Praktisch

- Hoocollege nu, werkcollege woensdag
- Huiswerkopgave 4...
- Nakijken huiswerkopgave 3...

A slide from lecture 13:

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 13:

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 $z = z_1 z_2 \ldots z_j$ is a string, where each $z_i \in \mathcal{S}$,

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

A slide from lecture 13:

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.

Computability e itself...

A slide from lecture 13:

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

Let

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

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

A slide from lecture 13:

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

Proof. . .

A slide from lecture 13:

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 $e(T)$?*

instances...

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 13

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,

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

$E(P)$ is 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

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.

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.

$$A_1 : \quad I_1 \xrightarrow[F]{} I_2 \xrightarrow[A_2]{} \text{yes/no}$$

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

Two more decision problems:
Accepts: Given a TM T and a string x, is $x \in L(T)$?
Halts: Given a TM T and a string x, does T halt on input x ?

Self-Accepting: Given a TM T, does T accept the string $e(T)$?
Accepts: Given a TM T and a string x, is $x \in L(T)$?

Theorem 9.8. Both *Accepts* and *Halts* are undecidable.
Proof.
1. Prove that *Self-Accepting* $\leq$ *Accepts* ...

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.

...

Accepts: Given a TM T and a string x, is $x \in L(T)$?
Halts: Given a TM T and a string x, does T halt on input x ?

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.

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 $P_1 \leq P_2$

The formal proof of this result does not have to be known for the exam

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.**)

Exercise 9.7.

As discussed at the beginning of Section 9.3, there is at least one TM T such that the decision problem

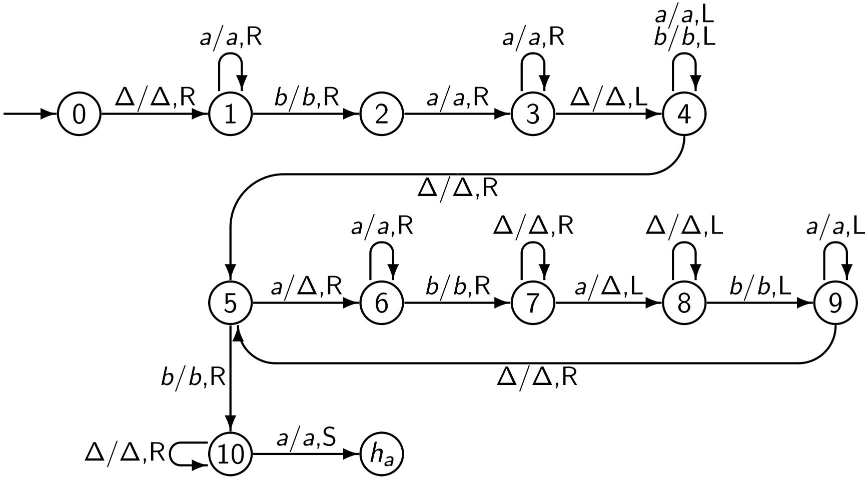
"Given w, does T accept w ?"

is unsolvable.

Show that every TM accepting a nonrecursive language has this property.

A slide from lecture 11

Example 7.7. Accepting $L = \{a^i b a^j \mid 0 \le i < j\}$



What if $x \notin L$?

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

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.
$T_2 = F(T_1, x) =$

$$\mathit{Write(x)} \to T_1$$

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 9.9. 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.

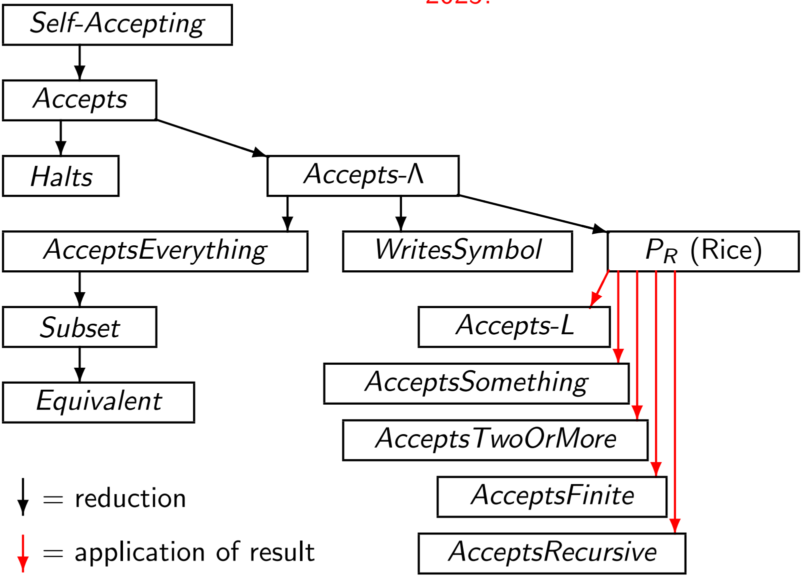
2. Prove that *Accepts-*$\Lambda \leq$ *AcceptsEverything* ...

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 $\mathit{AcceptsEverything} \leq \mathit{Subset}$ ...

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* ...

Undecidable Decision Problems (we have discussed)

WritesSymbol, Rice and its applications have not been discussed in fall 2025!



Planning

- tentamen, vrijdag 9 januari 2026, 09.00-12.00 uur
- vragenuur, donderdag 8 januari 2026, 11.00-12.45 uur