Computability

voorjaar 2023

`https://liacs.leidenuniv.nl/~vlietrvan1/computability/`

college 7, 23 maart 2023

8. Recursively Enumerable Languages
8.5. Not Every Language is Recursively Enumerable
9. Undecidable Problems
9.2. Reductions and the Halting Problem
9.3. More Decision Problems Involving Turing Machines

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 Foundations of Computer Science:

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.

9.2. Reductions and the Halting Problem

A slide from lecture 6:

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 6:

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

A slide from lecture 6:

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 $Y(P) = \{e(I) \mid \ I$ is a yes-instance of $P\}$ is a recursive language.

A slide from lecture 6:

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.

. . .

A slide from lecture 6:

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.

A slide from lecture 6:

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 ?

9.3. More Decision Problems Involving Turing Machines

Accepts: Given a TM T and a string x, is $x \in L(T)$?
Instances are . . .

Halts: Given a TM T and a string x, does T halt on input x ?
Instances are . . .

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

Accepts: Given a TM T and a string x, is $x \in L(T)$?
Instances are . . .

Halts: Given a TM T and a string x, does T halt on input x ?
Instances are . . .

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

Now fix a TM T:
T-*Accepts*: Given a string x, does T accept x ?
Instances are . . .
Decidable or undecidable ? (cf. **Exercise 9.7.**)

Theorem 9.9. The following five decision problems are undecidable.

1. *Accepts*-Λ: Given a TM T, is $\Lambda \in L(T)$?

Proof.

1. Prove that *Accepts* $\leq$ *Accepts*-Λ . . .

Reduction from *Accepts* to *Accepts*-Λ.

Instance of *Accepts* is (T_1, x) for TM T_1 and string x.
Instance of *Accepts*-Λ is TM T_2.

$T_2 = F(T_1, x) =$

$$\textit{Write}(x) \rightarrow T_1$$

T_2 accepts Λ, if and only if T_1 accepts x.

If we had an algorithm/TM A_2 to solve *Accepts*-Λ,
then we would also have an algorithm/TM A_1 to solve *Accepts*,
as follows:

A_1:
Given instance (T_1, x) of *Accepts*,
1. construct $T_2 = F(T_1, x)$;
2. run A_2 on T_2.

A_1 answers 'yes' for (T_1, x),
if and only if A_2 answers 'yes' for T_2,
if and only if T_2 is yes-instance of *Accepts*-Λ (T_2 accepts Λ),
if and only if (T_1, x) is yes-instance of *Accepts* (T_1 accepts x)

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. . .

Informal proof:
Suppose that $P_1 \leq P_2$, and that function F maps instance I_1 of P_1 to instance $I_2 = F(I_1)$ of P_2 with same answer yes/no

If we have an algorithm/TM A_2 to solve P_2,
then we also have an algorithm/TM A_1 to solve P_1,
as follows:

A_1:
Given instance I_1 of P_1,
1. construct $I_2 = F(I_1)$;
2. run A_2 on I_2.

$$\begin{array}{lccccc} & I_1 & \longrightarrow & I_2 & \longrightarrow & \text{yes/no} \\ A_1: & & F & & A_2 & \end{array}$$

A_1 answers 'yes' for I_1,
if and only if A_2 answers 'yes' for I_2,
if and only $I_2 = F(I_1)$ is yes-instance of P_2,
if and only if I_1 is yes-instance of P_1

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 $x \in Y(P)$?”

Theorem 9.9. The following five decision problems are undecidable.

1. *Accepts*-Λ: Given a TM T, is $\Lambda \in L(T)$?

Proof.

1. Prove that *Accepts* $\leq$ *Accepts*-Λ . . .

Theorem 9.9. The following five decision problems are undecidable.

2. *AcceptsEverything*:
   Given a TM T with input alphabet Σ, is $L(T) = \Sigma^*$?

Proof.

2. Prove that *Accepts-*$\Lambda \leq$ *AcceptsEverything* . . .

Theorem 9.9. The following five decision problems are undecidable.

3. *Subset*: Given two TMs T_1 and T_2, is $L(T_1) \subseteq L(T_2)$?

Proof.

3. Prove that *AcceptsEverything* $\leq$ *Subset* . . .

Theorem 9.9. The following five decision problems are undecidable.

4. *Equivalent*: Given two TMs T_1 and T_2, is $L(T_1) = L(T_2)$

Proof.

4. Prove that *Subset* $\leq$ *Equivalent* . . .

‘The intersection of two Turing machines’

Accepts-Λ: Given a TM T, is $\Lambda \in L(T)$?

Theorem 9.9. The following five decision problems are undecidable.

5. *WritesSymbol*:
   Given a TM T and a symbol a in the tape alphabet of T, does T ever write a if it starts with an empty tape ?

Proof.

5. Prove that *Accepts-Λ* $\leq$ *WritesSymbol* . . .

AtLeast10MovesOn-Λ:
Given a TM T, does T make at least ten moves on input Λ ?

WritesNonblank: Given a TM T, does T ever write a nonblank symbol on input Λ ?

Theorem 9.10.
The decision problem *WritesNonblank* is decidable.

Proof. . .

Definition 9.11. A Language Property of TMs

A property R of Turing machines is called a *language property* if, for every Turing machine T having property R, and every other TM T_1 with $L(T_1) = L(T)$, T_1 also has property R.

A language property of TMs is *nontrivial* if there is at least one TM that has the property and at least one that doesn't.

In fact, a language property is a property *of the languages accepted by* TMs.

Example of nontrivial language property:

2. *AcceptsSomething*:
Given a TM T, is there at least one string in $L(T)$?

Theorem 9.12. Rice's Theorem
If R is a nontrivial language property of TMs, then the decision problem

P_R: Given a TM T, does T have property R ?

is undecidable.

Proof. . .

Prove that *Accepts*-$\Lambda \leq P_R$ . . .

(or that *Accepts*-$\Lambda \leq P_{\text{not}-R}$ . . .)

Examples of decision problems to which Rice's theorem can be applied:

1. *Accepts-L*: Given a TM T, is $L(T) = L$? (assuming . . .)
2. *AcceptsSomething*:
   Given a TM T, is there at least one string in $L(T)$?
3. *AcceptsTwoOrMore*:
   Given a TM T, does $L(T)$ have at least two elements ?
4. *AcceptsFinite*: Given a TM T, is $L(T)$ finite ?
5. *AcceptsRecursive*:
   Given a TM T, is $L(T)$ recursive ? (note that . . .)

All these problems are undecidable.

Rice's theorem cannot be applied (directly)

- if the decision problem does not involve just one TM

Equivalent: Given two TMs T_1 and T_2, is $L(T_1) = L(T_2)$

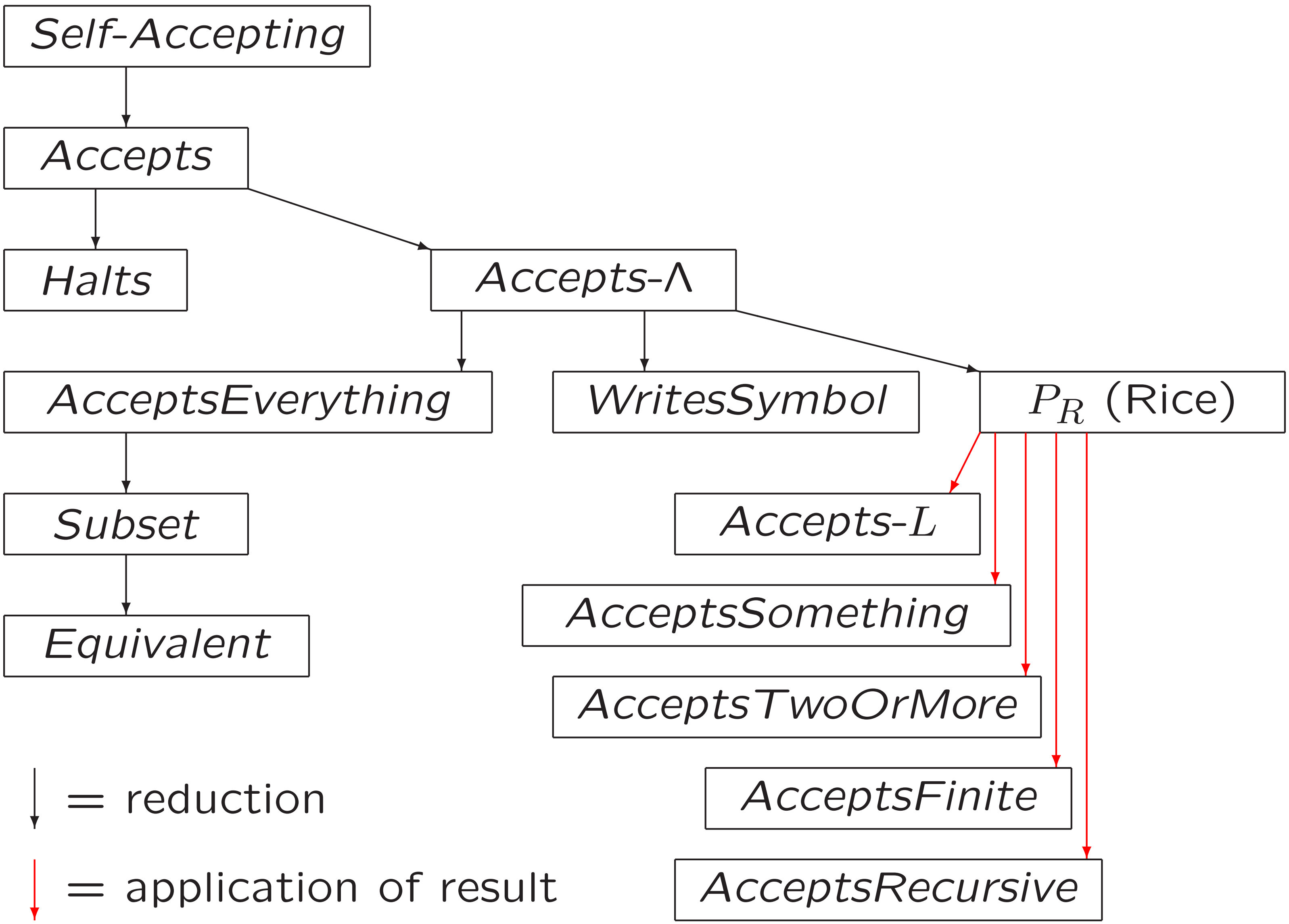
Rice's theorem cannot be applied (directly)

• if the decision problem does not involve just one TM
Equivalent: Given two TMs T_1 and T_2, is $L(T_1) = L(T_2)$

• if the decision problem involves the *operation* of the TM
WritesSymbol: Given a TM T and a symbol a in the tape alphabet of T, does T ever write a if it starts with an empty tape ?
WritesNonblank: Given a TM T, does T ever write a nonblank symbol on input Λ ?

• if the decision problem involves a *trivial* property
Accepts-NSA: Given a TM T, is $L(T) = \mathit{NSA}$?

Undecidable Decision Problems (we have discussed)



Planning

laatste hoor-/werkcollege,
vrijdag 24 maart 2023, 13.15-15.00 uur

tentamen, donderdag 30 maart 2023, 09.00-12.00 uur

vragenuur, 28 maart 2023, 11.00-12.45 uur? Ja!