

Lecture 8: Turing Machines and Undecidability

David Dill

Department of Computer Science

Outline

- Weakening the TM model
 - semi-infinite tape
 - multi-stack PDAs
 - two-counter machines
- Recursively enumerable vs. RE.
- A language that is not RE.
- Properties of recursive languages

Equivalence of NTMs and DTMs

Thm Every language accepted by a DTM is also accepted by an NTM.

Sketch Obviously, NTMs are no less powerful than DTMs.

We can simulate an NTM on a multi-tape DTM using *breadth-first search* on the tree of IDs generated by the possibilities of the NTM. If the NTM accepts an input (either by final state or by halting), there is at least one finite path in the tree.

Why not *depth-first search*? Because the machine can go down an infinite possible computation forever, and never get around to doing one of the finite (halting) ones.

The DTM accepts iff the simulated NTM enters a final state.

(Rejection? No path of the NTM has an accepting state. Paths may be finite or infinite.)

Simulation overhead: Suppose the maximum number of alternative moves in the NTM is m , and that the NTM takes n steps to accept its input. Then the number of nodes in the tree that must be explored is nm^n .

The cost is “single exponential” $O(2^{p(n)})$, where $p(n)$ is a polynomial function.

$P = NP$

Running time of an NTM on an input: Shortest computation to an accepting state (either a final state, or by halting).

Time complexity of an NTM – maximum running time over all inputs of length n .

Can we simulate an NTM in polynomial time on a DTM?

This is equivalent to the “P vs. NP” question.

Polynomial time – $T(n)$ (deterministic or nondeterministic) is $O(p(n))$ for some polynomial $p(n)$

P – languages accepted by a DTM whose time complexity is polynomial.

NP – languages accepted by an NTM whose time complexity is polynomial.

$P = NP$ iff, for each NTM, M with polynomial time complexity, there is a DTM, M' with the same language and polynomial time complexity.

At this point, no one knows, but “no” might be a good bet.

Weakening the TM Model

You can seemingly cripple a Turing machine without really reducing its power as a language recognizer.

Motivation: Sometimes it is easier to prove something is undecidable by using one of these models.

Semi-infinite tape The tape can be infinite to the right, but have a “beginning” on the left.

Construction: Use 2 tracks on semi-infinite tape to “fold” the infinite tape.

When the head gets to the fold, jump to the other track and reverse directions.

Multi-stack PDAs

A single-stack PDA is more powerful than a DFA but weaker than a TM.

(PDAs accept exactly the context-free languages, which are languages that can be expressed by context-free grammars.)

What happens if we generalize the PDA model to have more than one stack?

A k -PDA is a *deterministic machine* with k stacks.

Moves depend on state, input, tops of all stacks.

Moves modify state, push strings on all stacks.

Assume we have a special *end marker* $\$$ for the input.

2PDAs are as powerful as Turing Machines

Theorem For every Turing machine M , there is a 2-stack PDA S such that $L(M) = L(S)$.

Idea: Use the two stacks to represent the tape to the left and right of the tape head.

Stack 1: stuff to the left of the head, except for infinite blanks to the left.

Stack 2: stuff from the head and to the right, except for infinite blanks to the right.

Counter machines

Moves depend on

- State
- Input
- Counter 1 is 0
- Counter 2 is 0

Moves change by

- Changing state
- Incrementing counters by 1
- Decrementing by 1 (but not below 0).

Two-counter Machines

Amazingly, even a machine with two unbounded integer counters has the same power as a Turing machine.

Idea: Counters are unbounded, so they can represent arbitrarily many bits of data. Use arithmetic operations to manipulate the counters like TM tapes.

Recursive vs. RE

Recursively Enumerable Language – accepted when TM enters a final state, rejected otherwise (even if TM loops).

Accepting by halting is natural for this concept.

Recursive Language – accepted by a TM that always halts, even when it rejects the input.

Acceptance by final state is appropriate for this concept.

The membership problem for a language is *decidable* iff the language is *recursive*.

A TM that always halts is called an *algorithm*.

Turing machines can make/modify other Turing machines

Turing machines can be encoded as strings.

Strings can be read and written by Turing machines (so the definition of the machine can be changed).

Turing machines can *simulate* other Turing machines.

“Turing machine virtualization”

String encoding and simulation of Turing Machines

Turing machines as strings: Find an encoding for Turing machines as strings of 1's and 0's (binary encoding).

You should all believe this is possible, since everything in our computers is encoded in binary. I'll skip the specifics.

Each TM can be written as a string in $\{0, 1\}^*$. The string encoding of TM M will be $\langle M \rangle$.

Turing machines as numbers:

Order strings $x < y$ if x is shorter than y ; if they are of the same length, use lexicographical order.

The i th string in that order is $\langle M_i \rangle$ encoding M_i .

Some of these machines are ill-defined. For those, say $L(M_i) = \emptyset$.

By this definition, every Turing machine will have an index.

L_d is not RE

Define the *diagonal language* $L_d = \{\langle M_i \rangle \mid \langle M_i \rangle \notin L(M_i)\}$

“All those TMs that do not accept themselves.”

	M_0	M_1	M_2	...
$\langle M_0 \rangle$	A	R	A	...
$\langle M_1 \rangle$	A	R	A	...
$\langle M_2 \rangle$	A	A	A	...
\vdots	\vdots	\vdots	\vdots	\ddots

$$L_d = \{M_2, \dots\}$$

Thm: L_d is not recursive.

proof: $L(M_i)$ differs from L_d on input M_i (so $L(M_i) \neq L_d$) for all i . Since every TM has an index, there is no TM that accepts L_d

Recursive Languages

A language is recursive if it is accepted by a TM that always halts.

So: Accept means it halts in a final state, Reject means it halts in but does not visit a final state.

Theorem: Recursive languages are closed under complementation.

Construction: If the language is recursive, there is a TM that accepts it and always halts.

⟨ Picture of TM with Yes/No answers, with answers reversed. ⟩

- Make all accepting states non-accepting.
- Add a new accepting state.
- Whenever the δ function is undefined in a non-final state, change it to go to a new final state.

Recursive Languages

Theorem: If L and \bar{L} are both RE, L is recursive (and so is \bar{L}).

Construction:

Simulate M_L and $M_{\bar{L}}$ in parallel in input w . (This is easy on a 2-tape Turing machine.)

⟨ Picture of a “yes” TM and a “no” TM in parallel, making a “yes/no” TM. ⟩

One of the two machines must accept w . If it's M_L , accept.

If the accepting machine is $M_{\bar{L}}$, reject.