Fundamentele Informatica 3

voorjaar 2019

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

Rudy van Vliet
kamer 140 Snellius, tel. 071-527 2876
rvvliet(at)liacs(dot)nl

college 6, 18 maart 2019

8. Recursively Enumerable Languages
8.1. Recursively Enumerable and Recursive
8.5. Not Every Language is Recursively Enumerable
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

Huiswerkopgave,
inleverdatum 25 maart 2019, 11:05 uur

A slide from lecture 4:

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.

A slide from lecture 4:

Theorem 8.2.
Every recursive language is recursively enumerable.

Proof. . .

A slide from lecture 4:

Theorem 8.4. If L_1 and L_2 are both recursively enumerable languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable.

Proof. . .

For intersection: not just $T_1 \to T_2$

An exercise from exercise class 4:

Exercise 8.1.

Show that if L_1 and L_2 are recursive languages, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also.

Theorem 8.5. If L_1 and L_2 are both recursive languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursive.

Proof. Exercise 8.1.

Theorem 8.6. If L is a recursive language over Σ, then its complement L' is also recursive.

Proof. . .

Theorem 8.7. If L is a recursively enumerable language, and its complement L' is also recursively enumerable, then L is recursive (and therefore, by Theorem 8.6, L' is recursive).

Proof. . .

Corollary.
Let L be a recursively enumerable language.
Then

L' is recursively enumerable,
if and only
if L is recursive.

Corollary.
There exist languages that are not recursively enumerable,
if and only if
there exist languages that are not recursive.

8.5. Not Every Language is Recursively Enumerable

reg. languages	FA	reg. grammar	reg. expression
determ. cf. languages	DPDA		
cf. languages	PDA	cf. grammar	
cs. languages	LBA	cs. grammar	
re. languages	TM	unrestr. grammar	

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 $f : \mathbb{N} \to A$, 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.

uncountable: not countable

Example 8.29. Languages Are Countable Sets

$$L \subseteq \Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$$

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.

A slide from lecture 4

Assumptions:

1. Names of the states are irrelevant.

2. Tape alphabet Γ of every Turing machine T is subset of infinite set $\mathcal{S} = \{a_1, a_2, a_3, \ldots\}$, where $a_1 = \Delta$.

A slide from lecture 4

Definition 7.33. An Encoding Function

Assign numbers to each state:
$n(h_a) = 1$, $n(h_r) = 2$, $n(q_0) = 3$, $n(q) \geq 4$ for other $q \in Q$.

Assign numbers to each tape symbol:
$n(a_i) = i$.

Assign numbers to each tape head direction:
$n(R) = 1$, $n(L) = 2$, $n(S) = 3$.

A slide from lecture 4

Definition 7.33. An Encoding Function (continued)

For each move m of T of the form $\delta(p, \sigma) = (q, \tau, D)$

$$e(m) = 1^{n(p)}01^{n(\sigma)}01^{n(q)}01^{n(\tau)}01^{n(D)}0$$

We list the moves of T in some order as $m_1, m_2, \ldots, m_k$, and we define

$$e(T) = e(m_1)0e(m_2)0\ldots0e(m_k)0$$

If $z = z_1z_2\ldots z_j$ is a string, where each $z_i \in \mathcal{S}$,

$$e(z) = 01^{n(z_1)}01^{n(z_2)}0\ldots01^{n(z_j)}0$$

Example 8.30. The Set of Turing Machines Is Countable

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

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.

(Not) Recursively enumerable

vs.

(Not) Countable

A slide from lecture 4:

Theorem 8.4. If L_1 and L_2 are both recursively enumerable languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable.

Proof. . .

Exercise 8.3.

Is the following statement true or false?

If $L_1, L_2, \ldots$ are any recursively enumerable subsets of Σ^*, then $\cup_{i=1}^{\infty} L_i$ is recursively enumerable.

Give reasons for your answer.

9. Undecidable Problems

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

A slide from lecture 4

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.

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

	$e(T_0)$	$e(T_1)$	$e(T_2)$	$e(T_3)$	$e(T_4)$	$e(T_5)$	$e(T_6)$	$e(T_7)$	$e(T_8)$	$e(T_9)$
$L(T_0)$	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
$\ldots$						$\ldots$				
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 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:
   $e(T_i) \in \mathit{NSA} \iff e(T_i) \notin L(T_i)$

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

Set-up of constructing language that is not RE:

1. Start with list of 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

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

1. Start with list of 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

	Λ	0	1	00	01	10	11	000	001	010	...
$L(T_0)$	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	...
...							...				
newL	0	0	1	1	0	0	1	0	1	1	...

Hence, newL is not recursively enumerable.

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

1. Start with collection of 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:
   $e(T) \in \textit{NSA} \iff e(T) \notin L(T)$

3. Conclusion: for all TM T, $\textit{NSA} \neq L(T)$
   Hence, *NSA* 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.

Given a TM T, does T accept the string $e(T)$?

Decision problem: problem for which the answer is ‘yes’ or ‘no’:

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

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 $e(T)$?

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,
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 . . .

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.

. . .

Theorem 9.7.

. . .

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.

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.

Exercise 9.5.

Fermat's last theorem, until recently one of the most famous unproved statements in mathematics, asserts that there are no integer solutions (x, y, z, n) to the equation $x^n + y^n = z^n$ satisfying $x, y > 0$ and $n > 2$.

Ignoring the fact that the theorem has now been proved, explain how a solution to the halting problem would allow you to determine the truth or falsity of the statement.

Theorem 9.7.

. . .

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.

Order $P_1 \leq P_2$

Proof. . .

Exercise 9.1.

Show that the relation $\leq$ on the set of decision problems is reflexive and transitive.

Give an example to show that it is not symmetric.

NSE 2019

Jouw mening is belangrijk!

Wat?

- De *N*ationale *S*tudenten *E*nquête

Waarom?

- Omdat je graag je mening wilt geven & wilt meehelpen je opleiding te verbeteren
- Omdat bij 25% respons studenten koeken krijgen
- Omdat er per ingevulde enquête 25 cent wordt gedoneerd aan stichting vluchteling-student (UAF)
- Omdat de studievereniging met de hoogste respons een gratis sportactiviteit mag organiseren

Hoe?

- Via de persoonlijke link in de uitnodigingsmail
- Link kwijt? Vul je uMail-ibox e-mailadres in op www.nse.nl

Gemakkelijk via je telefoon!