Fundamentele Informatica 3

voorjaar 2016

`http://www.liacs.leidenuniv.nl/~vlietrvan1/fi3/`

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

college 9, 4+5 april 2016

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

A slide from lecture 8:

From Fundamentele Informatica 1:

Definition 8.24.
Countably Infinite and Countable Sets

A set A is *countably infinite* (the same size as $\mathbb{N}$) if there is a bijection $f : \mathbb{N} \rightarrow A$, or a list $a_0, a_1, \ldots$ of elements of A such that every element of A appears exactly once in the list.

A is *countable* if A is either finite or countably infinite.

A slide from lecture 8:

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 $\{0,1\}$

A slide from lecture 8:

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

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

A slide from lecture 8:

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

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

$$\begin{array}{lcl}
A_0 & = & \{0, 2, 5, 9, \ldots\} \\
A_1 & = & \{1, 2, 3, 8, 12, \ldots\} \\
A_2 & = & \{0, 3, 6\} \\
A_3 & = & \emptyset \\
A_4 & = & \{4\} \\
A_5 & = & \{2, 3, 5, 7, 11, \ldots\} \\
A_6 & = & \{8, 16, 24, \ldots\} \\
A_7 & = & \mathbb{N} \\
A_8 & = & \{1, 3, 5, 7, 9, \ldots\} \\
A_9 & = & \{n \in \mathbb{N} \mid n > 12\} \\
\ldots & &
\end{array}$$

A slide from lecture 8:

	0	1	2	3	4	5	6	7	8	9	...
$A_0 = \{0, 2, 5, 9, \ldots\}$	1	0	1	0	0	1	0	0	0	1	...
$A_1 = \{1, 2, 3, 8, 12, \ldots\}$	0	1	1	1	0	0	0	0	1	0	...
$A_2 = \{0, 3, 6\}$	1	0	0	1	0	0	1	0	0	0	...
$A_3 = \emptyset$	0	0	0	0	0	0	0	0	0	0	...
$A_4 = \{4\}$	0	0	0	0	1	0	0	0	0	0	...
$A_5 = \{2, 3, 5, 7, 11, \ldots\}$	0	0	1	1	0	1	0	1	0	0	...
$A_6 = \{8, 16, 24, \ldots\}$	0	0	0	0	0	0	0	0	1	0	...
$A_7 = \mathbb{N}$	1	1	1	1	1	1	1	1	1	1	...
$A_8 = \{1, 3, 5, 7, 9, \ldots\}$	0	1	0	1	0	1	0	1	0	1	...
$A_9 = \{n \in \mathbb{N} \mid n > 12\}$	0	0	0	0	0	0	0	0	0	0	...
...						...					

A slide from lecture 8:

	0	1	2	3	4	5	6	7	8	9	...
$A_0 = \{0, 2, 5, 9, \ldots\}$	1	0	1	0	0	1	0	0	0	1	...
$A_1 = \{1, 2, 3, 8, 12, \ldots\}$	0	1	1	1	0	0	0	0	1	0	...
$A_2 = \{0, 3, 6\}$	1	0	0	1	0	0	1	0	0	0	...
$A_3 = \emptyset$	0	0	0	0	0	0	0	0	0	0	...
$A_4 = \{4\}$	0	0	0	0	1	0	0	0	0	0	...
$A_5 = \{2, 3, 5, 7, 11, \ldots\}$	0	0	1	1	0	1	0	1	0	0	...
$A_6 = \{8, 16, 24, \ldots\}$	0	0	0	0	0	0	0	0	1	0	...
$A_7 = \mathbb{N}$	1	1	1	1	1	1	1	1	1	1	...
$A_8 = \{1, 3, 5, 7, 9, \ldots\}$	0	1	0	1	0	1	0	1	0	1	...
$A_9 = \{n \in \mathbb{N} \mid n > 12\}$	0	0	0	0	0	0	0	0	0	0	...
...						...					
$A = \{2, 3, 6, 8, 9, \ldots\}$	0	0	1	1	0	0	1	0	1	1	...

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

Set-up of Example 8.31:

1. Start with list of all subsets of $\mathbb{N}$: $A_0, A_1, A_2, \ldots$,
   each one associated with specific element of $\mathbb{N}$ (namely i)

2. Define another subset A by:
   $i \in A \iff i \notin A_i$

3. Conclusion: for all i, $A \neq A_i$
   Hence, contradiction
   Hence, there are uncountably many subsets of $\mathbb{N}$

Exercise 8.45.

The two parts of this exercise show that for every set S (not necessarily countable), 2^S is larger than S.

a. For every S, describe a simple bijection from S to a subset of 2^S.

b. Show that for every S, there is no bijection from S to 2^S. (You can copy the proof in Example 8.31, as long as you avoid trying to list the elements of S or making any reference to the countability of S.)

9. Undecidable Problems

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

A slide from lecture 5:

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.

Set-up of Example 8.31:

1. Start with list of all subsets of $\mathbb{N}$: $A_0, A_1, A_2, \ldots$,
   each one associated with specific element of $\mathbb{N}$ (namely i)

2. Define another subset A by:
   $i \in A \iff i \notin A_i$

3. Conclusion: for all i, $A \neq A_i$
   Hence, contradiction
   Hence, there are uncountably many subsets of $\mathbb{N}$

Set-up of constructing language that is not RE:

1. Start with list of all RE languages over $\{0,1\}$ (which are subsets of $\{0,1\}^*$): $L(T_0), L(T_1), L(T_2), \ldots$ each one associated with specific element of $\{0,1\}^*$

2. Define another language L by: $x \in L \iff x \notin$ (language that x is associated with)

3. Conclusion: for all i, $L \neq L(T_i)$ Hence, L is not RE

	$e(T_0)$	$e(T_1)$	$e(T_2)$	$e(T_3)$	$e(T_4)$	$e(T_5)$	$e(T_6)$	$e(T_7)$	$e(T_8)$	$e(T_9)$
$L(T_0)$	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
$\dots$						$\dots$				

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

Hence, *NSA* is not recursively enumerable.

A slide from lecture 4:

Some Crucial features of any encoding function e:

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 with a given input alphabet Σ, or at most one string z.
3. If $w = e(T)$ or $w = e(z)$, there should be an algorithm for *decoding* w.

Set-up of constructing language *NSA* that is not RE:

1. Start with list of all RE languages over $\{0, 1\}$
   (which are subsets of $\{0, 1\}^*$): $L(T_0), L(T_1), L(T_2), \ldots$
   each one associated with specific element of $\{0, 1\}^*$
   (namely $e(T_i)$)

2. Define another language *NSA* by:
   $e(T_i) \in \textit{NSA} \iff e(T_i) \notin L(T_i)$

3. Conclusion: for all i, $\textit{NSA} \neq L(T_i)$
   Hence, *NSA* is not RE

Set-up of constructing language *NSA* that is not RE:

1. Start with collection of all RE languages over $\{0,1\}$ (which are subsets of $\{0,1\}^*$): $\{L(T) \mid \text{ TM } T\}$ each one associated with specific element of $\{0,1\}^*$ (namely $e(T)$)

2. Define another language *NSA* by:
   $e(T) \in \mathit{NSA} \iff e(T) \notin L(T)$

3. Conclusion: for all TM T, $\mathit{NSA} \neq L(T)$
   Hence, *NSA* is not RE

Set-up of constructing language L that is not RE:

1. Start with list of all RE languages over $\{0,1\}$ (which are subsets of $\{0,1\}^*$): $L(T_0), L(T_1), L(T_2), \ldots$ each one associated with specific element of $\{0,1\}^*$ (namely x_i)

2. Define another language L by:
   $x_i \in L \iff x_i \notin L(T_i)$

3. Conclusion: for all i, $L \neq L(T_i)$
   Hence, L is not RE

Every infinite list $x_0, x_1, x_2, \ldots$ of different elements of $\{0,1\}^*$ yields language L that is not RE

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

Let

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

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

A slide from lecture 4:

Some Crucial features of any encoding function e:

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 with a given input alphabet Σ, or at most one string z.
3. If $w = e(T)$ or $w = e(z)$, there should be an algorithm for *decoding* w.

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

Proof. . .

Exercise 9.2.

Describe how a universal Turing machine could be used in the proof that SA is recursively enumerable.

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’

Decision problems

Given an undirected graph $G = (V, E)$,
does G contain a Hamiltonian path?

Given a list of integers $x_1, x_2, \ldots, x_n$,
is the list sorted?

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,
an encoding e of instances I as strings $e(I)$ over alphabet Σ
is called *reasonable*, if

1. there is algorithm to decide if string over Σ is encoding $e(I)$
2. e is injective
3. string $e(I)$ can be decoded

A slide from lecture 4:

Some Crucial features of any encoding function e:

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 with a given input alphabet Σ, 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. . .

SA vs. *NSA*

Self-Accepting vs. *Non-Self-Accepting*

9.2. Reductions and the Halting Problem

(Informal) Examples of reductions

1. Recursive algorithms

2. Given NFA M and string x, is $x \in L(M)$?

3. Given FAs M_1 and M_2, is $L(M_1) \subseteq L(M_2)$?

Theorem 2.15.
Suppose $M_1 = (Q_1, \Sigma, q_1, A_1, \delta_1)$ and $M_2 = (Q_2, \Sigma, q_2, A_2, \delta_2)$ are finite automata accepting L_1 and L_2, respectively.
Let M be the FA $(Q, \Sigma, q_0, A, \delta)$, where

$Q = Q_1 \times Q_2$

$q_0 = (q_1, q_2)$

and the transition function δ is defined by the formula

$\delta((p,q),\sigma) = (\delta_1(p,\sigma), \delta_2(q,\sigma))$

for every $p \in Q_1$, every $q \in Q_2$, and every $\sigma \in \Sigma$.

Then

1. If $A = \{(p,q) |\ p \in A_1$ or $q \in A_2\}$,
   M accepts the language $L_1 \cup L_2$.
2. If $A = \{(p,q) |\ p \in A_1$ and $q \in A_2\}$,
   M accepts the language $L_1 \cap L_2$.
3. If $A = \{(p,q) |\ p \in A_1$ and $q \notin A_2\}$,
   M accepts the language $L_1 - L_2$.

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.