Lecture 12 – Undecidable problems, Post's correspondence problem

NTIN071 Automata and Grammars

Jakub Bulín (KTIML MFF UK)

Spring 2024

** Adapted from the Czech-lecture slides by Marta Vomlelová with gratitude. The translation, some modifications, and all errors are mine.*

Recap of Lecture 11

- Recursively enumerable languages are exactly those generated by (Type 0) grammars
  - TM to G: simulate moves on a reversed non-terminal copy of ω, generate sufficient space, cleanup if accepting state
  - G to TM: generate all strings, check if any of them represents a valid derivation of ω (sentential forms separated by $\#$)
- Context-sensitive languages:
  - context-sensitive grammars are equivalent to monotone grammars
  - Linear Bounded Automaton (LBA): nondeterministic TM with tape limited to the length of input
  - constructions: monotone grammar to LBA, LBA to monotone grammar
- Intro to computability: an overview
- decision problem $\leftrightsquigarrow$ the language of all 'YES' instances
- machine-readable encoding of TMs

The Diagonal language

Let $\mathrm{decode}(w)$ be the TM M such that $\mathrm{code}(M) = w$. (Recall: if w is not a valid code, then $\mathrm{decode}(w)$ is a fixed one-state TM with no instructions.) Then:

$$L_D = \{w \mid w \notin L(\mathrm{decode}(w))\}$$

Theorem

L_D is not recursively enumerable.

Proof idea: there cannot exist a TM recognizing L_D: running it on its own code would lead to Barber's paradox

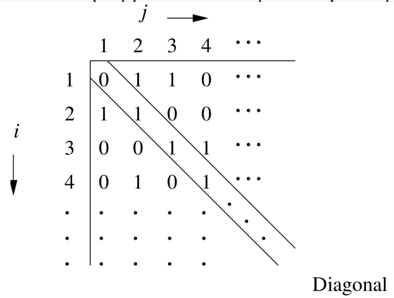
> *"The program accepts all programs that don't accept themselves. Does the program accept itself?"*

Proof that $L_D = \{w \mid w \notin L(\mathrm{decode}(w))\}$ is not RE

Proof: Assume for contradiction that $L_D = L(M)$ for some M. Let $w = \mathrm{code}(M)$. Then $L_D = \{w \mid w \notin L(M)\}$. Is $w \in L(M)$?

$$w \in L(M) \;\Leftrightarrow\; w \in L_D \;\Leftrightarrow\; w \notin L(M) \qquad \square$$

Why 'diagonal'? A variant of Cantor's diagonal argument. Order all TMs by $M_i = \mathrm{decode}(w_i)$. Does M_i accept w_i?



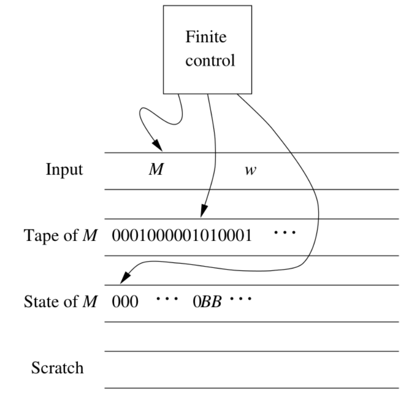
A TM for L_D would be one of the rows but differs from each row in the diagonal element. (Same as the proof that $\mathbb{R}$ is uncountable.)

The Universal Turing Machine

The Universal Turing Machine U can simulate any TM (given by its code) on any input. More precisely, U accepts exactly inputs of the form $\langle \mathrm{code}(M), w \rangle$ where $w \in L(M)$.

The construction: four tapes

1. input tape (w and the encoded transitions of M)
2. simulated tape of M, symbols encoded as 0^i, separated by 1s
3. state of M, again represented by 0^i
4. scratch tape



The operation of U

Initialize:

- Check if the input code is valid, if not, halt without accepting
- Initialize Tape 2 with w in its encoded form: 10 for 0 in w, 100 for 1
  Blanks are left blank and replaced with 1000 only 'on demand'
  Move 2nd head to the first simulated cell.
- Place 0 (the start state of M) on Tape 3.

Simulate moves of M:

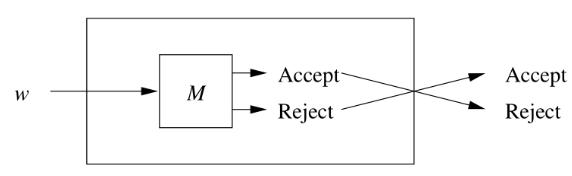
- Search Tape 1 for the appropriate transition $0^i10^j10^k10^\ell10^m$, where 0^i on Tape 3, 0^j on Tape 2.
- Change the content of Tape 3 to 0^k.
- Replace 0^j on Tape 2 by 0^ℓ. Scratch tape to manage spacing.
- Move head on Tape 2 to the next 1 left or right, depending on m.

Termination: If M has no transition matching simulated state & tape symbol, halt without accepting. If M enters accepting state, U accepts.

Recursive languages are closed under complement

Lemma

If L is recursive, then $\overline{L}$ is recursive as well.



Proof: Given M deciding L, construct M' deciding L'. Since M always halts, if it does not accept, the reason is missing transition.

M' has a single, new accepting state q_{ACCEPT}. For every non-accepting state of M and every tape symbol X such that $\delta(q, X)$ is undefined, redefine $\delta'(q, X) = (q_{\text{ACCEPT}}, X, L)$.

Clearly, $L(M') = \overline{L}$. Since M is guaranteed to halt, so is M'. $\square$

Post's theorem

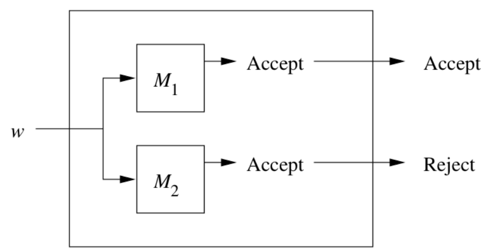
Theorem

L is recursive iff both L and $\overline{L}$ are recursively enumerable.

Proof: $\Rightarrow$ Follows from the lemma.

$\Leftarrow$ Let $L = L(M_1)$ and $\overline{L} = L(M_2)$. For an input w, simulate both M_1 and M_2 (two tapes, states with two components). L and $\overline{L}$ are complementary, one of M_1 or M_2 will halt and accept.

- If M_1 accepts, accept.
- If M_2 accepts, reject. $\square$



The Universal language and its undecidability

$L_U = \{\langle \mathrm{code}(M), w\rangle \mid w \in L(M)\}$ RE but not R

$\overline{L_D} = \{w \mid w \in L(\mathrm{decode}(w))\}$ RE but not R

$L_D = \{w \mid w \notin L(\mathrm{decode}(w))\}$ not RE

Theorem

The Universal language is recursively enumerable but not recursive. Same is true for complement of the Diagonal language.

Proof:

- L_U is RE: it is recognized by the Universal TM U
- $\overline{L_D}$ is RE: rewrite input w to $\langle w, w\rangle = w111w$, then run on U
- $\overline{L_D}$ is not recursive: if it were, by Post's theorem L_D would be RE (actually R), but we know it is not
- L_U is not recursive: if it were, $\overline{L_D}$ would be recursive (rewrite w to $\langle w, w\rangle$, run on the hypothetical M deciding L_U) $\square$

Reductions between decision problems

Definition

A reduction R is an algorithm mapping all instances of P_1 to instances of P_2 that always halts, and for every instance w of P_1 outputs an instance $R(w)$ of P_2 such that:

- w is a YES instance of P_1 iff $R(W)$ is a YES instance of P_2
- w is a NO instance of P_1 iff $R(W)$ is a NO instance of P_2

(Technically, $R = f_M$ for some TM M that always halts.)

Example The mapping $w \rightsquigarrow \langle w, w \rangle = w111w$ (from the previous proof) can clearly be done algorithmically. It is a reduction from $\overline{L_D}$ to L_U (and also from L_D to $\overline{L_U}$).

Only easy reduce to easy, hard only reduce to hard

Theorem

If there is a reduction from P_1 to P_2, then:

(i) *If P_1 is not decidable then neither is P_2.*

(ii) *If P_2 is decidable, then so is P_1.*

(iii) *If P_1 is not partially decidable then neither is P_2.*

(iv) *If P_2 is partially decidable, then so is P_1.*

(i&ii) Let P_1 be undecidable. If P_2 were decidable, we could combine the reduction from P_1 to P_2 with the algorithm deciding P_2 to construct an algorithm that decides P_1.

(iii&iv) Assume P_1 is not partially decidable, but P_2 is. Similarly as above, we could combine the reduction and the algorithm for P_2 to get an algorithm partially deciding P_1–a contradiction.$\square$

“Does the given program halt for the given input?’

An instance of the Halting Problem Halt: $\langle \mathrm{code}(M), w\rangle \in \{0,1\}^*$. The answer is YES iff M halts on input w; otherwise it is NO.

Theorem

The Halting Problem is undecidable.

(Note that it is partially decidable: we can simulate using U.)

Proof: Reduce the undecidable problem $\overline{L_D}$ to Halt. Given an instance w of $\overline{L_D}$, let $M = \mathrm{decode}(w)$. Modify M to get M' such that if M halts without accepting, M' goes to an infinite loop.

Set $R(w) = \langle \mathrm{code}(M'), w\rangle$. Clearly, it can be done algorithmically.

- If $w \in \overline{L_D}$, i.e. $w \in L(M)$, then M' accepts (thus halts) on w.
- If $w \notin \overline{L_D}$, i.e. $w \notin L(M)$, then either M doesn't halt or halts without accepting. In either case M' doesn't halt on w. $\square$

Accepts no inputs?

- $L_e = \{\text{code(M)} \mid L(M) = \emptyset\}$
- $L_{ne} = \{\text{code(M)} \mid L(M) \neq \emptyset\} = \overline{L_e}$

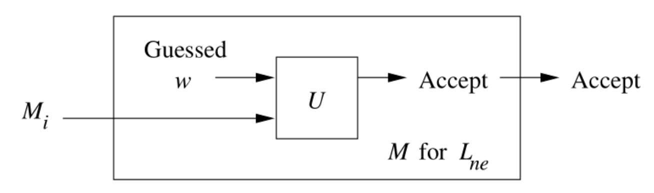
Theorem

(i) *L_e is not recursively enumerable.*

(ii) *L_{ne} is recursively enumerable but not recursive.*

Proof: As $L_{ne} = \overline{L_e}$, (i) follows from (ii) by Post's theorem.

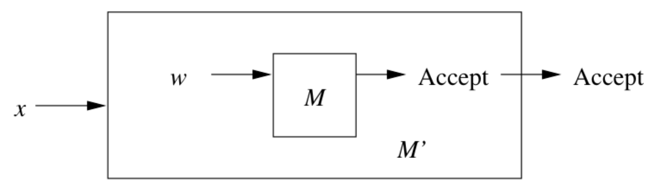
L_{ne} is RE: nondeterministically guess $w \in L(M)$, verify using U



Proof cont'd

L_{ne} is not recursive: reduction from undecidable $\overline{L_D}$

Given $w = \mathrm{code(M)}$, $R(w)$ is a TM M' that ignores its input, rewrites the input tape with w, and simulates M on w.



- If $w \in \overline{L_D}$, i.e. $w \in L(M)$, then $R(w)$ always accepts.
- If $w \notin \overline{L_D}$, i.e. $w \notin L(M)$, then $R(w)$ never accepts. $\square$

Rice's theorem

Which properties of programs are decidable?

None of them!

(Except for trivial properties true/false for all programs.)

We have all the tools, but not the time to prove Rice's theorem.

"Theoretically, static analysis of programs cannot be done automatically?"

Summary of Lecture 12

- the Diagonal language L_D is not recursively enumerable
- the Universal language L_U, the Universal TM: simulate any M on any w
- recursive languages are closed under complement
- Post's theorem: L recursive iff both $L, \overline{L}$ are RE
- L_U, $\overline{L_D}$ are recursively enumerable but not recursive
- reductions between decision problems
- the Halting problem is undecidable
- (Rice's thm: nontriv. properties of programs are undecidable)