Non computable functions

1 Computablity and Decidability

Curch's Thesis: computability = computability by TM's

computable functions so far:

$$f : \mathbb{N}_0^r \to \mathbb{N}_0$$

1 Computablity and Decidability

Curch's Thesis: computability = computability by TM's

computable functions so far:

$$f : \mathbb{N}_0^r \to \mathbb{N}_0$$

Let A be an alphabet not containing the blank symbol:

$$B \notin A$$

so it is no tape alphabet of a TM (which has to include B)

allowing strings $w \in A^*$ as inputs:

1 Computablity and Decidability

Curch's Thesis: computability = computability by TM's

computable functions so far:

$$f : \mathbb{N}_0^r \to \mathbb{N}_0$$

Let A be an alphabet not containing the blank symbol:

$$B \notin A$$

so it is no tape alphabet of a TM (which has to include B)

allowing strings $w \in A^*$ as inputs:

def: function f_M computed by TM M :

$$f_M : A^* \to A^*$$

- if M started with input w (on tape 1 and other tapes empty) halts with inscription $v \in A*$ on tape 1, then

$$f_M(w) = v$$

- otherwise

$$f_M(w) = \Omega$$

  is undefinded

1 Computablity and Decidability

def: function f_M computed by TM M :

$$f_M : A^* \to A^*$$

- if M started with input w (on tape 1 and other tapes empty) halts with inscription $v \in A*$ on tape 1, then

$$f_M(w) = v$$

- otherwise

$$f_M(w) = \Omega$$

  is undefinded

of special interest: 0/1 valued functions.

def: characteristic function χ_L of a language L Let $L \subseteq A^*$. Then

$$\chi_L : A^* \to \{0,1\}$$

is defined by

$$\chi_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

1 Computablity and Decidability

def: function f_M computed by TM M :

$$f_M : A^* \to A^*$$

- if M started with input w (on tape 1 and other tapes empty) halts with inscription $v \in A*$ on tape 1, then

$$f_M(w) = v$$

- otherwise

$$f_M(w) = \Omega$$

  is undefinded

of special interest: 0/1 valued functions.

def: characteristic function χ_L of a language L Let $L \subseteq A^*$. Then

$$\chi_L : A^* \to \{0,1\}$$

is defined by

$$\chi_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

def: decidable languages L is *decidable* if its characteristic function is computed by a TM M

$$\chi_L = f_M$$

Note: as χ_L is total, the machine must halt for each input $w \in A^*$. We say M decides (membership in) L.

1 Computablity and Decidability

def: function f_M computed by TM M :

$$f_M : A^* \to A^*$$

- if M started with input w (on tape 1 and other tapes empty) halts with inscription $v \in A*$ on tape 1, then

$$f_M(w) = v$$

- otherwise

$$f_M(w) = \Omega$$

  is undefinded

of special interest: 0/1 valued functions.

def: characteristic function χ_L of a language L Let $L \subseteq A^*$. Then

$$\chi_L : A^* \to \{0,1\}$$

is defined by

$$\chi_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

def: decidable languages L is *decidable* if its characteristic function is computed by a TM M

$$\chi_L = f_M$$

Note: as χ_L is total, the machine must halt for each input $w \in A^*$. We say M decides (membership in) L.

def: acceptor for L M is acceptor for l iff

$$\forall w \in A^*. \quad M \text{ started with } w \text{ halts } \leftrightarrow w \in L$$

Note:

- L decidable $\to \exists$ acceptor for L (trivial exercise).
- we will see, that the converse is not true.

1 Computablity and Decidability

def: function f_M computed by TM M :

$$f_M : A^* \to A^*$$

- if M started with input w (on tape 1 and other tapes empty) halts with inscription $v \in A*$ on tape 1, then

$$f_M(w) = v$$

- otherwise

$$f_M(w) = \Omega$$

  is undefinded

of special interest: 0/1 valued functions.

def: characteristic function χ_L of a language L Let $L \subseteq A^*$. Then

$$\chi_L : A^* \to \{0,1\}$$

is defined by

$$\chi_L(w) = \begin{cases} 1 & w \in L \\ 0 & w \notin L \end{cases}$$

def: decidable languages L is *decidable* if its characteristic function is computed by a TM M

$$\chi_L = f_M$$

Note: as χ_L is total, the machine must halt for each input $w \in A^*$. We say M decides (membership in) L.

def: acceptor for L M is acceptor for l iff

$$\forall w \in A^*. \quad M \text{ started with } w \text{ halts } \leftrightarrow w \in L$$

Note:

- L decidable $\to \exists$ acceptor for L (trivial exercise).
- we will see, that the converse is not true.

no condition if $w \notin L$

2 Treating Turing machines as data; Goedelisations

For digital natives:

- Turing machines are programs
- programs = strings = data

is obvious.

This was not always so

2 Treating Turing machines as data; Goedelisations

coding 1 tape Turing machines as strings:

- Let

$$M = (Z, A, \delta, z_0, E)$$

be a 1 tape TM. W.l.o.g

$$\begin{aligned} Z &= \{z_0, \ldots, z_r\} \\ A &= \{a_0, \ldots, a_s\} \\ E &= \{z : \delta(z,a) \text{ undefined for all } a\} \end{aligned}$$

For digital natives:

- Turing machines are programs
- programs = strings = data

is obvious.

This was not always so

2 Treating Turing machines as data; Goedelisations

coding 1 tape Turing machines as strings:

- Let

$$M = (Z, A, \delta, z_0, E)$$

  be a 1 tape TM. W.l.o.g

$$\begin{aligned} Z &= \{z_0, \ldots, z_r\} \\ A &= \{a_0, \ldots, a_s\} \\ E &= \{z \,:\, \delta(z,a) \text{ undefined for all } a\} \end{aligned}$$

- now code

$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$

  as

$$\#bin(i)\#bin(j)\#bin(k)\#bin(\ell)\#m'\#$$

  with

$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

For digital natives:

- Turing machines are programs
- programs = strings = data

is obvious.

This was not always so

2 Treating Turing machines as data; Goedelisations

coding 1 tape Turing machines as strings:

- Let

$$M = (Z, A, \delta, z_0, E)$$

  be a 1 tape TM. W.l.o.g

$$\begin{aligned} Z &= \{z_0, \dots, z_r\} \\ A &= \{a_0, \dots, a_s\} \\ E &= \{z \,:\, \delta(z,a) \text{ undefined for all } a\} \end{aligned}$$

- now code

$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$

  as

$$\#bin(i)\#bin(j)\#bin(k)\#bin(\ell)\#m'\#$$

  with

$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

- concatenate all these strings with result

$$code'(M) \in \{0,1,\#\}^*$$

For digital natives:

- Turing machines are programs
- programs = strings = data

is obvious.

This was not always so

2 Treating Turing machines as data; Goedelisations

coding 1 tape Turing machines as strings:

- Let

$$M = (Z, A, \delta, z_0, E)$$

be a 1 tape TM. W.l.o.g

$$\begin{aligned} Z &= \{z_0, \dots, z_r\} \\ A &= \{a_0, \dots, a_s\} \\ E &= \{z \,:\, \delta(z,a) \text{ undefined for all } a\} \end{aligned}$$

- now code

$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$

as

$$\#bin(i)\#bin(j)\#bin(k)\#bin(\ell)\#m'\#$$

with

$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

- concatenate all these strings with result

$$code'(M) \in \{0, 1, \#\}^*$$

- finally obtain $code(x)$ by replacing in $code'(M)$ symbol by symbol

$$0 \text{ by}: \ 00 \quad , \quad 1 \text{ by}: \ 01 \quad , \quad \# \text{ by}: \ 10$$

2 Treating Turing machines as data; Goedelisations

coding 1 tape Turing machines as strings:

- Let
$$M = (Z, A, \delta, z_0, E)$$
be a 1 tape TM. W.l.o.g
$$\begin{aligned} Z &= \{z_0, \ldots, z_r\} \\ A &= \{a_0, \ldots, a_s\} \\ E &= \{z \,:\, \delta(z,a) \text{ undefined for all } a\} \end{aligned}$$

- now code
$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$
as
$$\#bin(i)\#bin(j)\#bin(k)\#bin(\ell)\#m'\#$$
with
$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

- concatenate all these strings with result
$$code'(M) \in \{0, 1, \#\}^*$$

- finally obtain $code(x)$ by replacing in $code'(M)$ symbol by symbol
$$0 \text{ by}: \ 00 \quad , \quad 1 \text{ by}: \ 01 \quad , \quad \# \text{ by}: \ 10$$

names:

- $code(x)$ is called *Goedelisation* of M
- $\langle code(x) \rangle \in \mathbb{N}_0$ is called *Goedel number* of M

2 Treating Turing machines as data; Goedelisations

coding 1 tape Turing machines as strings:

- Let

$$M = (Z, A, \delta, z_0, E)$$

be a 1 tape TM. W.l.o.g

$$\begin{aligned} Z &= \{z_0, \ldots, z_r\} \\ A &= \{a_0, \ldots, a_s\} \\ E &= \{z \,:\, \delta(z,a) \text{ undefined for all } a\} \end{aligned}$$

- now code

$$\delta(z_i, a_j) = (z_k, a_\ell, m)$$

as

$$\#bin(i)\#bin(j)\#bin(k)\#bin(\ell)\#m'\#$$

with

$$m' = \begin{cases} 00 & m = L \\ 01 & m = N \\ 10 & m = R \end{cases}$$

- concatenate all these strings with result

$$code'(M) \in \{0,1,\#\}^*$$

- finally obtain $code(x)$ by replacing in $code'(M)$ symbol by symbol

$$0 \text{ by}:\ 00 \quad , \quad 1 \text{ by}:\ 01 \quad , \quad \# \text{ by}:\ 10$$

names:

- $code(x)$ is called *Goedelisation* of M
- $\langle code(x) \rangle \in \mathbb{N}_0$ is called *Goedel number* of M

inverse mapping Let $u \in \mathbb{B}^*$. We define Turing machine M_u (the TM coded by u) as

- the machine M with $code(M) = u$ if u is code of a machine
- M_0 otherwise; M_0 ignores the input and halts ($E = \{z_0\}$).

3 Halting problem

def: halting problem

$$H = \{u\#v : u, v \in \mathbb{B}^* \ , \ \ M_u \text{ started with } v \text{ halts}\}$$

def: special halting problem

$$K = \{u \in \mathbb{B}^* : \ M_u \text{ started with } u \text{ halts}\}$$

3 Halting problem

def: halting problem

$$H = \{u\#v : u, v \in \mathbb{B}^* \;,\; M_u \text{ started with } v \text{ halts}\}$$

def: special halting problem

$$K = \{u \in \mathbb{B}^* : \; M_u \text{ started with } u \text{ halts}\}$$

THE classic result of TCS

Lemma 1. *The special halting problem K is undecidable.*

3 Halting problem

def: halting problem

$$H = \{u\#v : u,v \in \mathbb{B}^* \ , \ M_u \text{ started with } v \text{ halts}\}$$

def: special halting problem

$$K = \{u \in \mathbb{B}^* : \ M_u \text{ started with } u \text{ halts}\}$$

THE classic result of TCS

Lemma 1. *The special halting problem K is undecidable.*

- by contradiction. Assume machine Q computes χ_K, i.e.
- Q started with u halts for all inputs with result

$$f_Q(u) = \begin{cases} 1 & M_u \text{ started with } u \text{ halts} \\ 0 & M_u \text{ started with } u \text{ does not halt} \end{cases}$$

3 Halting problem

def: halting problem

$$H = \{u\#v : u, v \in \mathbb{B}^* \ , \ \ M_u \text{ started with } v \text{ halts}\}$$

def: special halting problem

$$K = \{u \in \mathbb{B}^* : \ M_u \text{ started with } u \text{ halts}\}$$

THE classic result of TCS

Lemma 1. *The special halting problem K is undecidable.*

- by contradiction. Assume machine Q computes χ_K, i.e.
- Q started with u halts for all inputs with result

$$f_Q(u) = \begin{cases} 1 & M_u \text{ started with } u \text{ halts} \\ 0 & M_u \text{ started with } u \text{ does not halt} \end{cases}$$

- modify Q to machine R as illustrated in figure 1.

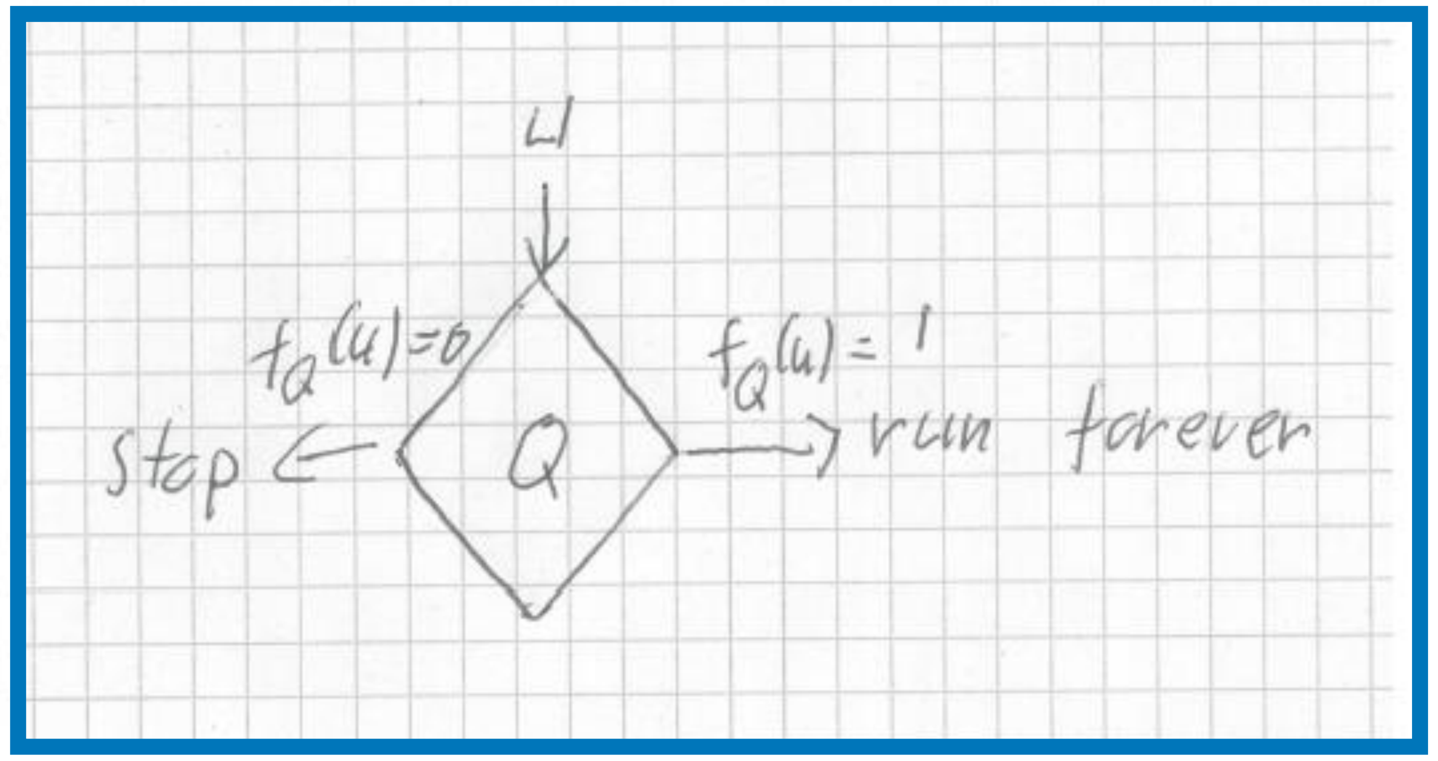

Figure 1: construction of machine R from machine Q

With input u

run Q with input u with result $f_Q(u)$. If $f_Q(u) = 0$: stop. If $f_Q(u) = 1$: run forever.

3 Halting problem

def: halting problem

$$H = \{u\#v : u, v \in \mathbb{B}^* \ , \ \ M_u \text{ started with } v \text{ halts}\}$$

def: special halting problem

$$K = \{u \in \mathbb{B}^* : \ M_u \text{ started with } u \text{ halts}\}$$

THE classic result of TCS

Lemma 1. *The special halting problem K is undecidable.*

- by contradiction. Assume machine Q computes χ_K, i.e.
- Q started with u halts for all inputs with result

$$f_Q(u) = \begin{cases} 1 & M_u \text{ started with } u \text{ halts} \\ 0 & M_u \text{ started with } u \text{ does not halt} \end{cases}$$

- modify Q to machine R as illustrated in figure 1.

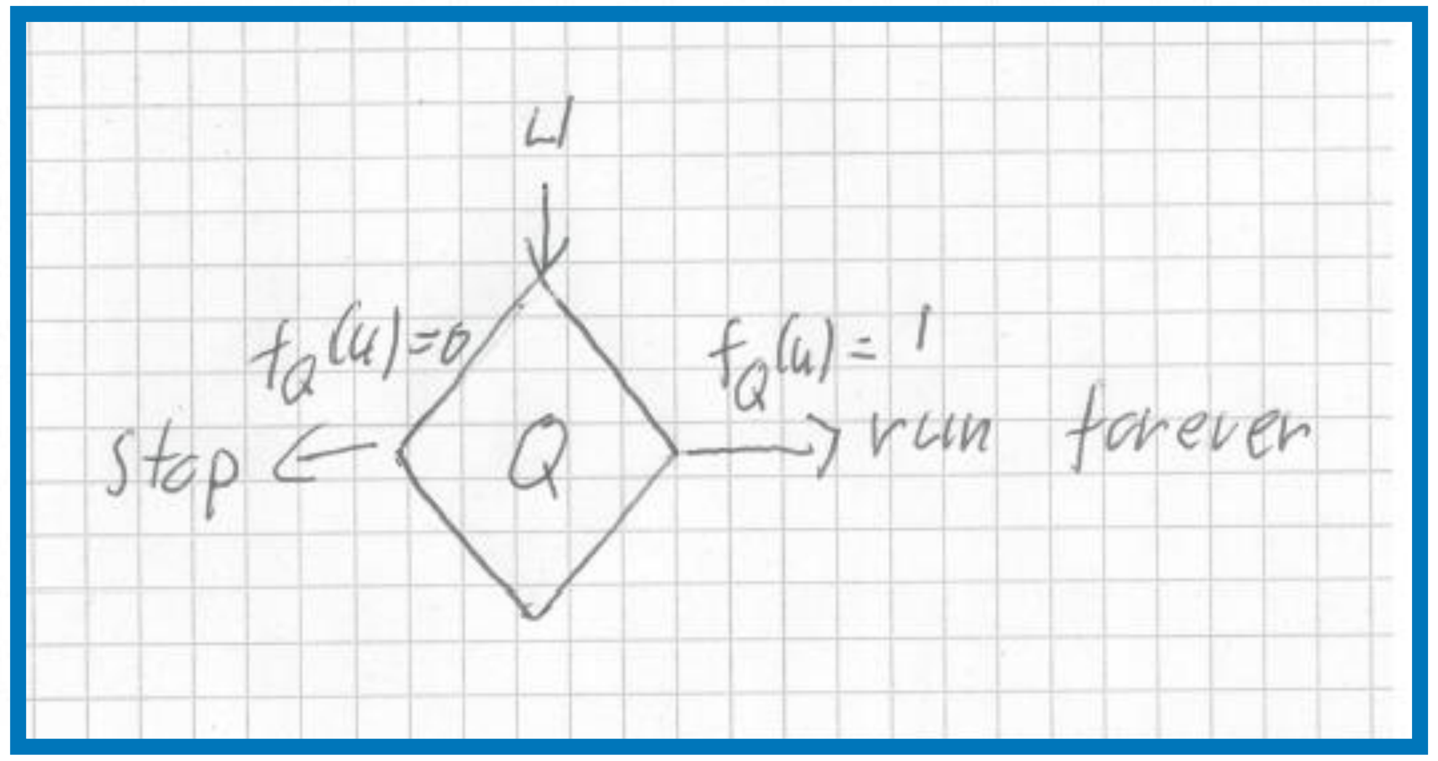

Figure 1: construction of machine R from machine Q

With input u

run Q with input u with result $f_Q(u)$. If $f_Q(u) = 0$: stop. If $f_Q(u) = 1$: run forever.

- Let $r = code(R)$ and consider $R = M_r$ started with r. Then

$$\begin{aligned} & R \text{ started with } r \text{ halts} \\ \leftrightarrow \quad & f_Q(r) = 1 \quad \text{(assumption about } Q) \\ \leftrightarrow \quad & R \text{ started with } r \text{ does not halt} \quad \text{(construction of } R) \end{aligned}$$

4 Reducibility

4.1 Basics

def: reducibility Let

$$L, L' \subseteq A^*$$

be languages. L is *reducible* to L' and write

$$L \leq L'$$

iff there is a total computable function

$$f : A^* \to A^*$$

such that

$$\forall w \in A. \quad w \in L \leftrightarrow f(w) \in L'$$

i.e the question $w \in L$? is reformulated as $f(w) \in L'$?

4 Reducibility

4 Reducibility

4.1 Basics

def: reducibility Let

$$L, L' \subseteq A^*$$

be languages. L is *reducible* to L' and write

$$L \leq L'$$

iff there is a total computable function

$$f : A^* \to A^*$$

such that

$$\forall w \in A. \quad w \in L \leftrightarrow f(w) \in L'$$

i.e the question $w \in L$? is reformulated as $f(w) \in L'$?

Lemma 2. *Let $L \leq L'$. Then*

- *if L' is decidable, then L is decidable*
- *if L is undecidable, then L' is undecidable.*

Proof. easy exercise

4 Reducibility

4 Reducibility

4.1 Basics

def: reducibility Let

$$L, L' \subseteq A^*$$

be languages. L is *reducible* to L' and write

$$L \leq L'$$

iff there is a total computable function

$$f : A^* \to A^*$$

such that

$$\forall w \in A. \quad w \in L \leftrightarrow f(w) \in L'$$

i.e the question $w \in L$? is reformulated as $f(w) \in L'$?

Lemma 2. *Let $L \leq L'$. Then*

- *if L' is decidable, then L is decidable*
- *if L is undecidable, then L' is undecidable.*

Proof. easy exercise

4 Reducibility

example:

Lemma 3. *$K \leq H$, hence the halting problem H is undecidable:*

Proof.

$$f(u) = u\#u$$

4 Reducibility

4.1 Basics

def: reducibility Let

$$L, L' \subseteq A^*$$

be languages. L is *reducible* to L' and write

$$L \leq L'$$

iff there is a total computable function

$$f : A^* \to A^*$$

such that

$$\forall w \in A. \quad w \in L \leftrightarrow f(w) \in L'$$

i.e the question $w \in L$? is reformulated as $f(w) \in L'$?

Lemma 2. *Let $L \leq L'$. Then*

- *if L' is decidable, then L is decidable*
- *if L is undecidable, then L' is undecidable.*

Proof. easy exercise

4 Reducibility

example:

Lemma 3. *$K \leq H$, hence the halting problem H is undecidable:*

Proof.

$$f(u) = u\#u$$

remarks

- $\leq$ is transitive
- we define
$$L \equiv L' \leftrightarrow L \leq L' \,\wedge\, L' \leq L$$
- $\equiv$ is equivalenve relation
- classes of equally undecidable problems (studied recursion theory; IMHO moderately exciting)

4 Reducibility

4.2 Two more examples (of program properties)

- $$C = \{u \ : \ M_u \text{computes a constant function}\}$$

- $$K_0 = \{u \ : \ M_u \text{started with empty tape halts}\}$$

4.2 Two more examples (of program properties)

- $$C = \{u \ : \ M_u \text{computes a constant function}\}$$
- $$K_0 = \{u \ : \ M_u \text{started with empty tape halts}\}$$

Lemma 4. *$K \leq K_0$, i.e. K_0 is undecidable.*

- f will transform goedelisations u into goedelisations $f(u)$
- for $u \in A^*$ machine $M_{f(u)}$ writes u on the empty tape and then behaves like M_u.

4 Reducibility

4.2 Two more examples (of program properties)

- $$C = \{u \; : \; M_u \text{computes a constant function}\}$$

- $$K_0 = \{u \; : \; M_u \text{started with empty tape halts}\}$$

Lemma 4. *$K \leq K_0$, i.e. K_0 is undecidable.*

- f will transform goedelisations u into goedelisations $f(u)$
- for $u \in A^*$ machine $M_{f(u)}$ writes u on the empty tape and then behaves like M_u.

Lemma 5. *$K_0 \leq C$, i.e. C is undecidable.*

machine $M_{f(u)}$ started with input x

- erases x
- behaves like M_u (started with empty tape)
- if it halts: output 1.

4.3 Rice's theorem

question: can we decide any nontrivial property of programs/goedelisations u?

trivial property: holds for all programs or for none.

4.3 Rice's theorem

question: can we decide any nontrivial property of programs/goedelisations u?

trivial property: holds for all programs or for none.

Lemma 6. *Partition the set of computable functions*

$$R = \{f : \{0,1,\#\}^* \rightarrow \{0,1,\#\}^* \mid f \textit{ is computable}\}$$

into

$$R = R_1 \,\dot\cup\, R_2$$

in a nontrivial way

$$R_1 \neq \emptyset \;,\; R_2 \neq \emptyset$$

Then the set

$$R' = \{u \;:\; f_{M_u} \in R_1\}$$

is not decidable

show $K_0 \leq R'$ by program transformation f s.t,

$M_{f(u)}$ computes function in $R' \leftrightarrow M_u$ started with empty tape halts

4.3 Rice's theorem

question: can we decide any nontrivial property of programs/goedelisations u?

trivial property: holds for all programs or for none.

Lemma 6. *Partition the set of computable functions*

$$R = \{f \ : \{0,1,\#\}^* \rightarrow \{0,1,\#\}^* \mid f \text{ is computable}\}$$

into

$$R = R_1 \,\dot{\cup} R_2$$

in a nontrivial way

$$R_1 \neq \emptyset \ , \ R_2 \neq \emptyset$$

Then the set

$$R' = \{u \ : \ f_{M_u} \in R_1\}$$

is not decidable

show $K_0 \leq R'$ by program transformation f s.t,

$M_{f(u)}$ computes function in $R' \leftrightarrow M_u$ started with empty tape halts

- let Ω be the function, which is everywhere undefined. W.l.o.g $\Omega \in R_2$.
- let M' a TM computing a function in R_1 (exists, as $R_1 \neq \emptyset$)
- definition of transformed machine $M_{f(u)}$ started with input x
  1. ignore but save input x (e.g. save on extra track)
  2. behave as M_u started with empty tape
  3. if M_u halts: behave like M' started with x
- then the function computed by $M_{f(u)}$ is

$$f_{M_{f(u)}} = \begin{cases} \Omega & u \notin K_0 \\ f_{M'} & u \in K_0 \end{cases}$$

hence

$$f(u) \in R' \leftrightarrow u \ \in K_0$$

5 Universal Turing machines

Lemma 7. *There exists a (universal) Turing machine U such that for all u, v machine U started with $u\#v$ simulates M_u started with v*

For digital natives

- being able to write in language L
- an interpreter for programs in L

is not so surprising. This was not always so

5 Universal Turing machines

Lemma 7. *There exists a (universal) Turing machine U such that for all u, v machine U started with $u\#v$ simulates M_u started with v*

Very quick proof: write in C a TM interpreter U'. With input $u\#v$

- decode u to TM M
- simulate M started with v
- simulate U' by 1 tape TM.

In the future we will construct U directly as a TM.

For digital natives

- being able to write in language L
- an interpreter for programs in L

is not so surprising. This was not always so

5 Universal Turing machines

Lemma 7. *There exists a (universal) Turing machine U such that for all u, v machine U started with $u\#v$ simulates M_u started with v*

Very quick proof: write in C a TM interpreter U'. With input $u\#v$

- decode u to TM M
- simulate M started with v
- simulate U' by 1 tape TM.

In the future we will construct U directly as a TM.

observe: $\boxed{U \text{ is acceptor for the halting problem } H.}$

For digital natives

- being able to write in language L
- an interpreter for programs in L

is not so surprising. This was not always so

5 Universal Turing machines

For digital natives

- being able to write in language L
- an interpreter for programs in L

is not so surprising. This was not always so

Lemma 7. *There exists a (universal) Turing machine U such that for all u, v machine U started with $u\#v$ simulates M_u started with v*

Very quick proof: write in C a TM interpreter U'. With input $u\#v$

- decode u to TM M
- simulate M started with v
- simulate U' by 1 tape TM.

In the future we will construct U directly as a TM.

observe: $\boxed{U \text{ is acceptor for the halting problem } H.}$

Lemma 8. *A language L is decidable iff there exist acceptors for L and $\overline{L}$*

- $\rightarrow$: trivial (why?)
- $\leftarrow$: by *dovetailing*. Given input $w \in A^*$ alternate between simultating 1 step of
  1. acceptor A_1 for L started with w
  2. acceptor A_2 for $\overline{L}$ strated with w
- if A_1 halts first output 1, if A_2 halts first output 0.

5 Universal Turing machines

Lemma 7. *There exists a (universal) Turing machine U such that for all u,v machine U started with $u\#v$ simulates M_u started with v*

Very quick proof: write in C a TM interpreter U'. With input $u\#v$

- decode u to TM M
- simulate M started with v
- simulate U' by 1 tape TM.

In the future we will construct U directly as a TM.

observe: $\boxed{U \text{ is acceptor for the halting problem } H.}$

Lemma 8. *A language L is decidable iff there exist acceptors for L and $\overline{L}$*

- $\rightarrow$: trivial (why?)
- $\leftarrow$: by *dovetailing*. Given input $w \in A^*$ alternate between simultating 1 step of
  1. acceptor A_1 for L started with w
  2. acceptor A_2 for $\overline{L}$ strated with w
- if A_1 halts first output 1, if A_2 halts first output 0.

Lemma 9. *$\overline{H}$ has no acceptor*

Proof. universal machine U is acceptor for H; if $\overline{H}$ would have an acceptor, then H would be decidable. □

6 Recursively enumerable languages/sets

def: recursively enumerable language Language $L \subseteq A^*$ is recursively enumerable (r.e.) if there is a TM M s.t. M started with empty tape outputs (say on an output tape) exactly the words in L in some order.

6 Recursively enumerable languages/sets

def: recursively enumerable language Language $L \subseteq A^*$ is recursively enumerable (r.e.) if there is a TM M s.t. M started with empty tape outputs (say on an output tape) exactly the words in L in some order.

Lemma 10. *L is r.e. iff L has an acceptor.*

6 Recursively enumerable languages/sets

def: **recursively enumerable language** Language $L \subseteq A^*$ is recursively enumerable (r.e.) if there is a TM M s.t. M started with empty tape outputs (say on an output tape) exactly the words in L in some order.

Lemma 10. *L is r.e. iff L has an acceptor.*

$\rightarrow$: assume M enumerates L. Construct acceptor M' for L.

- input w
- enumerate $L = v_1, v_2, \ldots$
- for each output v_i test $v_i = w$. If true, stop.

6 Recursively enumerable languages/sets

def: recursively enumerable language Language $L \subseteq A^*$ is recursively enumerable (r.e.) if there is a TM M s.t. M started with empty tape outputs (say on an output tape) exactly the words in L in some order.

Lemma 10. *L is r.e. iff L has an acceptor.*

$\rightarrow$: assume M enumerates L. Construct acceptor M' for L.

- input w
- enumerate $L = v_1, v_2, \ldots$
- for each output v_i test $v_i = w$. If true, stop.

$\leftarrow$: slightly trickier. Let M be an acceptor for L.

- good news: tapes are infinite, configurations of M are finite. Thus we can store any number r of configurations of M on one tape.
- one can enumerate $A^* = \{v_1, v_2, \ldots\}$ e.g. by length and for equal length in lexicographic order.
- now proceed in rounds. In round 1 create start configuration k_1 of M started with v_1.

6 Recursively enumerable languages/sets

def: recursively enumerable language Language $L \subseteq A^*$ is recursively enumerable (r.e.) if there is a TM M s.t. M started with empty tape outputs (say on an output tape) exactly the words in L in some order.

Lemma 10. *L is r.e. iff L has an acceptor.*

→: assume M enumerates L. Construct acceptor M' for L.

- input w
- enumerate $L = v_1, v_2, \ldots$
- for each output v_i test $v_i = w$. If true, stop.

←: slightly trickier. Let M be an acceptor for L.

- good news: tapes are infinite, configurations of M are finite. Thus we can store any number r of configurations of M on one tape.
- one can enumerate $A^* = \{v_1, v_2, \ldots\}$ e.g. by length and for equal length in lexicographic order.
- now proceed in rounds. In round 1 create start configuration k_1 of M started with v_1.
- In rounds $r > 1$
  1. simulate 1 step of M for each configuration k_i, $i < r$ on the tape. Each such step might require to shift the tape inscription right of k_i to the right or the portion left of k to the left. If the step of k_i leads to an accepting configuration output v_i.
  2. add on the tape as new configuration k_r the start configuration of M started with v_r.

6 Recursively enumerable languages/sets

def: recursively enumerable language Language $L \subseteq A^*$ is recursively enumerable (r.e.) if there is a TM M s.t. M started with empty tape outputs (say on an output tape) exactly the words in L in some order.

Lemma 10. *L is r.e. iff L has an acceptor.*

→: assume M enumerates L. Construct acceptor M' for L.

- input w
- enumerate $L = v_1, v_2, \ldots$
- for each output v_i test $v_i = w$. If true, stop.

←: slightly trickier. Let M be an acceptor for L.

- good news: tapes are infinite, configurations of M are finite. Thus we can store any number r of configurations of M on one tape.
- one can enumerate $A^* = \{v_1, v_2, \ldots\}$ e.g. by length and for equal length in lexicographic order.
- now proceed in rounds. In round 1 create start configuration k_1 of M started with v_1.
- In rounds $r > 1$
  1. simulate 1 step of M for each configuration k_i, $i < r$ on the tape. Each such step might require to shift the tape inscription right of k_i to the right or the portion left of k to the left. If the step of k_i leads to an accepting configuration output v_i.
  2. add on the tape as new configuration k_r the start configuration of M started with v_r.

Lemma 11. *Let L be a type-0 language. Then L has an acceptor.*

6 Recursively enumerable languages/sets

def: recursively enumerable language Language $L \subseteq A^*$ is recursively enumerable (r.e.) if there is a TM M s.t. M started with empty tape outputs (say on an output tape) exactly the words in L in some order.

Lemma 10. *L is r.e. iff L has an acceptor.*

$\rightarrow$: assume M enumerates L. Construct acceptor M' for L.

- input w
- enumerate $L = v_1, v_2, \ldots$
- for each output v_i test $v_i = w$. If true, stop.

$\leftarrow$: slightly trickier. Let M be an acceptor for L.

- good news: tapes are infinite, configurations of M are finite. Thus we can store any number r of configurations of M on one tape.
- one can enumerate $A^* = \{v_1, v_2, \ldots\}$ e.g. by length and for equal length in lexicographic order.
- now proceed in rounds. In round 1 create start configuration k_1 of M started with v_1.
- In rounds $r > 1$
  1. simulate 1 step of M for each configuration k_i, $i < r$ on the tape. Each such step might require to shift the tape inscription right of k_i to the right or the portion left of k to the left. If the step of k_i leads to an accepting configuration output v_i.
  2. add on the tape as new configuration k_r the start configuration of M started with v_r.

Lemma 11. *Let L be a type-0 language. Then L has an acceptor.*

- let $L = L(G)$ be generated by type-0 grammar G.
- on input w enumerate all derivations of G, say by increasing length and for each length in lexicographic order. If any such derivation produces w accept.

6 Recursively enumerable languages/sets

Lemma 12. *If $L \subseteq (A \setminus \{B\})^*$ has an acceptor, then L is a type-0 language.*

6 Recursively enumerable languages/sets

Lemma 12. *If $L \subseteq (A \setminus \{B\})^*$ has an acceptor, then L is a type-0 language.*

Given an acceptor $M = (Z, A, \delta, z_0, E)$ we construct a grammar $G = (A \setminus \{B\}, N, P, S)$ with $L(G) = \{w \in (A \setminus \{B\})^* \ : \ M \text{ accepts } w\}$.

- nonterminals of G:

$$N = A \ \cup \ A \times A \ \cup \ Z \ \cup \ \{S, A_1, A_2\}$$

  Alphabet symbols $(a_1, a_2) \in A \times A$ occupy 2 tracks with a_1 on upper track, a_2 on lower track.

6 Recursively enumerable languages/sets

Lemma 12. *If $L \subseteq (A \setminus \{B\})^*$ has an acceptor, then L is a type-0 language.*

Given an acceptor $M = (Z, A, \delta, z_0, E)$ we construct a grammar $G = (A \setminus \{B\}, N, P, S)$ with $L(G) = \{w \in (A \setminus \{B\})^* \ : \ M \text{ accepts } w\}$.

- nonterminals of G:

$$N = A \ \cup \ A \times A \ \cup \ Z \ \cup \ \{S, A_1, A_2\}$$

  Alphabet symbols $(a_1, a_2) \in A \times A$ occupy 2 tracks with a_1 on upper track, a_2 on lower track.

- intuition for next rules: for possible inputs $w = w[1:n]$ of TM M the grammar can generate strings of the form

$$(B,B)^u z_0 (w_1, w_1) \dots (w_n, w_n)(B,B)^v$$

  i.e. copies of w on both tracks, surrounded by enough B's on both tracks. Productions for this

  1. $S \to (B,B)S \mid z_0 A_1$
  2. $A_1 \to (a,a)A_1 \mid A_2$ for all $a \in A \setminus \{B\}$
  3. $A_2 \to (B,B)A_2 \mid \varepsilon$

6 Recursively enumerable languages/sets

Lemma 12. *If $L \subseteq (A \setminus \{B\})^*$ has an acceptor, then L is a type-0 language.*

Given an acceptor $M = (Z, A, \delta, z_0, E)$ we construct a grammar $G = (A \setminus \{B\}, N, P, S)$ with $L(G) = \{w \in (A \setminus \{B\})^* \;:\; M \text{ accepts } w\}$.

- nonterminals of G:

$$N = A \cup A \times A \cup Z \cup \{S, A_1, A_2\}$$

  Alphabet symbols $(a_1, a_2) \in A \times A$ occupy 2 tracks with a_1 on upper track, a_2 on lower track.

- intuition for next rules: for possible inputs $w = w[1:n]$ of TM M the grammar can generate strings of the form

$$(B,B)^u z_0 (w_1, w_1) \ldots (w_n, w_n)(B,B)^v$$

  i.e. copies of w on both tracks, surrounded by enough B's on both tracks. Productions for this

  1. $S \to (B,B)S \mid z_0 A_1$
  2. $A_1 \to (a,a)A_1 \mid A_2$ for all $a \in A \setminus \{B\}$
  3. $A_2 \to (B,B)A_2 \mid \varepsilon$

- simulate M on the lower track (and remember input on upper track)

  1. if $\delta(z,a) = (z', c, L)$ then for all $b, C, D \in A$ production $(C,b)z(D,a) \to z'(C,b)(D,c)$
  2. if $\delta(z,a) = (z', c, N)$ then for all $b, D \in A$ production $z(D,a) \to z'(D,c)$
  3. if $\delta(z,a) = (z', c, R)$ then for all $b, D \in A$ production $z(D,a) \to (D,c)z'$

6 Recursively enumerable languages/sets

Lemma 12. *If $L \subseteq (A \setminus \{B\})^*$ has an acceptor, then L is a type-0 language.*

Given an acceptor $M = (Z, A, \delta, z_0, E)$ we construct a grammar $G = (A \setminus \{B\}, N, P, S)$ with $L(G) = \{w \in (A \setminus \{B\})^* \ : \ M \text{ accepts } w\}$.

- nonterminals of G:

$$N = A \cup A \times A \cup Z \cup \{S, A_1, A_2\}$$

  Alphabet symbols $(a_1, a_2) \in A \times A$ occupy 2 tracks with a_1 on upper track, a_2 on lower track.

- intuition for next rules: for possible inputs $w = w[1:n]$ of TM M the grammar can generate strings of the form

$$(B,B)^u z_0 (w_1, w_1) \dots (w_n, w_n)(B,B)^v$$

  i.e. copies of w on both tracks, surrounded by enough B's on both tracks. Productions for this

  1. $S \to (B,B)S \mid z_0 A_1$
  2. $A_1 \to (a,a)A_1 \mid A_2$ for all $a \in A \setminus \{B\}$
  3. $A_2 \to (B,B)A_2 \mid \varepsilon$

- simulate M on the lower track (and remember input on upper track)

  1. if $\delta(z,a) = (z', c, L)$ then for all $b, C, D \in A$ production $(C,b)z(D,a) \to z'(C,b)(D,c)$
  2. if $\delta(z,a) = (z', c, N)$ then for all $b, D \in A$ production $z(D,a) \to z'(D,c)$
  3. if $\delta(z,a) = (z', c, R)$ then for all $b, D \in A$ production $z(D,a) \to (D,c)z'$

- if this reaches an end state $z \in E$ throw away right components of pairs, i.e. the lower track. For all $z \in E$ and all $a, D \in A'$ productions

$$z(a,D) \to zaz \quad , \quad (a,D)z \to zaz$$

6 Recursively enumerable languages/sets

Lemma 12. *If $L \subseteq (A \setminus \{B\})^*$ has an acceptor, then L is a type-0 language.*

Given an acceptor $M = (Z, A, \delta, z_0, E)$ we construct a grammar $G = (A \setminus \{B\}, N, P, S)$ with $L(G) = \{w \in (A \setminus \{B\})^* \ : \ M \text{ accepts } w\}$.

- nonterminals of G:

$$N = A \ \cup \ A \times A \ \cup \ Z \ \cup \ \{S, A_1, A_2\}$$

  Alphabet symbols $(a_1, a_2) \in A \times A$ occupy 2 tracks with a_1 on upper track, a_2 on lower track.

- intuition for next rules: for possible inputs $w = w[1:n]$ of TM M the grammar can generate strings of the form

$$(B,B)^u z_0 (w_1, w_1) \dots (w_n, w_n)(B,B)^v$$

  i.e. copies of w on both tracks, surrounded by enough B's on both tracks. Productions for this

  1. $S \rightarrow (B,B)S \mid z_0 A_1$
  2. $A_1 \rightarrow (a,a)A_1 \mid A_2$ for all $a \in A \setminus \{B\}$
  3. $A_2 \rightarrow (B,B)A_2 \mid \varepsilon$

- simulate M on the lower track (and remember input on upper track)

  1. if $\delta(z,a) = (z', c, L)$ then for all $b, C, D \in A$ production $(C,b)z(D,a) \rightarrow z'(C,b)(D,c)$
  2. if $\delta(z,a) = (z', c, N)$ then for all $b, D \in A$ production $z(D,a) \rightarrow z'(D,c)$
  3. if $\delta(z,a) = (z', c, R)$ then for all $b, D \in A$ production $z(D,a) \rightarrow (D,c)z'$

- if this reaches an end state $z \in E$ throw away right components of pairs, i.e. the lower track. For all $z \in E$ and all $a, D \in A'$ productions

$$z(a,D) \rightarrow zaz \quad , \quad (a,D)z \rightarrow zaz$$

- finally trow away blanks and the copies of states z. For all $z \in E$ productions

$$B \rightarrow \varepsilon \quad , \quad z \rightarrow \varepsilon$$

6 Recursively enumerable languages/sets

Lemma 12. *If $L \subseteq (A \setminus \{B\})^*$ has an acceptor, then L is a type-0 language.*

Given an acceptor $M = (Z, A, \delta, z_0, E)$ we construct a grammar $G = (A \setminus \{B\}, N, P, S)$ with $L(G) = \{w \in (A \setminus \{B\})^* \ : \ M \text{ accepts } w\}$.

- nonterminals of G:

$$N = A \cup A \times A \cup Z \cup \{S, A_1, A_2\}$$

  Alphabet symbols $(a_1, a_2) \in A \times A$ occupy 2 tracks with a_1 on upper track, a_2 on lower track.

- intuition for next rules: for possible inputs $w = w[1:n]$ of TM M the grammar can generate strings of the form

$$(B,B)^u z_0 (w_1, w_1) \dots (w_n, w_n)(B,B)^v$$

  i.e. copies of w on both tracks, surrounded by enough B's on both tracks. Productions for this

  1. $S \to (B,B)S \mid z_0 A_1$
  2. $A_1 \to (a,a)A_1 \mid A_2$ for all $a \in A \setminus \{B\}$
  3. $A_2 \to (B,B)A_2 \mid \varepsilon$

- simulate M on the lower track (and remember input on upper track)

  1. if $\delta(z,a) = (z', c, L)$ then for all $b, C, D \in A$ production $(C,b)z(D,a) \to z'(C,b)(D,c)$
  2. if $\delta(z,a) = (z', c, N)$ then for all $b, D \in A$ production $z(D,a) \to z'(D,c)$
  3. if $\delta(z,a) = (z', c, R)$ then for all $b, D \in A$ production $z(D,a) \to (D,c)z'$

- if this reaches an end state $z \in E$ throw away right components of pairs, i.e. the lower track. For all $z \in E$ and all $a, D \in A'$ productions

$$z(a,D) \to zaz \quad , \quad (a,D)z \to zaz$$

- finally trow away blanks and the copies of states z. For all $z \in E$ productions

$$B \to \varepsilon \quad , \quad z \to \varepsilon$$

Lemma 13. *The type-0 languages are exactly the r.e. languages.*

Lemma 14. *There are undecidable type-0 languages.* why?

7 The recursion theorem (Kleene 1938)

question: is there a TM M_u, such that M_u started on empty tape prints its own goedelisation u.

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

7 The recursion theorem (Kleene 1938)

question: is there a TM M_u, such that M_u started on empty tape prints its own goedelisation u.

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

proof (magic): construct total computable program transformation $\tilde{g}$ by

$$\varphi_{\tilde{g}(u)}(x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \varphi_u(u) \text{ defined} \\ \Omega & \text{otherwise} \end{cases}$$

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

proof (magic): construct total computable program transformation $\tilde{g}$ by

$$\varphi_{\tilde{g}(u)}(x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \varphi_u(u) \text{ defined} \\ \Omega & \text{otherwise} \end{cases}$$

program for computation of $\tilde{g}(u)$

- decode transition function of M_u from u.
- create $code(M)$ of the following machine M
  1. save input x on free track
  2. run M_u with input u
  3. if this halts with result $e \in \mathbb{B}^*$ run universal TM U with input $e\#x$, i.e. simulate M_e on input x.
  4. if this terminates, output result of the simulation.

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

proof (magic): construct total computable program transformation $\tilde{g}$ by

$$\varphi_{\tilde{g}(u)}(x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \varphi_u(u) \text{ defined} \\ \Omega & \text{otherwise} \end{cases}$$

program for computation of $\tilde{g}(u)$

- decode transition function of M_u from u.
- create $code(M)$ of the following machine M
  1. save input x on free track
  2. run M_u with input u
  3. if this halts with result $e \in \mathbb{B}^*$ run universal TM U with input $e\#x$, i.e. simulate M_e on input x.
  4. if this terminates, output result of the simulation.

Then

$$h \circ \tilde{g} = \varphi_v$$

is a total computable function with

$$\begin{aligned} \varphi_{\tilde{g}(v)} &= \varphi_{\varphi_v(v)} && \text{(definition of } \tilde{g}) \\ &= \varphi_{h \circ \tilde{g}(v)} && \text{(definition of } \varphi_v) \\ &= \varphi_{h(\tilde{g}(v))} \end{aligned}$$

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

proof (magic): construct total computable program transformation $\tilde{g}$ by

$$\varphi_{\tilde{g}(u)}(x) = \begin{cases} \varphi_{\varphi_u(u)}(x) & \varphi_u(u) \text{ defined} \\ \Omega & \text{otherwise} \end{cases}$$

program for computation of $\tilde{g}(u)$

- decode transition function of M_u from u.
- create $code(M)$ of the following machine M
  1. save input x on free track
  2. run M_u with input u
  3. if this halts with result $e \in \mathbb{B}^*$ run universal TM U with input $e\#x$, i.e. simulate M_e on input x.
  4. if this terminates, output result of the simulation.

Then

$$h \circ \tilde{g} = \varphi_v$$

is a total computable function with

$$\begin{aligned} \varphi_{\tilde{g}(v)} &= \varphi_{\varphi_v(v)} && \text{(definition of } \tilde{g}) \\ &= \varphi_{h \circ \tilde{g}(v)} && \text{(definition of } \varphi_v) \\ &= \varphi_{h(\tilde{g}(v))} \end{aligned}$$

$$u = \tilde{g}(v) \quad \to \quad \varphi_u = \varphi_{h(u)}$$

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

Lemma 16. *There is a TM M_u such that M_u started with empty tape prints u*

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

Lemma 16. *There is a TM M_u such that M_u started with empty tape prints u*

consider program transformation h. With input u

- print goedelisation $code(D_u)$ of the following machine:
- D_u started with empty tape prints u

notation For $u \in \mathbb{B}^*$ we denote by

$$\varphi_u = f_{M_u}$$

the function computed by TM M_u.

Lemma 15. *Let $h : \mathbb{B}^* \to \mathbb{B}^*$ be any total computable function (a program transformation). Then there is $u \in \mathbb{B}^*$ such that*

$$f_{M_{h(u)}} = f_{M_u}$$

resp.

$$\varphi_{h(u)} = \varphi_u$$

i.e. the (TM's with the) original program u and the transformed program $h(u)$ compute the same function.

Lemma 16. *There is a TM M_u such that M_u started with empty tape prints u*

consider program transformation h. With input u

- print goedelisation $code(D_u)$ of the following machine:
- D_u started with empty tape prints u

By lemma 15 there is u such that

$$\begin{aligned}\varphi_u(\varepsilon) &= \varphi_{h(u)}(\varepsilon) \\ &= \text{result of } D_u \text{ started with empty tape} \\ &= u\end{aligned}$$