university of groningen

Languages and Machines

L11: Decidability (Parts II and III)

Jorge A. Pérez

Bernoulli Institute for Mathematics, Computer Science, and AI
University of Groningen, Groningen, the Netherlands

June 7, 2023

Outline

Last Lecture: The Halting Problem

Universal TMs

Semidecidable and Decidable

Problem Reducibility

Rice's Theorem

Gödel's Incompleteness Theorem

The halting problem for TMs (1/3)

Theorem

The halting problem for TMs is undecidable.

Idea for a proof by contradiction.

1. Assume there is a TM H that solves the halting problem.

A string is accepted by H if

- the input consists of two strings, $R(M)$ and w.
  $R(M)$ is the **representation** of a TM M, and w is the input to M
- the computation of M with input w halts.

Otherwise, H rejects the input.

The halting problem for TMs (1/3)

Theorem

The halting problem for TMs is undecidable.

Idea for a proof by contradiction.

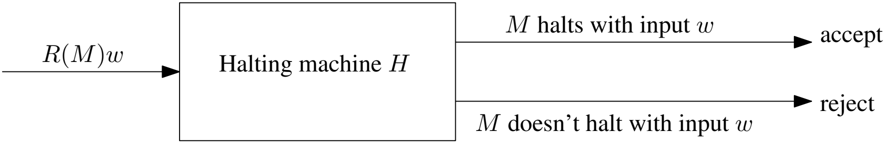
1. Assume there is a TM H that solves the halting problem.

A string is accepted by H if

- the input consists of two strings, $R(M)$ and w.
  $R(M)$ is the **representation** of a TM M, and w is the input to M
- the computation of M with input w halts.

Otherwise, H rejects the input.

Graphically:



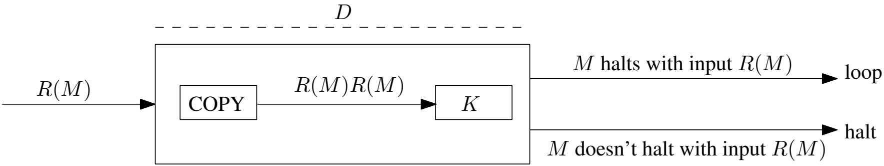
The halting problem for TMs (2/3)

2. Modify H to build another TM, called K: the computations of K are the same as H, but K loops indefinitely whenever H terminates in an accepting state, i.e., whenever M halts on w.

The halting problem for TMs (2/3)

2. Modify H to build another TM, called K: the computations of K are the same as H, but K loops indefinitely whenever H terminates in an accepting state, i.e., whenever M halts on w.

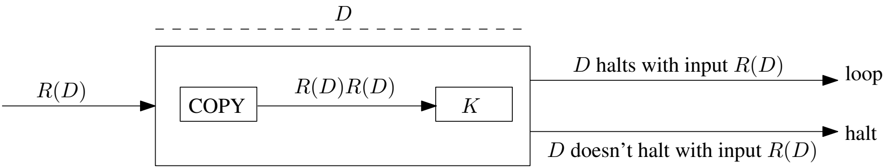
3. Combine K with a "copy machine" to build another TM, called D, with $D(M) = K(M, M)$:



If the call $D(M)$ terminates, then the call $M(M)$ won't terminate

The halting problem for TMs (3/3)

4. The input to D may be the representation of any TM, even D itself. Adapting the diagram in the previous slide:



Thus, $D(D)$ terminates iff $D(D)$ doesn't terminate.
A contradiction, derived from the assumption that there is a machine H that solves the halting problem.

Some Terminology

Recall: A TM is **always terminating** (or **total**) if it halts on (accepts or rejects) all inputs

A language (set of strings) L is

- **recursive**
  if $L = L(M)$ for some always terminating TM M
- **recursively enumerable (r.e.)**
  if $L = L(M)$ for some TM M

Alternatively, let P be a **property** of strings.

- P is decidable
  if the set of all strings having P is recursive: there is a total TM that
  accepts strings that have P and rejects those that don't
- P is semi-decidable
  if the set of strings having P is r.e.: there is a TM that
  accepts x if x has P and *rejects or loops if not*

Outline

Last Lecture: The Halting Problem

Universal TMs

Semidecidable and Decidable

Problem Reducibility

Rice's Theorem

Gödel's Incompleteness Theorem

The Universal TM

- A Universal TM can read representations for TMs and their inputs, and simulate running a TM on its input
- $TM0$: a very simple class of TMs with acceptance by termination and bits as input alphabet
- Given M, we write $R(M)$ to denote its representation
- M terminates on input w iff
  the UTM terminates on input $R(M)w$
- We need to define/establish R and UTM

From M to $R(M)$

- Define a **numbering function** n that maps each state q into a positive integer $n(q)$
- Define numbering functions also for symbols in the tape alphabet and directions L and R
- Mappings may clash, as in $n(q_0) = 1$, $n(0) = 1$, and $n(L) = 1$.

From M to $R(M)$

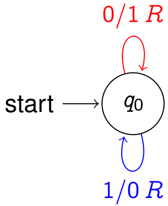
- Define a **numbering function** n that maps each state q into a positive integer $n(q)$
- Define numbering functions also for symbols in the tape alphabet and directions L and R
- Mappings may clash, as in $n(q_0) = 1$, $n(0) = 1$, and $n(L) = 1$.
- Let $1^k = \underbrace{1\,1\,\cdots\,1}_{k \text{ times}}$. A transition $\delta(q, X) = [r, Y, d]$:

$$00\,1^{n(q)}\,0\,1^{n(X)}\,0\,1^{n(r)}\,0\,1^{n(Y)}\,0\,1^{n(d)}$$

- Given M, its representation $R(M)$ corresponds to a sequence of encoded transitions, followed by ‘000’.
- Given an input alphabet of bits, $R(M)w$ corresponds to the regular expression

$$(0(01^+)^5)^*\,000\,(0|1)^*$$

From M to $R(M)$: Example



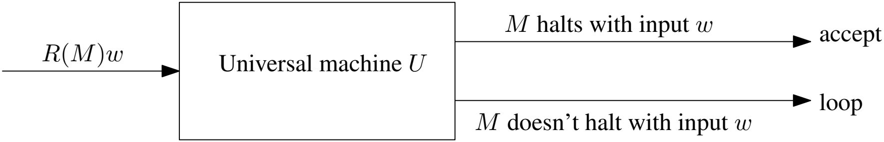
- Encoding states, tape alphabet, directions:

$$n(q_0) = 1 \quad n(0) = 1,\ n(1) = 2,\ n(B) = 3 \quad n(L) = 1,\ n(R) = 2$$

- $R(M)$:

$$\overbrace{00\underbrace{1}_{q_0}0\underbrace{1}_{0}0\underbrace{1}_{q_0}0\underbrace{11}_{1}0\underbrace{11}_{R}}^{\delta(q_0,0)\,=\,[q_0,1,R]}\;\overbrace{00\underbrace{1}_{q_0}0\underbrace{11}_{1}0\underbrace{1}_{q_0}0\underbrace{1}_{0}0\underbrace{11}_{R}}^{\delta(q_0,1)\,=\,[q_0,0,R]}000$$

A Universal TM



Turing's halting language:

$$L_H = \{R(M)w \mid R(M) \text{ represents a TM } M \text{ and } M \text{ halts with input } w\}$$

Theorem

L_H is recursively enumerable.

Proof (Sketch).

It is possible to give a deterministic, three-tape machine U that accepts L_H, simulating the transitions of M—see next. $\square$

Running the UTM

A deterministic, 3-tape TM simulating TMs with a binary alphabet:

1. Check the format of the input; enter into an infinite loop if invalid.
2. Move input to tape 2
3. Write 1 on tape 3 — state 1 should always be the start state
4. Simulate the machine by repeating the following:
   i. Find a transition based on
      - the state (tape 3) and
      - the current symbol (tape 2)
   ii. If no transition is found, terminate
   iii. Otherwise, if a transition is found: change the state (tape 3), change the symbol (tape 2), and move the head (tape 2).

The reader explains how to handle a non-binary alphabet; this requires a fourth tape.

Outline

Decidable $\neq$ Semi-decidable

Recall:

- **Theorem**. If L is recursive (decidable) then L is recursively enumerable (semi-decidable).
- **Theorem**. If L is recursive, then $\overline{L}$ is also recursive.

As already seen, the UTM terminates for input u iff $u \in L_H$, where L_H is Turing's halting language (which the UTM accepts precisely).

- **Theorem**. Language L_H is recursively enumerable.
- **Theorem**. Language $\overline{L_H}$ is not recursively enumerable.
  Similar to the proof of undecidability of the halting problem.
- **Theorem**. Language L_H is not recursive.
  If L_H were recursive, $\overline{L_H}$ would be recursive too (closure properties), and therefore recursively enumerable: contradiction.

Outline

Problem Reducibility

Obtaining new undecidability results from known results

Problem Reducibility

Obtaining new undecidability results from known results

- The halting problem (call it “HALT”) is not decidable
- Suppose we have a problem at hand, called “NEW”.

Problem Reducibility

Obtaining new undecidability results from known results

- The halting problem (call it “HALT”) is not decidable
- Suppose we have a problem at hand, called “NEW”.
- Reducing HALT to NEW means giving a computable (decidable) function so we can use a solution to NEW to solve HALT
- With the reduction, we could decide HALT if NEW was decidable
- But HALT is not decidable, so NEW must also be undecidable

Problem Reducibility

Obtaining new undecidability results from known results

- The halting problem (call it “HALT”) is not decidable
- Suppose we have a problem at hand, called “NEW”.
- Reducing HALT to NEW means giving a computable (decidable) function so we can use a solution to NEW to solve HALT
- With the reduction, we could decide HALT if NEW was decidable
- But HALT is not decidable, so NEW must also be undecidable

More formally: given problems A and B, we can define a relation $A \leq_{red} B$ (“A effectively reducible to B”):

- If $A \leq_{red} B$ and B is decidable then A is also decidable
- If $A \leq_{red} B$ and A is undecidable then B is also undecidable

The previous strategy: establish undecidability of NEW by formalizing HALT $\leq_{red}$ NEW (“HALT is effectively reducible to NEW”)

Acceptance of the empty string (1/2)

The **blank tape problem**: deciding whether a TM halts when a computation is initated with a blank tape (an empty string ϵ)

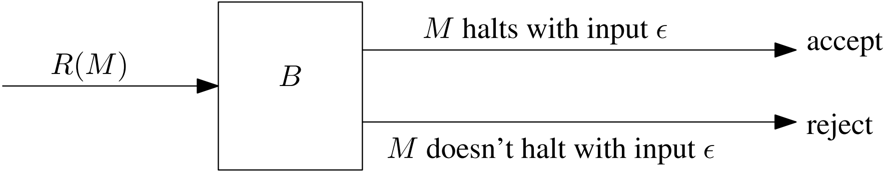
Theorem

The blank tape problem is undecidable.

Idea for a proof by contradiction:

We show that HALT is reducible to the blank tape problem.

1. Assume there's a TM B that solves the blank tape problem:



Acceptance of the empty string (2/2)

2. To reduce HALT to the blank tape problem, we add a preprocessor N to B. The preprocessor N inputs a TM M followed by input w and produces $R(M')$, where M' is a machine that:

   i) Writes w on a blank tape
   ii) Transfers control to the initial state of M
   iii) Runs M

M' halts when run with a blank tape iff M halts with input w.

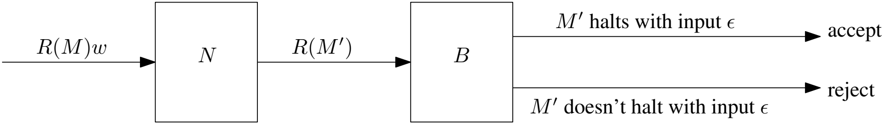
Acceptance of the empty string (2/2)

2. To reduce HALT to the blank tape problem, we add a preprocessor N to B. The preprocessor N inputs a TM M followed by input w and produces $R(M')$, where M' is a machine that:

   i) Writes w on a blank tape
   ii) Transfers control to the initial state of M
   iii) Runs M

M' halts when run with a blank tape iff M halts with input w.

3. Construct the composite machine



This machine solves HALT, which is undecidable.
It then follows that the blank tape problem is undecidable.

Outline

Rice’s Theorem, Intuitively

- Let’s say we are interested in deciding some non trivial property of programs:
  - true of some programs but not of others
  - insensitive to the program’s syntax and to its underlying algorithm
- A non trivial property talks about what a program does, rather than how it does it
- Rice’s theorem says that no such properties can be decided. Hence, undecidability is the rule, rather than the exception

Rice's Theorem

Theorem

Every non trivial property of the recursively enumerable sets is undecidable.

- A property P is a map from r.e. sets to $\top$ (true) or $\bot$ (false)
  Example: the property of emptiness is the map

$$P(A) = \begin{cases} \top & \text{if } A = \emptyset \\ \bot & \text{if } A \neq \emptyset \end{cases}$$

- We represent r.e. sets by TMs that accept them
  Still, we are interested in properties of r.e. sets, not of TMs
- Non trivial properties: there's at least one r.e. set that satisfies the property, and at least one that doesn't. Examples:
  - $L(M)$ is finite / regular / CFL
  - M accepts 1010101 (i.e., $1010101 \in L(M)$)
- Non example: M has at least 42 states

Rice's Theorem: Proof Sketch

- Let P be a non trivial property with $P(\emptyset) = \bot$.
  There must exist an r.e. set A such that $P(A) = \top$.
  Let K be a TM accepting A.
- We reduce the halting problem to the set $\{M \mid P(L(M)) = \top\}$.
  Given $R(M)w$, construct a machine M' that on input y:
  1. Keeps y on a separate track somewhere
  2. Writes w on its tape (w is "hardwired" in the control of M')
  3. Runs M with input w (M is also "hardwired" in the control of M')
  4. If M halts on w, M' runs K on y, and accepts if K accepts
- The simulation in (3) may halt or not. We then have:

$$\begin{aligned} M \text{ doesn't halt on } w \;&\Rightarrow\; L(M') = \emptyset \;\Rightarrow\; P(L(M')) = P(\emptyset) = \bot \\ M \text{ halts on } w \;&\Rightarrow\; L(M') = A \;\Rightarrow\; P(L(M')) = P(A) = \top \end{aligned}$$

- This reduces the halting problem to set $\{M \mid P(L(M)) = \top\}$.
  Hence, it is undecidable whether $L(M)$ satisfies P.

Rice's Theorem - In the Reader (1/3)

Theorem

Let P be a class of languages over $\mathbb{B}$.
Let L_1 and L_2 be semi-decidable languages over $\mathbb{B}$ with $L_1 \in P$ and $L_2 \notin P$ and $L_1 \subseteq L_2$.
Then the language L_P is not semi-decidable.

Rice's Theorem - In the Reader (2/3)

- From 'not decidable' to 'not semi-decidable' properties
  $L_1 \in P$ and $L_2 \notin P$ mean 'non trivial property';
  together with $L_1 \subseteq L_2$, they mean 'non monotone property'
- Properties not as mappings but as classes of languages, i.e., sets of TMs M such that $P(L(M)) = \top$
- Proof contradicts the following theorem: The complement of L_H (semi-decidable) is not semi-decidable (Theorem 6.4)
- Assumption: there's a TM M_P that accepts L_P.
  Use M_P to construct a TM K such that $L(K) = \overline{L_H}$.
  - Given input $u = R(M)w$, K executes M_P with input $R(M')$
    K accepts u iff M_P accepts $R(M')$
  - What is M'? M' runs machines M_1 and M_2 (accepters for L_1 and L_2): $v \in L(M')$ iff $v \in L(M_1)$ OR ($v \in L(M_2)$ followed by $w \in L(M)$)
  - We have: (i) $w \in L(M) \Rightarrow L_2 \notin P$ and (ii) $w \notin L(M) \Rightarrow L_1 \in P$ and therefore that $w \notin L(M)$ iff $L(M') \in P$
    Assumption $L_1 \subseteq L_2$ is used here.

Rice's Theorem - In the Reader (3/3)

Theorem

Let P be a class of languages over $\mathbb{B}$. Let L_1 and L_2 be semi-decidable languages over $\mathbb{B}$ with $L_1 \in P$ and $L_2 \notin P$ and $L_1 \subseteq L_2$. Then the language L_P is not semi-decidable.

Proof (Sketch).

Recall $L_P = \{R(M_i) \mid M_i \in TM0 : L(M_i) \in P\}$. Based on the previous constructions (K and $M\,/\,M'$), we have the following:

$$
\begin{aligned}
R(M)w \in L(K) &\equiv R(M') \in L(M_P) && [K \text{ executes } M_P \text{ on } R(M')] \\
&\equiv R(M') \in L_P && [\text{assumption on } M_P] \\
&\equiv L(M') \in P && [\text{definition of } L_P] \\
&\equiv w \notin L(M') && [w \notin L(M) \text{ iff } \in L(M') \in P]
\end{aligned}
$$

Therefore, $L(K) = \overline{L_H}$, and so $\overline{L_H}$ is semi-decidable.
This contradicts Thm 6.4: hence, L_P is not semi-decidable. □

Try it yourselves

Theorem

Let L be the language: $\{R(M) \mid L(M)$ is not regular$\}$. Then L is not semi-decidable (i.e., recursively enumerable).

You may reuse the previous theorem:

Theorem

Let P be a class of languages over $\mathbb{B}$. Let L_1 and L_2 be semi-decidable languages over $\mathbb{B}$ with $L_1 \in P$ and $L_2 \notin P$ and $L_1 \subseteq L_2$. Then the language L_P is not semi-decidable.

Try it yourselves

Theorem
Let L be the language: $\{R(M) \mid L(M)$ is not regular$\}$.
Then L is not semi-decidable (i.e., recursively enumerable).

Hints:

- The property P in this case concerns "non regularity"
- We can use the previous theorem by finding L_1 and L_2 such that:
  - $L_1 \in P$ (i.e., a language that is not regular);
  - $L_2 \notin P$ (i.e., a language that is regular);
  - $L_1 \subseteq L_2$.

Outline

Gödel's Incompleteness Theorem

- A proof system is sound if all theorems are true, i.e., it is not possible to prove a false sentence
- A proof system is complete if all true sentences are theorems of the system

Gödel's Result: No reasonable formal system for number theory is complete—can prove all true sentences

In the following:

- The language of number theory
- Peano arithmetic (a proof system for number theory)
- Sketch of Gödel's proof

The language of number theory

A language for expressing properties of the naturals $\mathbb{N} = \{0, 1, \ldots\}$.

- variables x, y, z, $\ldots$ ranging over $\mathbb{N}$
- operator symbols $+$ and $\cdot$, and
- constant symbols 0 and 1 (identities for $+$ and $\cdot$)
- relation symbol $=$ (symbols such as $<$, $\leq$, $>$, $\geq$ are definable)
- quantifiers $\forall$, $\exists$, and propositional operators $\vee$, $\wedge$, $\neg$, $\Rightarrow$, etc.
- parentheses

The language can define concepts such as "y divides x", "x is odd", and bit-manipulation formulas (cf. TM encodings)

A formula without free (unquantified) variables is called a sentence. Sentences have a well-defined truth value.

Th($\mathbb{N}$): the set of all true sentences. The decision problem for number theory is to decide whether a sentence is in Th($\mathbb{N}$).

Peano Arithmetic (PA)

A proof system for number theory.

Consists of axioms (basic assumptions) + rules of inference (applied mechanically to derive theorems from the axioms)
Write $\varphi(x)$ to denote a formula with free variable x

- Axioms from first-order logic (propositional formulas, quantifiers, equality) but also from number theory (successor, identities, induction axiom)
- Inference rules:

$$\frac{\varphi \quad \varphi \Rightarrow \psi}{\psi} \qquad \frac{\varphi}{\forall x\, \varphi}$$

A proof of φ_n is a sequence $\varphi_0, \ldots, \varphi_n$ of formulas s.t. each φ_i either is an axiom or follows from earlier formulas by an inference rule. A sentence is a theorem if it has a proof.

PA is sound: the set of theorems of PA is a subset of Th($\mathbb{N}$).
Gödel's remarkable result is that PA is not complete.

Incompleteness Theorem

Gödel proved incompleteness by constructing a sentence of number theory φ that asserts its own unprovability:

$$\varphi \text{ is true} \iff \varphi \text{ is not provable}$$

The construction of φ is interesting, as it captures self-reference as present in TMs and programming languages.

Incompleteness Theorem - Proof Sketch

A proof approach due to Turing: In a proof system such as PA one can show that

1. The set of theorems (provable sentences) is r.e., but
2. The set of true sentences $\mathrm{Th}(\mathbb{N})$ is not r.e.

Therefore, the two sets cannot be equal, and the proof system cannot be complete.

It is relatively easy to show (1), but proving (2) is much harder.

Th($\mathbb{N}$) is not r.e - Proof Sketch

A reduction from $\overline{L_H}$ to Th($\mathbb{N}$).

- Given $R(M)w$, we produce a sentence γ in the language of number theory such that $R(M)w \in \overline{L_H} \iff \gamma \in$ Th($\mathbb{N}$). Thus, M doesn't halt on $w \iff \gamma$ is true.
- Intuitively, γ uses number theory to say "M doesn't halt on w".
- Construct a formula VALCOMP$_{M,w}(y)$ that says that y represents a valid computation history of M on input w.
- Hence, VALCOMP$_{M,w}(y)$ says that y represents a sequence of configurations of M, written $\alpha_0, \ldots, \alpha_N$, such that α_0 is the start configuration (with w) and α_N is a halt configuration.
- The desired formula is then $\gamma = \neg\exists y$ VALCOMP$_{M,w}(y)$.

Taking Stock

This lecture:

- A Universal Turing machine
- Undecidability results
- Acceptance of the empty string (the blank tape problem)
- Rice's Theorem
- Incompleteness of arithmetic

Next Lecture

- Unrestricted and Context-Sensitive Grammars/Languages
- Course Evaluation