Formal Languages, Automata and Computation

Advanced Topics in Computability

RICE'S THEOREM – MOTIVATION

Consider the following undecidable languages:

- $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Phi\}$
- $TOTAL_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Sigma^*\}$
- $REGULAR_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is regular}\}$
- $L_{0101010} = \{\langle M \rangle \mid M \text{ is a TM and } 0101010 \in L(M)\}$

QUESTION

What do these questions about languages have in common, so that they are all undecidable?

- They ask whether the language defined by a TM has a certain property.
- The properties are "nontrivial".
  - What is a "nontrivial" property?

IDEA

We can generalize the undecidability proofs into a meta-theorem that works for all languages that talk about nontrivial properties of Turing machine languages.

What is a Nontrivial Property?

Definition (Property)

A language $\mathcal{P}$ is called a property of Turing machine languages iff

- $\mathcal{P} \subseteq \{\langle M \rangle \mid M \text{ is a TM}\}$
- For any two TMs M_1, M_2, if $L(M_1) = L(M_2)$ then $\langle M_1 \rangle \in \mathcal{P}$ iff $\langle M_2 \rangle \in \mathcal{P}$.

Definition (Nontrivial Property)

A language $\mathcal{P}$ which is a property of Turing machine languages is nontrivial iff:

- There is a TM M_1 such that $\langle M_1 \rangle \in \mathcal{P}$, and
- There is a TM M_2 such that $\langle M_2 \rangle \notin \mathcal{P}$.

- All these languages are nontrivial
  - $E_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Phi\}$
  - $TOTAL_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) = \Sigma^*\}$
  - $L_{0101010} = \{\langle M \rangle \mid M \text{ is a TM and } 0101010 \in L(M)\}$

Rice's Theorem

Theorem

Every language $\mathcal{P}$ which is a nontrivial property of Turing machine languages is undecidable!

Proof – Preliminaries

Assume a nontrivial property language $\mathcal{P} \subseteq \{\langle M \rangle \mid M \text{ is a TM}\}$. We want to show $\mathcal{P}$ is undecidable.

Consider the following two Turing machines:

- Let M_ϕ = "On input x: *reject*".
  - We can assume $\langle M_\phi \rangle \notin \mathcal{P}$.
- Let M_P be a TM such that $\langle M_P \rangle \in \mathcal{P}$.
  - M_P exists because $\mathcal{P}$ is nontrivial.

Proof by reduction from A_{TM} to $\mathcal{P}$

1. Assume we have a decider R_P for $\mathcal{P}$.
2. We show that using R_P we can construct a decider S for A_{TM}.

$S =$ "On input $\langle M, w \rangle$

1. Construct a TM M_w as follows:
   $M_w =$ "On input x:
   1. Run M on w.
      If M rejects then *reject*
   2. Else run M_P on x.
      If M_P accepts then *accept*."
2. Run R_P (the decider for $\mathcal{P}$) on $\langle M_w \rangle$
3. If R_P accepts then *accept*
   If R_P rejects then *reject*"

- If M accepts w, then $L(M_w) = L(M_P)$. So $\langle M_w \rangle \in \mathcal{P}$.
- If M does not accept w, then $L(M_w) = \Phi$.
  So $\langle M_w \rangle \notin \mathcal{P}$.
- So if R_P decides $\mathcal{P}$, then S decides A_{TM}.
- But we know the S does not exist, so R_P can not exist either.
- Conclusion: $\mathcal{P}$ is an undecidable language.

Applying Rice's Theorem

- The following languages are all undecidable:
  - $EPSILON_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } \epsilon \in L(M)\}$
  - $CFL_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is a CFL}\}$
  - $DECIDABLE_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ is decidable}\}$
  - $PAL_{TM} = \{\langle M \rangle \mid M \text{ is a TM and } L(M) \text{ contains all palindromes}\}$
- Rice's Theorem is a very powerful tool
  - Very Important: we need to be checking a property of the language of the TM, not a property of the TM and the behaviour of the TM.

Common Pitfalls

Rice's Theorem can not be applied to the following languages:

- $ALL = \{\langle M \rangle \mid M \text{ is a TM }\}$
  - Note that ALL is decidable!
  - There is no language property involved here.We need to check a property of the representation!
- $TWICE = \{\langle M \rangle \mid M \text{ is a TM that visits the initial state more than twice}\}$
  - Again, this is not a question about the language defined by M but rather on the behaviour of M (Undecidable)
- $EQ_{TM} = \{\langle M_1, M_2 \rangle \mid M_1 \text{ and } M_2 \text{ are TMs and } L(M_1) = L(M_2)\}$
  - Again, this is not a question about the property of a language. (Undecidable)

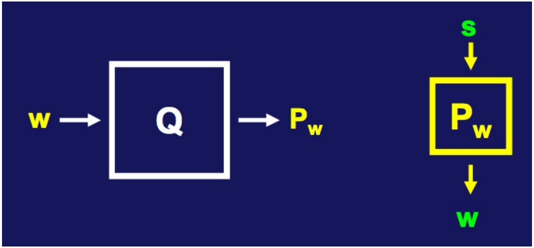
Self-Reference

- Can automata self-reproduce?
  - What do you mean?
    - Living things are "machines" and they reproduce!

Lemma

There is a computable function $q : \Sigma^* \longrightarrow \Sigma^*$ where

- if w is any string,
- $q(w)$ is the description of a Turing machine P_w that prints out w and halt.



Self-Reference

Lemma

There is a computable function $q : \Sigma^* \longrightarrow \Sigma^*$ where if w is any string, $q(w)$ is the description of a Turing machine P_w that prints out w and halt.

Proof:

The following TM Q computes $q(w)$.
$Q =$ "On input string w:

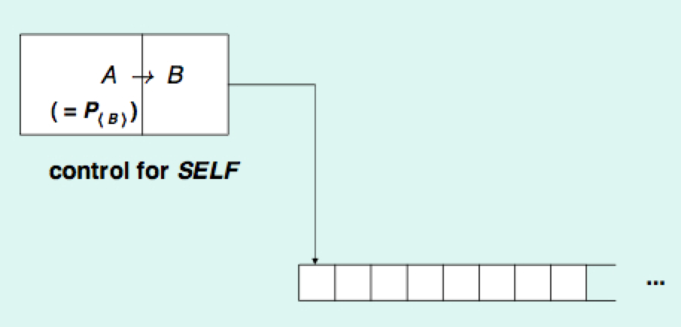
1. Construct the following Turing machine P_w
   $P_w =$ "On any input:
   1. Erase input.
   2. Write w on tape.
   3. Halt."
2. Output $\langle P_w \rangle$."

The TM SELF

- Next we build a TM, *SELF*, that ignores its own input and prints out a copy of its description.
- Print out this sentence.
  - Not clear what "this" refers to.
- Print out two copies of the following, the second one in quotes: "Print out two copies of the following, the second one in quotes:"
- `((lambda (x) (list x (list (quote quote) x))) (quote (lambda (x) (list x (list (quote quote) x)))))` (Lisp)

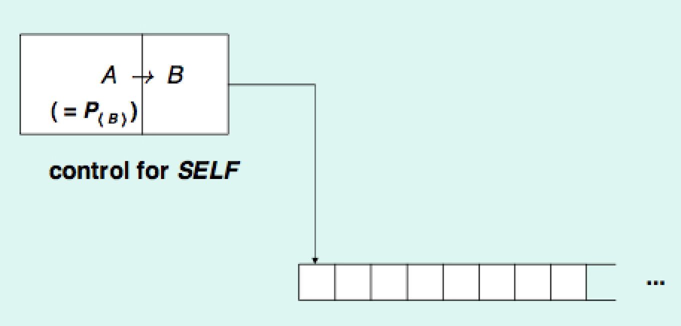
```
STk> ((lambda (x) (list x (list (quote
quote) x))) (quote (lambda (x) (list x
(list (quote quote) x)))))
((lambda (x) (list x (list (quote quote)
x))) (quote (lambda (x) (list x (list
(quote quote) x)))))
STk>
```

A TM that prints itself

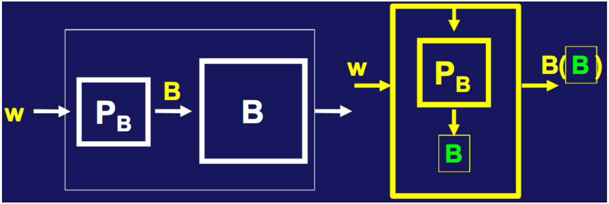


- Part A runs first and upon completion passes control to part B.
- The job of A is to print a description of B on the tape (hence $A = P_{\langle B \rangle}$).
- The job of B is to print out a description of A.
- The tasks are similar, but are carried out differently.

A TM THAT PRINTS ITSELF



- If B can obtain $\langle B \rangle$, it can apply q to that and obtain $\langle A \rangle$.
- What how can B obtain $\langle B \rangle$?
- Well, it was printed on the tape, just before A passed control to B.
- So, B computes $q(\langle B \rangle) = \langle A \rangle$ and combines these and writes its own description $\langle AB \rangle$.



- $A = P_{\langle B \rangle}$: A is the TM that prints out the description of B (But we do not have B yet!)
- $B =$ "On input $\langle M \rangle$ where M is a portion of a TM:
    1. Compute $q(\langle M \rangle)$, (find the description of the machine which prints $\langle M \rangle$)
    2. Combine the result with $\langle M \rangle$ to make a complete TM.
    3. Print the description of this TM and halt."

How SELF Behaves

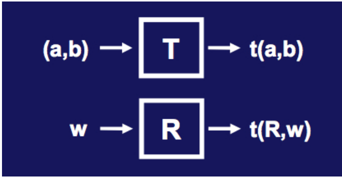
1. First A runs. It prints $\langle B \rangle$.
2. B starts. It looks at the tape and finds its input $\langle B \rangle$.
3. B computes $q(\langle B \rangle) = \langle A \rangle$ and combines that with $\langle B \rangle$ into a TM description $\langle SELF \rangle$.
4. B prints this description and halts.

Theorem 6.3 – The Recursion Theorem

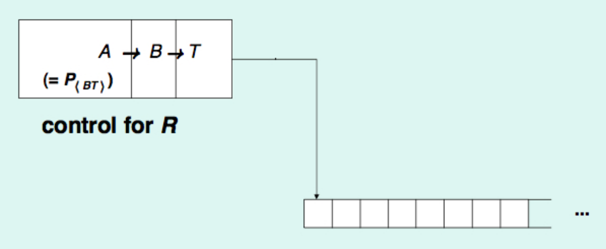
Let T be a TM that computes a function $t : \Sigma^* \times \Sigma^* \longrightarrow \Sigma^*$. There is a TM R that computes $r : \Sigma^* \longrightarrow \Sigma^*$, where for every w,

$$r(w) = t(\langle R \rangle, w)$$

- What is this Theorem saying?
- Informally, a TM can obtain its own description and compute with it.
- To make a TM, that can obtain its own description and then compute with it
  1. Make a TM T that receives the description of the machine as an extra input.
  2. Then the recursion theorem produces a new machine, R which operates as T does, with $R's$, description filled in automatically.



- We construct a machine with 3 parts: A, B and T.



- A is the TM $P_{\langle BT \rangle}$, described by $q(\langle BT \rangle)$
  - Technical point: We redesign q so that $P_{\langle BT \rangle}$ writes its output following any preexisting string on the tape.
- So, after A runs, the tape contains $w\langle BT \rangle$
- B examines the tape and applies q to $\langle BT \rangle$ getting $\langle A \rangle$.
- B then combines A, B and T into a single machine and obtains its description $\langle ABT \rangle = \langle R \rangle$
- It encodes these as $\langle R, w \rangle$ and places it on the tape and passes the control to T.

Significance of the Recursion Theorem

- It is yet another handy tool for solving certain problems in the theory of algorithms.
- When you are designing a TM M, you can "make a call" to "obtain own description $\langle M \rangle$" and use this description in the computation.
  - Just print out the description
  - Count the number of states in M.
  - Simulate M.
- Consider the TM

  $T =$ "On input $\langle M, w \rangle$:

  1. Print $\langle M \rangle$ and halt."

  The recursion theorem tells us how to construct R which on input w, behaves just like T on input $\langle R, w \rangle$.
- Thus R prints the description of R, exactly what is required of the machine $SELF$.
- Technology for Computer Viruses (:-)

SIGNIFICANCE OF THE RECURSION THEOREM

THEOREM 6.5

A_{TM} is undecidable.

PROOF

- Suppose H decides A_{TM}, we construct B:
- $B =$ "On input w:
  1. Obtain, via the recursion theorem, own description $\langle B \rangle$.
  2. Run H on input $\langle B, w \rangle$.
  3. Do the opposite of what H says.
     - *accept* if H rejects.
     - *reject* if H accepts.
- B conflicts with itself – hence can not exist
- H can not exist.

The Fixed-point version of the Recursion Theorem

- A fixed-point of a function is a value, that is not changed by the application of a function, e.g.,
  - $f(x) = \sqrt{x}$ has a fixed-point 1.
  - $f(y(x)) = y'(x)$ has a fixed-point $y(x) = e^x$.
- We consider functions that are computable transformations of TM descriptions.
- The Fixed-point version of the Recursion Theorem shows that
  - whatever the transformation is
  - there is some TM whose behaviour is unchanged by the transformation!

The Fixed-point version of the Recursion Theorem

Theorem 6.8

Let $t : \Sigma^* \longrightarrow \Sigma^*$. Then, there is a TM F such that $t(\langle F \rangle)$ describes a TM equivalent to F. (t is the transformation and F is the fixed point.)

Proof

- Let F be the following TM:
- $F =$ "On input w
  1. Obtain via the recursion theorem, own description $\langle F \rangle$.
  2. Compute $t(\langle F \rangle)$ to obtain the description of a TM G.
  3. Simulate G on w."
- It is clear that $\langle F \rangle$ and $\langle G \rangle$ describe equivalent TMs: they both compute what G computes with w.

Turing Reducibility

- Reducibility: If A is reducible to B then we can solve A by solving B.
- Mapping Reducibility ($A \leq_m B$) : Use a computable mapping f to transform an instance of A to an instance of B.
- It turns out that Mapping Reducibility is not general enough!
  - Consider A_{TM} and $\overline{A_{TM}}$
  - Clearly the solution to one can be used as a solution to the other, by simply reversing the answer.
  - But $\overline{A_{TM}}$ is not mapping reducible to A_{TM} because A_{TM} is Turing-recognizable while $\overline{A_{TM}}$ is not.
- We need a more general notion of reducibility.

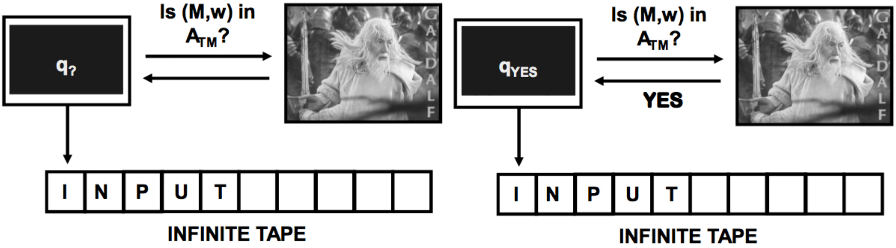
ORACLES

DEFINITION – ORACLE

An oracle for a language B is an external device that is capable of answering the question “Is $w \in B$?”

DEFINITION – ORACLE TURING MACHINE

An oracle TM is a modified TM, M^B, that has the capability of querying an oracle for language B.



Turing Reducibility

Definition

Language A is Turing reducible to language B, written as $A \leq_T B$, if A is decidable relative to B (that is, using an oracle for B)

Theorem

If $A \leq_T B$ and B is decidable, then A is decidable.

Proof

If B is decidable, then replace the oracle with the TM for B.

- Turing reducibility is a generalization of mapping reducibility $A \leq_M B$