Fundamentele Informatica 3

voorjaar 2012

`http://www.liacs.nl/home/rvvliet/fi3/`

Rudy van Vliet
kamer 124 Snellius, tel. 071-527 5777
rvvliet(at)liacs.nl

college 13, dinsdag 1 mei 2012

9. Undecidable Problems
9.1. A Language That Can't Be Accepted, and a Problem That Can't Be Decided
9.2. Reductions and the Halting Problem

Definition 8.1. Accepting a Language and Deciding a Language

A Turing machine T with input alphabet Σ accepts a language $L \subseteq \Sigma^*$,
if $L(T) = L$.

T *decides* L,
if T computes the characteristic function $\chi_L : \Sigma^* \to \{0,1\}$

A language L is *recursively enumerable*,
if there is a TM that accepts L,

and L is *recursive*,
if there is a TM that decides L.

Example 8.30. The Set of Turing Machines Is Countable

Let $\mathcal{T}$ be set of Turing machines
There is injective function $e : \mathcal{T} \to \{0,1\}^*$
(e is encoding function)

Hence, set of recursively enumerable languages is countable

Example 8.31. The Set $2^{\mathbb{N}}$ Is Uncountable

Hence, because $\mathbb{N}$ and $\{0,1\}^*$ are the same size,
there are uncountably many languages over $\{0,1\}$

Example 8.31. The Set $2^{\mathbb{N}}$ Is Uncountable (continued)

No list of subsets of $\mathbb{N}$ is complete,
i.e., every list $A_0, A_1, A_2, \ldots$ of subsets of $\mathbb{N}$ leaves out at least one.

Take

$$A = \{i \in \mathbb{N} \mid i \notin A_i\}$$

	0	1	2	3	4	5	6	7	8	9	...
A_0	1	0	1	0	0	1	0	0	0	1	...
A_1	0	1	1	1	0	0	0	0	1	0	...
A_2	1	0	0	1	0	0	1	0	0	0	...
A_3	0	0	0	0	0	0	0	0	0	0	...
A_4	0	0	0	0	1	0	0	0	0	0	...
A_5	0	0	1	1	0	1	0	1	0	0	...
A_6	0	0	0	0	0	0	0	0	1	0	...
A_7	1	1	1	1	1	1	1	1	1	1	...
A_8	0	1	0	1	0	1	0	1	0	1	...
A_9	0	0	0	0	0	0	0	0	0	0	...
...											

	0	1	2	3	4	5	6	7	8	9	...
A	0	0	1	1	0	0	1	0	1	1	...
A_0	1	0	1	0	0	1	0	0	0	1	...
A_1	0	1	1	1	0	0	0	0	1	0	...
A_2	1	0	0	1	0	0	1	0	0	0	...
A_3	0	0	0	0	0	0	0	0	0	0	...
A_4	0	0	0	0	1	0	0	0	0	0	...
A_5	0	0	1	1	0	1	0	1	0	0	...
A_6	0	0	0	0	0	0	0	0	1	0	...
A_7	1	1	1	1	1	1	1	1	1	1	...
A_8	0	1	0	1	0	1	0	1	0	1	...
A_9	0	0	0	0	0	0	0	0	0	0	...
...											

Hence, there are uncountable many subsets of $\mathbb{N}$.

Theorem 8.32. Not all languages are recursively enumerable. In fact, the set of languages over $\{0,1\}$ that are not recursively enumerable is uncountable.

Proof...

(including Exercise 8.38)

9. Undecidable Problems

9.1. A Language That Can't Be Accepted, and a Problem That Can't Be Decided

From lecture 9:

Crucial features of any encoding function e:
(of a Turing machine)

1. It should be possible to decide algorithmically, for any string $w \in \{0,1\}^*$, whether w is a legitimate value of e.
2. A string w should represent at most one Turing machine, or at most one string z.
3. If $w = e(T)$ or $w = e(z)$, there should be an algorithm for *decoding* w.

Definition 9.1. The Languages *NSA* and *SA*

Let

$$\textit{NSA} = \{e(T) \mid T \text{ is a TM, and } e(T) \notin L(T)\}$$
$$\textit{SA} = \{e(T) \mid T \text{ is a TM, and } e(T) \in L(T)\}$$

(*NSA* and *SA* are for "non-self-accepting" and "self-accepting.")

	$e(T_0)$	$e(T_1)$	$e(T_2)$	$e(T_3)$	$e(T_4)$	$e(T_5)$	$e(T_6)$	$e(T_7)$	$e(T_8)$	$e(T_9)$
$L(T_0)$	1	0	1	0	0	1	0	0	0	1
$L(T_1)$	0	1	1	1	0	0	0	0	1	0
$L(T_2)$	1	0	0	1	0	0	1	0	0	0
$L(T_3)$	0	0	0	0	0	0	0	0	0	0
$L(T_4)$	0	0	0	0	1	0	0	0	0	0
$L(T_5)$	0	0	1	1	0	1	0	1	0	0
$L(T_6)$	0	0	0	0	0	0	0	0	1	0
$L(T_7)$	1	1	1	1	1	1	1	1	1	1
$L(T_8)$	0	1	0	1	0	1	0	1	0	1
$L(T_9)$	0	0	0	0	0	0	0	0	0	0

...

	$e(T_0)$	$e(T_1)$	$e(T_2)$	$e(T_3)$	$e(T_4)$	$e(T_5)$	$e(T_6)$	$e(T_7)$	$e(T_8)$	$e(T_9)$
NSA	0	0	1	1	0	0	1	0	1	1
$L(T_0)$	1	0	1	0	0	1	0	0	0	1
$L(T_1)$	0	1	1	1	0	0	0	0	1	0
$L(T_2)$	1	0	0	1	0	0	1	0	0	0
$L(T_3)$	0	0	0	0	0	0	0	0	0	0
$L(T_4)$	0	0	0	0	1	0	0	0	0	0
$L(T_5)$	0	0	1	1	0	1	0	1	0	0
$L(T_6)$	0	0	0	0	0	0	0	0	1	0
$L(T_7)$	1	1	1	1	1	1	1	1	1	1
$L(T_8)$	0	1	0	1	0	1	0	1	0	1
$L(T_9)$	0	0	0	0	0	0	0	0	0	0

...

Hence, *NSA* is not recursively enumerable.

Theorem 9.2. The language *NSA* is not recursively enumerable. The language *SA* is recursively enumerable but not recursive.

Proof...

Decision problem: problem for which the answer is 'yes' or 'no':

Given ..., is it true that ...?

yes-instances of a decision problem:
instances for which the answer is 'yes'

no-instances of a decision problem:
instances for which the answer is 'no'

Self-Accepting: Given a TM T, does T accept the string $e(T)$?

Three languages corresponding to this problem:
1. *SA*: strings representing yes-instances
2. *NSA*: strings representing no-instances
3. ...

Self-Accepting: Given a TM T, does T accept the string $e(T)$?

Three languages corresponding to this problem:
1. *SA*: strings representing yes-instances
2. *NSA*: strings representing no-instances
3. E': strings not representing instances

For general decision problem P, let e be *reasonable* encoding of instances I as strings $e(I)$ over alphabet Σ.
1. e is injective
2. string $e(I)$ can be decoded
3. there is algorithm to decide if string over Σ is encoding $e(I)$

From lecture 9:

Crucial features of any encoding function e:
(of a Turing machine)

1. It should be possible to decide algorithmically, for any string $w \in \{0,1\}^*$, whether w is a legitimate value of e.
2. A string w should represent at most one Turing machine, or at most one string z.
3. If $w = e(T)$ or $w = e(z)$, there should be an algorithm for *decoding* w.

For general decision problem P and reasonable encoding e,

$$\begin{aligned} Y(P) &= \{e(I) \mid I \text{ is yes-instance of } P\} \\ N(P) &= \{e(I) \mid I \text{ is no-instance of } P\} \\ E(P) &= Y(P) \cup N(P) \end{aligned}$$

$E(P)$ must be recursive

Definition 9.3. Decidable Problems

If P is a decision problem, and e is a reasonable encoding of instances of P over the alphabet Σ, we say that P is *decidable* if $Y(P) = \{e(I) \mid I \text{ is a yes-instance of } P\}$ is a recursive language.

Theorem 9.4. The decision problem *Self-Accepting* is undecidable.

Proof...

For every decision problem, there is *complementary* problem P', obtained by changing 'true' to 'false' in statement.

Non-Self-Accepting:
Given a TM T, does T fail to accept $e(T)$?

Theorem 9.5. For every decision problem P, P is decidable if and only if the complementary problem P' is decidable.

Proof...

9.2. Reductions and the Halting Problem

Definition 9.6. Reducing One Decision Problem to Another, and Reducing One Language to Another

Suppose P_1 and P_2 are decision problems. We say P_1 is reducible to P_2 ($P_1 \leq P_2$)
• if there is an algorithm
• that finds, for an arbitrary instance I of P_1, an instance $F(I)$ of P_2,
• such that for every I the answers for the two instances are the same, or I is a yes-instance of P_1 if and only if $F(I)$ is a yes-instance of P_2.

Definition 9.6. Reducing One Decision Problem to Another, and Reducing One Language to Another (continued)

If L_1 and L_2 are languages over alphabets Σ_1 and Σ_2, respectively, we say L_1 is reducible to L_2 ($L_1 \leq L_2$)
• if there is a Turing-computable function
• $f : \Sigma_1^* \to \Sigma_2^*$
• such that for every $x \in \Sigma_1^*$,

$$x \in L_1 \text{ if and only if } f(x) \in L_2$$

Less / more formal definitions.

Theorem 9.7. Suppose $L_1 \subseteq \Sigma_1^*$, $L_2 \subseteq \Sigma_2^*$, and $L_1 \leq L_2$. If L_2 is recursive, then L_1 is recursive.

Suppose P_1 and P_2 are decision problems, and $P_1 \leq P_2$. If P_2 is decidable, then P_1 is decidable.

Proof. . .

In context of decidability: decision problem $P \approx$ language $Y(P)$

Question

“is instance I of P a yes-instance ?”

is essentially the same as

“does string x represent yes-instance of P ?”,

i.e.,

“is string $x \in Y(P)$?”

Therefore, $P_1 \leq P_2$, if and only if $Y(P_1) \leq Y(P_2)$.

Two more decision problems:

Accepts: Given a TM T and a string w, is $w \in L(T)$?

Halts: Given a TM T and a string w, does T halt on input w ?

Theorem 9.8 Both *Accepts* and *Halts* are undecidable.

Proof.

1. Prove that *Self-Accepting* $\leq$ *Accepts* . . .

Theorem 9.8 Both *Accepts* and *Halts* are undecidable.

Proof.

1. Prove that *Self-Accepting* $\leq$ *Accepts* . . .

2. Prove that *Accepts* $\leq$ *Halts* . . .

Application:

```
n = 4;
while (n is the sum of two primes)
  n = n+2;
```

This program loops forever, if and only if Goldbach's conjecture is true.