - (q', L) and $w = \text{left}(x, \text{length}(x) - 1)$,
 $b = \text{right}(x, 1), z = ay$, or
 - (q', R) and $w = xa, b = \text{left}(y, 1)$,
 $z = \text{right}(y, \text{length}(y) - 1)$
- Intuitively, under δ either b replaces a , or else the tape head moves left or right

Finite TM computations

- A *computation* is a sequence of configurations C_0, C_1, \dots, C_n , where $(\forall i < n) C_i \Rightarrow_M C_{i+1}$
- By convention that input is initial contents of tape and output is final contents, a TM computes a function $f: \Sigma^* \rightarrow \Sigma^*$
- Tape is erased between computations; hence $C_0 = (q_0, \lambda, x[1], \text{right}(x, \text{length}(x) - 1))$ where x is input
- Some TM computations may be non-halting (infinite)
- A TM that halts on all inputs computes a total function

Binary incrementer



1. Scan to right, then step left once
2. While input is 1
write 0 and move left
3. Read 0 or '\0'
4. Write 1 and accept

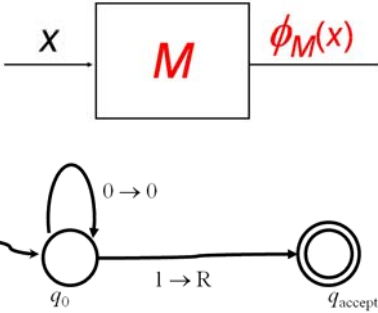
Multitape TMs

- For k -tape TM, we have
 $\delta : Q \times (\Sigma \cup \Gamma)^k \rightarrow Q \times (\Sigma \cup \Gamma \cup \{L, R\})^k$
- *Theorem:* For any k -tape TM M_k , an equivalent single-tape TM M exists
- *Proof sketch:* construct M to simulate M_k
 - M 's tape contains contents of all M_k 's tapes, delimited by blanks ('#')
 - k head locations on M_k are denoted by special symbols using alphabet k times as large as Γ (e.g., $a, b, \dots, \underline{a}, \underline{b}, \dots$)

TMs compute partial functions

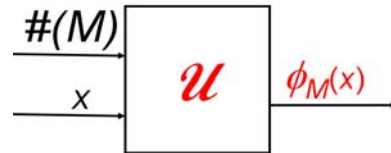
- Let $\phi_M(x)$ be the string left on the tape, after M halts, on input x
- Consider TM M at right:

$$\phi_M(x) = \begin{cases} x & \text{if } x[1] = 1 \\ \uparrow & \text{otherwise} \end{cases}$$


- M computes a *partial but not total function*; i.e., $\phi_M(x)$ is undefined for some x
- A TM that always halts computes a *total function*

Universal Turing machines

- Suppose TM U takes as inputs a pair:
 - $\#(M)$, encoding of M ;
 - $x \in \Sigma^*$, an input to TM M
- Then for all inputs $(\#(M), x)$, U outputs what M would output on input x
- Then U is called a *universal Turing machine*
- *Example:* Any general-purpose stored-program computer is equivalent to a universal TM
- *Theorem:* A universal TM exists



3. Turing decidability

- Given TM M , $x \in \Sigma^*$, let $\phi_M(x)$ denote the tape contents of M after a halting computation with input x
- Given function $f: \Sigma^* \rightarrow \Sigma^*$, M is said to *compute* f iff for all x in $\text{Dom}(f)$, M eventually halts on x and outputs $f(x)$, while for all other x , M hangs
- Function f is called *Turing computable (recursive)* if there is a TM M that computes f
- Language or problem L is said to be *decidable* if its characteristic function is Turing computable

Language accepted by a TM

- Every TM M partitions Σ^* into inputs that
 - (a) take M into the accept state and halt or
 - (b) take M into the reject state and halt or
 - (c) cause M to loop forever
- We say that the language (a) is *accepted* by M
- The interpretation of a TM as an *accepter* is different from the interpretation as a *transducer* (input transformer)
- $L(M) = \{x \mid \pi_1(\delta^*(q_0, x)) = q_{accept}\},$
 $\pi_1(x_1, \dots, x_n) = x_k$

Regular languages are decidable

Theorem: If L is regular, then L is recursive

Proof (by construction, defining TM M):

1. Let A be a DFA that accepts L
2. M has A 's transition function modified to change labels for every $a \in \Sigma$ to (a, R) , scanning the tape to the right and going through A 's states
3. Add states, q_{accept} and q_{reject}
4. For each accepting state of A , add to M an edge $(\#, \#)$ that goes from that state to q_{accept} ; for each other add edge $(\#, \#)$ to q_{reject}
5. Thus M accepts exactly the strings in L .

CFLs are TM-decidable

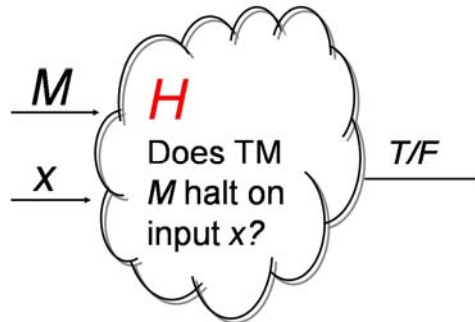
- *Theorem:* If L is context-free then L is recursive
- *Proof:* Let PDA A be a deterministic acceptor of L . Construct a 2-tape TM M from A as follows:
 - $Q_M = Q_A$, $\delta_M = \delta_A$ except as follows
 - For every stack-popping operation of A , let M travel to rightmost cell of tape 2, reading it and replacing it with '#'; simulate push by appending symbol at rightmost cell
- Then M simulates A and accepts L

4. Undecidability

- Some languages are not decidable, e.g., the set of descriptions of TMs that halt on a given input
- Some undecidable languages are recognizable
- *Goal:* to find the limits of algorithmic computation, i.e., the computable functions
- *Highest result:* *Nothing interesting* about the language accepted by an arbitrary TM or program can be determined by looking at its code or structure

The Halting Problem

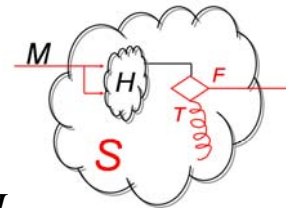
- Consider the decision problem or language *HALT*, consisting of the set of pairs (M,x) s.t. the TM with description M halts on input x
- Is there a TM that decides *HALT*?



Theorem: *HALT* is undecidable

Proof:

1. Suppose *HALT* is decidable.
2. Then construct TM S from *HALT*-decider H , where S makes a copy of its input M , feeds M and M to H , loops forever if $\phi_H(M,M) \downarrow$, halts if $\phi_H(M,M) \uparrow$
3. Consider whether S halts on input S . If $\phi_S(S) \downarrow$ then $\phi_S(S) \uparrow$ and conversely -- a contradiction.
4. Since S is clearly constructible except for component H , therefore H cannot be constructed
5. Hence *HALT* is undecidable.

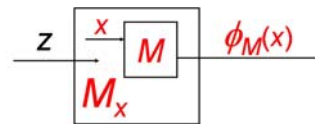


Cantor's and Turing's proofs

- Consider an infinite table listing inputs x across the top and TMs M down the side
- The entries tell whether M halts on x
- Consider TM S that is designed to halt if its input is a TM that hangs on its own description and to hang if it halts
- The behavior of S is the bitwise negation of the diagonal of our table; but by definition no entry in the table can have this behavior (see Cantor proof)
- Hence no TM M can detect halting behavior

Hardwiring a TM

- Any TM M can be *hardwired* to a given input, x , i.e., converted to M_x that ignores its input and outputs $\phi_M(x)$

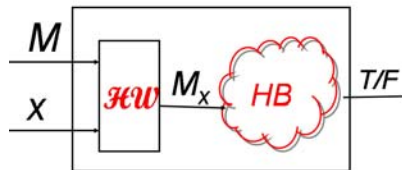


- Let HW be a TM that performs this conversion, by generating a description of a TM M_x that discards its input and replaces it with x



Halt-on-blank

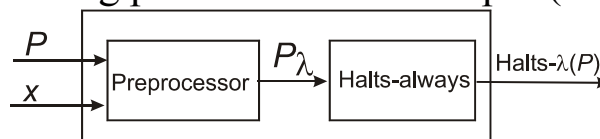
- Problem *HB*: Decide whether input is in $\{ M \mid M \text{ is a TM that halts on input } \lambda \text{ (blank)} \}$
- $HALT(M, x)$ reduces to *HB* as follows



- *Theorem*: The halt-on-blank problem is undecidable
- *Proof*: *HB* is reducible to *HALT* because *H* can be constructed from *HB*; but *H* cannot exist, so *HB* cannot exist

Halting on all inputs

- Let $HALT-ALWAYS(M)$ be the assertion that TM M halts on all inputs
- *Theorem*: $HALT-ALWAYS$ is undecidable
- *Proof*: The following TM decides the undecidable halting problem for blank input ($HALT \lambda$)



- ... where the preprocessor transforms M into TM M_λ that simulates TM M running with blank input

Reducibility

- Problem B is *reducible* to problem A iff a solution to A enables a solution to B
- Intuitively, B is at least as hard as A
- *Example*: multiplication is reducible to addition
- If B is undecidable and B is reducible to A , then A is undecidable
- By showing that the Halting Problem is reducible to problem P , we can show that P is undecidable too

Undecidability of TM behavior

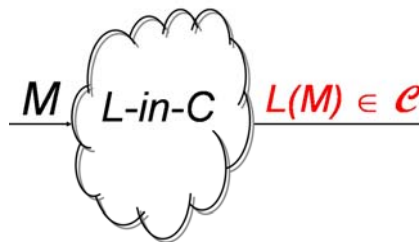
- Not only is the problem of deciding whether a given program halts uncomputable...
- ...but also no programs exist to decide *any* useful (nontrivial) property of programs
- By “nontrivial” we mean any behavioral property that does not hold for *all* programs or *no* programs
- *Examples*:
 - Is $L(M)$ infinite? Regular? CF?
 - Is x in $L(M)$?
 - Is $L(M_1)$ a subset of $L(M_2)$?

Rice's Theorem

Thm: For any nontrivial class C of r.e. languages (e.g., regular languages), and any TM M , $(L(M) \in C)$ is undecidable (All interesting behavioral properties of TMs are undecidable).

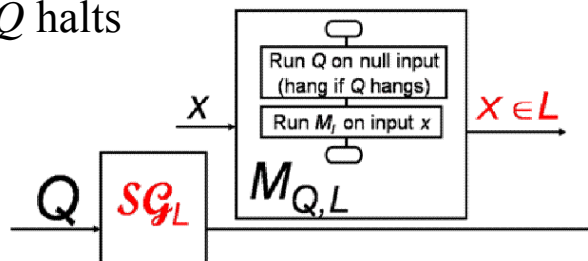
Proof:

1. Suppose $(L \in C)$ were decidable.
2. Then some program $L\text{-in-}C$ decides this problem.



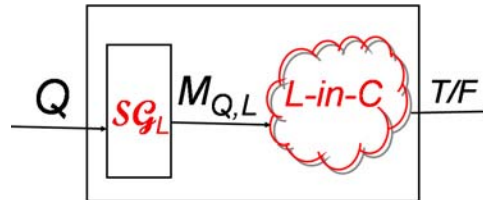
Rice proof, ii

3. Note that TM generator SG_L is constructible, generating from TM Q and L -acceptor M_L a TM $M_{Q,L}$, that loops forever if Q hangs, otherwise tells whether its input is in L . Hence $L(M_{Q,L}) = L$ iff Q halts



Rice proof, iii

4. Then construct from S_{GL} and $L\text{-in-}C$ the following TM:



5. But this TM decides the *Halt-always* problem, which is undecidable.
6. Hence $L\text{-in-}C$ is undecidable.

Linear bounded automata

- LBA: TM restricted to tape space occupied by input or proportional to size of input
- LBAs accept a strict superset of CFLs and a strict subset of decidable languages
- LBA-accepted languages are generated by unrestricted grammars

References

D. Harel. *Algorithmics*. Addison Wesley, 1992.

J. Hopcroft, R. Motwani, J. Ullman. *Introduction to Formal Language Theory and Automata*. Addison Wesley, 2007.

Daniel I. I. Cohen. *Introduction to Computer Theory*. 1997.

Peter Linz. *Introduction to Formal Languages and Automata*. 2006.