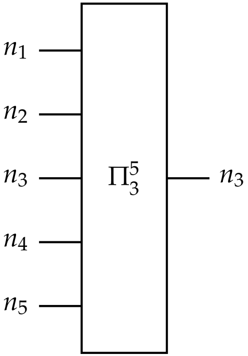
PHIL 222
Philosophical Foundations of Computer Science
Week 6, Tuesday

Oct. 1, 2024

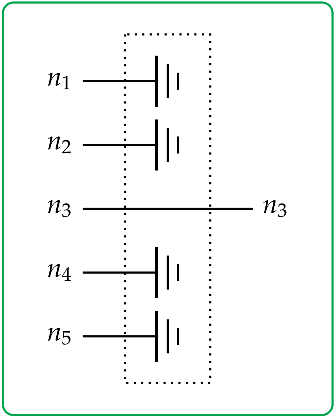
Technical Exercise 2
Review

Use the box-and-wire notation and show that
the projection $\Pi^5_3 : \mathbb{N}^5 \to \mathbb{N} :: (n_1, \ldots, n_5) \mapsto n_3$ is primitive recursive.

Use the box-and-wire notation and show that
the projection $\Pi^5_3 : \mathbb{N}^5 \to \mathbb{N} :: (n_1, \ldots, n_5) \mapsto n_3$ is primitive recursive.

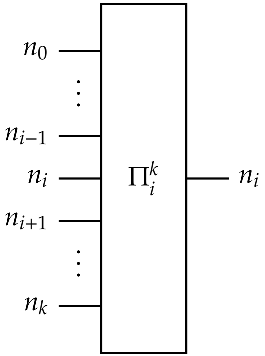


Use the box-and-wire notation and show that
the projection $\Pi^5_3 : \mathbb{N}^5 \to \mathbb{N} :: (n_1, \ldots, n_5) \mapsto n_3$ is primitive recursive.

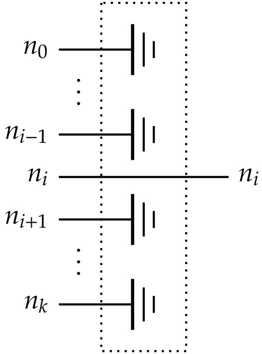


In general, for every k and $i < k$,
the projection $\Pi_i^k : \mathbb{N}^k \to \mathbb{N} :: (n_1, \ldots, n_k) \mapsto n_i$ is primitive recursive.

In general, for every k and $i < k$,
the projection $\Pi_i^k : \mathbb{N}^k \to \mathbb{N} :: (n_1, \ldots, n_k) \mapsto n_i$ is primitive recursive.



In general, for every k and $i < k$,
the projection $\Pi^k_i : \mathbb{N}^k \to \mathbb{N} :: (n_1, \ldots, n_k) \mapsto n_i$ is primitive recursive.



Prove that the following "conditional function" $\mathrm{cond} : \mathbb{N}^3 \to \mathbb{N}$

$$\mathrm{cond}(x, y, n) = \begin{cases} x & \text{if } n > 0, \\ y & \text{if } n = 0 \end{cases}$$

is primitive recursive, by (a) defining cond by primitive recursion from some f and g and (b) showing these f and g to be primitive recursive.

Prove that the following "conditional function" $\mathrm{cond} : \mathbb{N}^3 \to \mathbb{N}$

$$\mathrm{cond}(x, y, n) = \begin{cases} x & \text{if } n > 0, \\ y & \text{if } n = 0 \end{cases}$$

is primitive recursive, by (a) defining cond by primitive recursion from some f and g and (b) showing these f and g to be primitive recursive.

(a) Defining cond by primitive recursion from some f and g means

```
cond(x, y, 0) := f(x, y)
for i in (0, ..., n-1):
    cond(x, y, i+1) := g(x, y, i, cond(x, y, i))
```

Prove that the following "conditional function" $\mathrm{cond} : \mathbb{N}^3 \to \mathbb{N}$

$$\mathrm{cond}(x, y, n) = \begin{cases} x & \text{if } n > 0, \\ y & \text{if } n = 0 \end{cases}$$

is primitive recursive, by ⓐ defining cond by primitive recursion from some f and g and ⓑ showing these f and g to be primitive recursive.

ⓐ Defining cond by primitive recursion from some f and g means

```
cond(x, y, 0) := f(x, y)
for i in (0, ..., n-1):
    cond(x, y, i+1) := g(x, y, i, cond(x, y, i))
```

So take $f :: (x, y) \mapsto y$ and $g :: (x, y, i, k) \mapsto x$.

Prove that the following "conditional function" cond : $\mathbb{N}^3 \to \mathbb{N}$

$$\mathrm{cond}(x, y, n) = \begin{cases} x & \text{if } n > 0, \\ y & \text{if } n = 0 \end{cases}$$

is primitive recursive, by ⓐ defining cond by primitive recursion from some f and g and ⓑ showing these f and g to be primitive recursive.

ⓐ Defining cond by primitive recursion from some f and g means

```
cond(x, y, 0) := f(x, y)
for i in (0, ..., n-1):
    cond(x, y, i+1) := g(x, y, i, cond(x, y, i))
```

So take $f :: (x, y) \mapsto y$ and $g :: (x, y, i, k) \mapsto x$. Then

$$\mathrm{cond}(x, y, 0) = f(x, y) = y,$$
$$\mathrm{cond}(x, y, i + 1) = g(x, y, i, \mathrm{cond}(x, y, i)) = x.$$

Prove that the following "conditional function" $\mathrm{cond} : \mathbb{N}^3 \to \mathbb{N}$

$$\mathrm{cond}(x, y, n) = \begin{cases} x & \text{if } n > 0, \\ y & \text{if } n = 0 \end{cases}$$

is primitive recursive, by ⓐ defining cond by primitive recursion from some f and g and ⓑ showing these f and g to be primitive recursive.

ⓐ Defining cond by primitive recursion from some f and g means

```
cond(x, y, 0) := f(x, y)
for i in (0, ..., n-1):
    cond(x, y, i+1) := g(x, y, i, cond(x, y, i))
```

So take $f :: (x, y) \mapsto y$ and $g :: (x, y, i, k) \mapsto x$. Then

$$\mathrm{cond}(x, y, 0) = f(x, y) = y,$$

$$\mathrm{cond}(x, y, i + 1) = g(x, y, i, \mathrm{cond}(x, y, i)) = x.$$

ⓑ $f = \Pi^2_2$ and $g = \Pi^4_1$, which we have shown to be primitive recursive.

$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N)$$

$$\overline{\text{True}} := (\lambda x.\, (\lambda y.\, x)), \qquad \overline{\text{False}} := (\lambda x.\, (\lambda y.\, y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\, M)\, N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\, M)\, N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

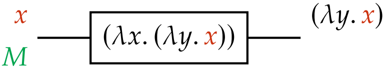
$$((\overline{\text{True}}\, M)\, N) = (((\lambda x.\, (\lambda y.\, x))\, M)\, N)$$

$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N)$$



$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N)}$$

$$\begin{array}{ccc} x & & (\lambda y.\,x) \\ & \text{---}\boxed{(\lambda x.\,(\lambda y.\,x))}\text{---} & \\ M & & (\lambda y.\,M) \end{array}$$

$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N)}$$

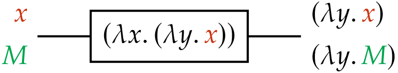
$$\begin{array}{c} x \\ M \end{array} \;\text{—}\; \boxed{(\lambda x.\,(\lambda y.\,x))} \;\text{—}\; \begin{array}{c} (\lambda y.\,x) \\ (\lambda y.\,M) \end{array}$$

$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N)}$$

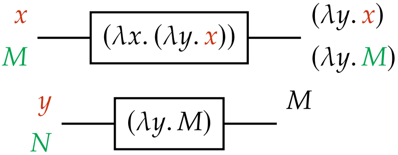


$$\overline{\mathrm{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\mathrm{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\mathrm{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\mathrm{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\mathrm{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N)}$$

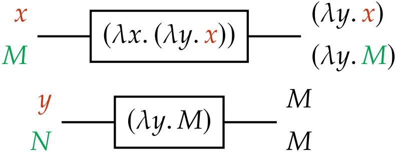


$$\overline{\text{True}} := (\lambda x.\, (\lambda y.\, x)), \qquad \overline{\text{False}} := (\lambda x.\, (\lambda y.\, y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\, M)\, N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\, M)\, N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\, M)\, N) = (((\lambda x.\, (\lambda y.\, x))\, M)\, N) \xrightarrow{\beta} ((\lambda y.\, M)\, N)}$$

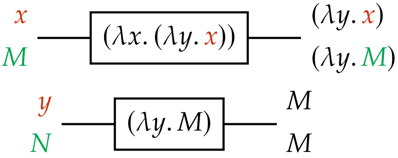


$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$

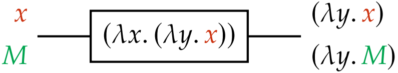


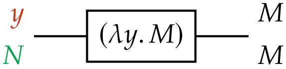
$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$





$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N)}$$

$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$

$$\begin{array}{c} x \\ M \end{array} \;—\; \boxed{(\lambda x.\,(\lambda y.\,x))} \;—\; \begin{array}{c} (\lambda y.\,x) \\ (\lambda y.\,M) \end{array}$$

$$\begin{array}{c} y \\ N \end{array} \;—\; \boxed{(\lambda y.\,M)} \;—\; \begin{array}{c} M \\ M \end{array}$$

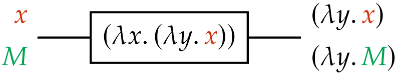
$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N)}$$

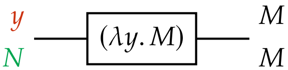
$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

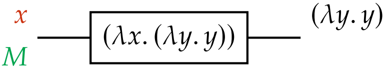
$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$





$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N)}$$



$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$

$$\begin{array}{c} x \\ M \end{array} \;—\; \boxed{(\lambda x.\,(\lambda y.\,x))} \;—\; \begin{array}{c} (\lambda y.\,x) \\ (\lambda y.\,M) \end{array}$$

$$\begin{array}{c} y \\ N \end{array} \;—\; \boxed{(\lambda y.\,M)} \;—\; \begin{array}{c} M \\ M \end{array}$$

$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N)}$$

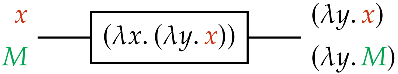
$$\begin{array}{c} x \\ M \end{array} \;—\; \boxed{(\lambda x.\,(\lambda y.\,y))} \;—\; \begin{array}{c} (\lambda y.\,y) \\ (\lambda y.\,y) \end{array}$$

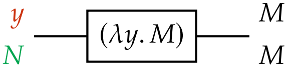
$$\overline{\text{True}} := (\lambda x.\, (\lambda y.\, x)), \qquad \overline{\text{False}} := (\lambda x.\, (\lambda y.\, y)).$$

Assuming neither x nor y occurs in M or N, show that

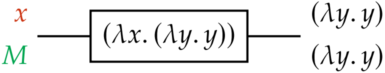
$$((\overline{\text{True}}\, M)\, N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\, M)\, N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\, M)\, N) = (((\lambda x.\, (\lambda y.\, x))\, M)\, N) \xrightarrow{\beta} ((\lambda y.\, M)\, N) \xrightarrow{\beta} M}$$





$$\boxed{((\overline{\text{False}}\, M)\, N) = (((\lambda x.\, (\lambda y.\, y))\, M)\, N) \xrightarrow{\beta} ((\lambda y.\, y)\, N)}$$

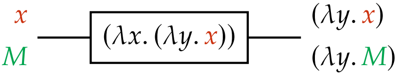


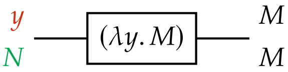
$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

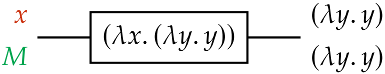
$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$





$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,y)\,N)}$$

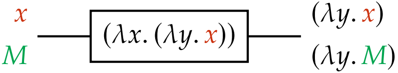


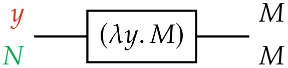
$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

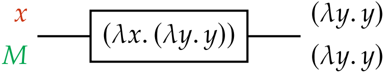
$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

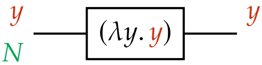
$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$





$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,y)\,N)}$$



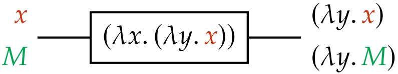


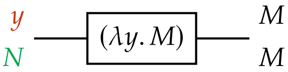
$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

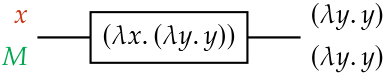
$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

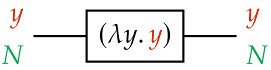
$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$





$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,y)\,N)}$$





$$\overline{\text{True}} := (\lambda x.\,(\lambda y.\,x)), \qquad \overline{\text{False}} := (\lambda x.\,(\lambda y.\,y)).$$

Assuming neither x nor y occurs in M or N, show that

$$((\overline{\text{True}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} M, \qquad ((\overline{\text{False}}\,M)\,N) \xrightarrow{\beta} \cdots \xrightarrow{\beta} N.$$

$$\boxed{((\overline{\text{True}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,x))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,M)\,N) \xrightarrow{\beta} M}$$

$$\begin{array}{c} x \\ M \end{array} \;\rule[0.5ex]{2em}{0.4pt}\; \boxed{(\lambda x.\,(\lambda y.\,x))} \;\rule[0.5ex]{2em}{0.4pt}\; \begin{array}{c} (\lambda y.\,x) \\ (\lambda y.\,M) \end{array}$$

$$\begin{array}{c} y \\ N \end{array} \;\rule[0.5ex]{2em}{0.4pt}\; \boxed{(\lambda y.\,M)} \;\rule[0.5ex]{2em}{0.4pt}\; \begin{array}{c} M \\ M \end{array}$$

$$\boxed{((\overline{\text{False}}\,M)\,N) = (((\lambda x.\,(\lambda y.\,y))\,M)\,N) \xrightarrow{\beta} ((\lambda y.\,y)\,N) \xrightarrow{\beta} N}$$

$$\begin{array}{c} x \\ M \end{array} \;\rule[0.5ex]{2em}{0.4pt}\; \boxed{(\lambda x.\,(\lambda y.\,y))} \;\rule[0.5ex]{2em}{0.4pt}\; \begin{array}{c} (\lambda y.\,y) \\ (\lambda y.\,y) \end{array}$$

$$\begin{array}{c} y \\ N \end{array} \;\rule[0.5ex]{2em}{0.4pt}\; \boxed{(\lambda y.\,y)} \;\rule[0.5ex]{2em}{0.4pt}\; \begin{array}{c} y \\ N \end{array}$$

The Church-Turing Thesis: Other Versions (cont'd)

For Ⅱ, what exactly is an algorithm?

For II, what exactly is an algorithm? We may try:

- E A process (abstractly speaking) that implements (an indendent-of-human version of) an effective method satisfying a–d.
- F A process (abstractly speaking) that satisfies (an indendent-of-human version of) i–vii.

For II, what exactly is an algorithm? We may try:

- E A process (abstractly speaking) that implements (an indendent-of-human version of) an effective method satisfying a–d.
- F A process (abstractly speaking) that satisfies (an indendent-of-human version of) i–vii.

A/E: *f can be computed by machines / algorithms that implements an effective method satisfying* a–d $\iff$ *f is Turing computable*

may be equivalent to the original Church-Turing thesis.

For II, what exactly is an algorithm? We may try:

- E A process (abstractly speaking) that implements (an indendent-of-human version of) an effective method satisfying a–d.
- F A process (abstractly speaking) that satisfies (an indendent-of-human version of) i–vii.

A / E: *f can be computed by machines / algorithms that implements an effective method satisfying* a–d $\iff$ *f is Turing computable*

may be equivalent to the original Church-Turing thesis.

B / F: *f can be computed by machines / algorithms that satisfies* i–vii $\iff$ *f is Turing computable*

may be mathematically provable.

For II, what exactly is an algorithm? We may try:

- E A process (abstractly speaking) that implements (an indendent-of-human version of) an effective method satisfying a–d.
- F A process (abstractly speaking) that satisfies (an indendent-of-human version of) i–vii.

A / E: *f can be computed by machines / algorithms that implements an effective method satisfying* a–d $\iff$ *f is Turing computable*

may be equivalent to the original Church-Turing thesis.

B / F: *f can be computed by machines / algorithms that satisfies* i–vii $\iff$ *f is Turing computable*

may be mathematically provable.

Let's investigate whether contemporary computer scientists undertand the Church-Turing thesis as A / E / B / F.

There are new paradigms of computers / algorithms that may not satisfy a–d / i–vii.

- Conway's "game of life" violates iv, vi, vii.
- Maybe we can say that quantum computers violate ii, v.

There are new paradigms of computers / algorithms that may not satisfy a–d/i–vii.

- Conway's "game of life" violates iv, vi, vii.
- Maybe we can say that quantum computers violate ii, v.

There may be an algorithm that can be peformed in such a paradigm but that no Turing machine can perform.

There are new paradigms of computers / algorithms that may not satisfy a–d/i–vii.

- Conway's "game of life" violates iv, vi, vii.
- Maybe we can say that quantum computers violate ii, v.

There may be an algorithm that can be peformed in such a paradigm but that no Turing machine can perform.

Does this mean a breakdown of the Church-Turing thesis?

There are new paradigms of computers / algorithms that may not satisfy a–d / i–vii.

- Conway's "game of life" violates iv, vi, vii.
- Maybe we can say that quantum computers violate ii, v.

There may be an algorithm that can be peformed in such a paradigm but that no Turing machine can perform.

Does this mean a breakdown of the Church-Turing thesis?

— No, because we may still have the following (mathematical) fact:

- *f can be computed by machines / algorithms in the paradigm*

  $\iff$ *f is Turing computable.*

There are new paradigms of computers / algorithms that may not satisfy a–d / i–vii.

- Conway's "game of life" violates iv, vi, vii.
- Maybe we can say that quantum computers violate ii, v.

There may be an algorithm that can be peformed in such a paradigm but that no Turing machine can perform.

Does this mean a breakdown of the Church-Turing thesis?

— No, because we may still have the following (mathematical) fact:

- *f can be computed by machines / algorithms in the paradigm*
  $\iff$ *f is Turing computable.*

One should not confuse the Church-Turing thesis with the (false) claim that every (possible / reasonable) algorithm can be performed by a Turing machine (see Copeland and Shagrir, p. 68). The thesis is about whether Turing machines can compute a given function / task f, rather than perform a given algorithm.

But let's supopse a new paradigm that violates a–d/i–vii should turn out to be capable of computing functions that Turing machines cannot.

But let's supopse a new paradigm that violates a–d / i–vii should turn out to be capable of computing functions that Turing machines cannot.

Note that it would not constitute a counterexample to

A / E / B / F: *f can be computed by machines / algorithms satisfying a–d/i–vii $\iff$ f is Turing computable*

(in fact, B / F may even be mathematically provable!). Why?

But let's supopse a new paradigm that violates a–d / i–vii should turn out to be capable of computing functions that Turing machines cannot.

Note that it would not constitute a counterexample to

A / E / B / F: *f can be computed by machines / algorithms satisfying a–d / i–vii $\iff$ f is Turing computable*

(in fact, B / F may even be mathematically provable!). Why?

— Because the new paradigm does not satisfy a–d / i–vii.

But let's supopse a new paradigm that violates a–d/i–vii should turn out to be capable of computing functions that Turing machines cannot.

Note that it would not constitute a counterexample to

A/E/B/F: *f can be computed by machines / algorithms satisfying a–d/i–vii $\iff$ f is Turing computable*

(in fact, B/F may even be mathematically provable!). Why?

— Because the new paradigm does not satisfy a–d/i–vii.

Yet, which of the following would you say?

But let's supopse a new paradigm that violates a–d / i–vii should turn out to be capable of computing functions that Turing machines cannot.

Note that it would not constitute a counterexample to

A / E / B / F: *f can be computed by machines / algorithms satisfying a–d / i–vii $\iff$ f is Turing computable*

(in fact, B / F may even be mathematically provable!). Why?

— Because the new paradigm does not satisfy a–d / i–vii.

Yet, which of the following would you say?

1. "That's no counterexample to the Church-Turing thesis. That's not the kind of computers / algorithms the thesis is about."
2. "We've found a counterexample to the Church-Turing thesis!"

But let's supopse a new paradigm that violates a–d / i–vii should turn out to be capable of computing functions that Turing machines cannot.

Note that it would not constitute a counterexample to

A / E / B / F: *f can be computed by machines / algorithms satisfying a–d / i–vii $\iff$ f is Turing computable*

(in fact, B / F may even be mathematically provable!). Why?

— Because the new paradigm does not satisfy a–d / i–vii.

Yet, which of the following would you say?

1. "That's no counterexample to the Church-Turing thesis. That's not the kind of computers / algorithms the thesis is about."
2. "We've found a counterexample to the Church-Turing thesis!"

Usual computer scientists would go 2, which means that they do not understand the Church-Turing thesis as A / E / B / F.

But let's supopse a new paradigm that violates a–d/i–vii should turn out to be capable of computing functions that Turing machines cannot.

Note that it would not constitute a counterexample to

A/E/B/F: *f can be computed by machines / algorithms satisfying a–d/i–vii $\iff$ f is Turing computable*

(in fact, B/F may even be mathematically provable!). Why?

— Because the new paradigm does not satisfy a–d/i–vii.

Yet, which of the following would you say?

1. "That's no counterexample to the Church-Turing thesis. That's not the kind of computers / algorithms the thesis is about."
2. "We've found a counterexample to the Church-Turing thesis!"

Usual computer scientists would go 2, which means that they do not understand the Church-Turing thesis as A/E/B/F.

Indeed, against any modification of A/E/B/F that replaces a–d/i–vii, you may devise a similar argument!

f is Turing computable, λ-definable, partial recursive

CTT-O

i–vii

f can be achieved by an effective method a–d

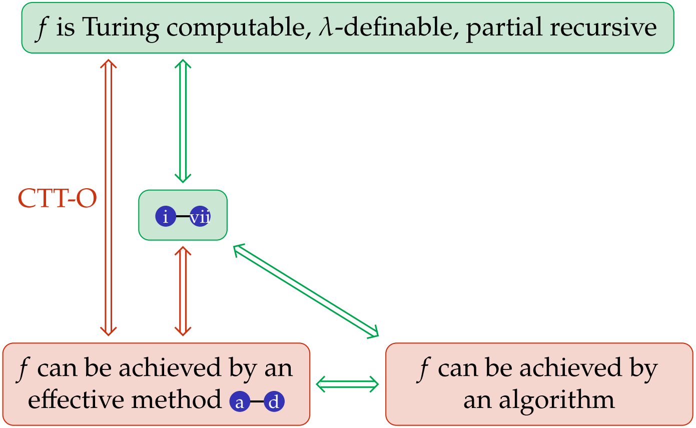
f is Turing computable, λ-definable, partial recursive

CTT-O

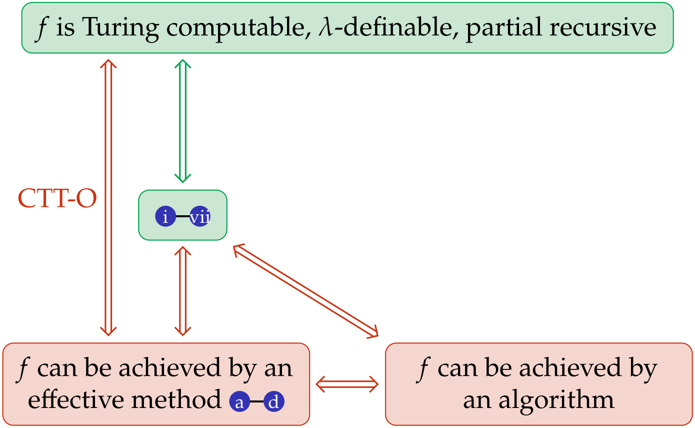
i–vii

f can be achieved by an effective method a–d

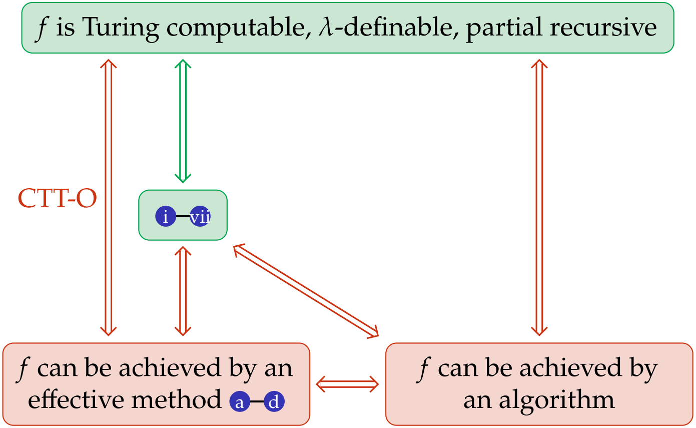
f can be achieved by an algorithm

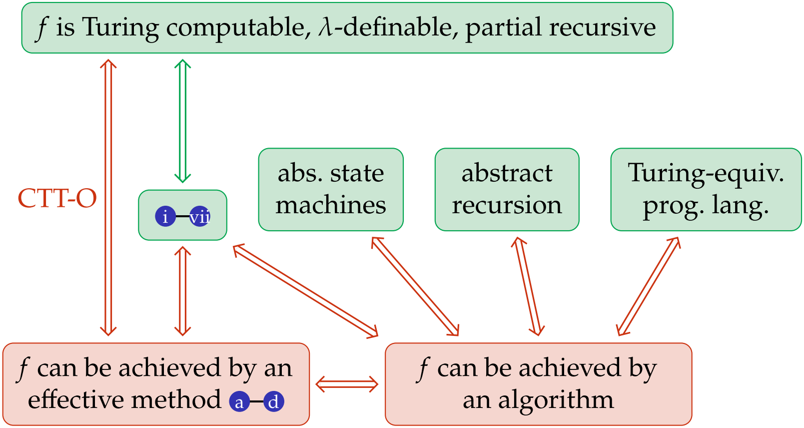


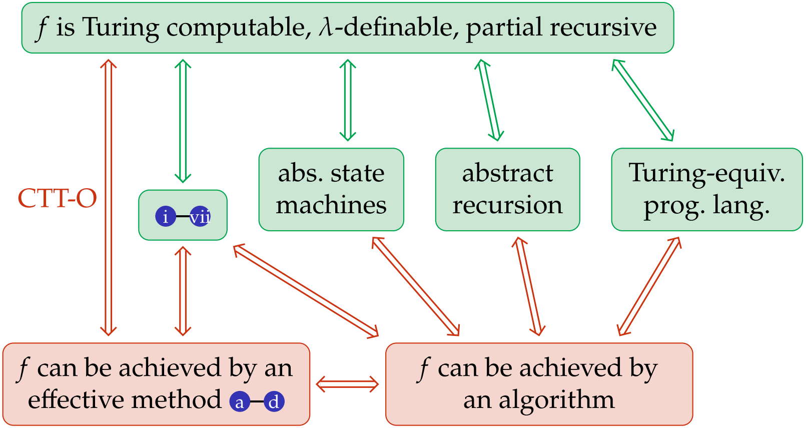
f is Turing computable, λ-definable, partial recursive
CTT-O
i—vii
f can be achieved by an effective method a—d
f can be achieved by an algorithm

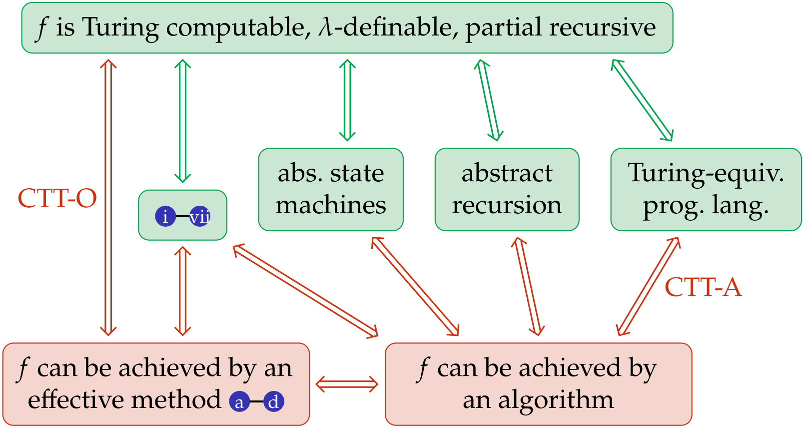


f is Turing computable, λ-definable, partial recursive
CTT-O
i–vii
f can be achieved by an effective method a–d
f can be achieved by an algorithm


f is Turing computable, λ-definable, partial recursive
CTT-O
i—vii
f can be achieved by an effective method a—d
f can be achieved by an algorithm


f is Turing computable, λ-definable, partial recursive
CTT-O
i—vii
abs. state machines
abstract recursion
Turing-equiv. prog. lang.
f can be achieved by an effective method a—d
f can be achieved by an algorithm


f is Turing computable, λ-definable, partial recursive
CTT-O
i—vii
abs. state machines
abstract recursion
Turing-equiv. prog. lang.
f can be achieved by an effective method a—d
f can be achieved by an algorithm



f is Turing computable, λ-definable, partial recursive
CTT-O
i–vii
abs. state machines
abstract recursion
Turing-equiv. prog. lang.
CTT-A
f can be achieved by an effective method a–d
f can be achieved by an algorithm

The Church-Turing Thesis: What the Thesis Does Not Say

D: *f can be generated by any machine that is conceivable regardless of the physical laws* $\iff$ *f is Turing computable*

D: *f can be generated by any machine that is conceivable regardless of the physical laws $\iff$ f is Turing computable*

is false (though it depends on "conceivable").

D: *f can be generated by any machine that is conceivable regardless of the physical laws* $\iff$ *f is Turing computable*

is false (though it depends on “conceivable”).

- “Extended Turing machines” are not physically implementable, but can store any real numbers in squares on the tape, and can compute Turing uncomputable functions.

D: *f can be generated by any machine that is conceivable regardless of the physical laws* $\iff$ *f is Turing computable*

is false (though it depends on "conceivable").

- "Extended Turing machines" are not physically implementable, but can store any real numbers in squares on the tape, and can compute Turing uncomputable functions.
- "Accelerating Turing machines" are not physically implementable:

$$\begin{aligned}
1 \text{ second on step } 0,\\
1/2 \text{ second on step } 1,\\
\vdots\\
1/2^n \text{ second on step } n,\\
\vdots
\end{aligned}$$

  So they can complete infinitely many steps in 2 seconds, and can compute Turing uncomputable functions.

D: *f can be generated by any machine that is conceivable regardless of the physical laws $\iff$ f is Turing computable*

is false (though it depends on "conceivable").

- "Extended Turing machines" are not physically implementable, but can store any real numbers in squares on the tape, and can compute Turing uncomputable functions.
- "Accelerating Turing machines" are not physically implementable:

$$
\begin{aligned}
1 \text{ second on step } 0,\\
1/2 \text{ second on step } 1,\\
\vdots\\
1/2^n \text{ second on step } n,\\
\vdots
\end{aligned}
$$

  So they can complete infinitely many steps in 2 seconds, and can compute Turing uncomputable functions.

"Finite" in the criteria of effectiveness / computability is essential.

C: *f can be generated by a machine that is physically possible / implementable*
$\iff$ *f is Turing computable.*

We *may* come back to this in a few weeks (if time permits).

C: *f can be generated by a machine that is physically possible / implementable*

$\iff$ *f is Turing computable.*

We *may* come back to this in a few weeks (if time permits).

Instead let's now discuss what Copeland calls the "simulation thesis":

> *Any process that can be given a mathematical description (or that is scientifically describable or scientifically explicable) can be simulated by a Turing machine.*

C: *f can be generated by a machine that is physically possible / implementable* $\iff$ *f is Turing computable.*

We *may* come back to this in a few weeks (if time permits).

Instead let's now discuss what Copeland calls the "simulation thesis":

> *Any process that can be given a mathematical description (or that is scientifically describable or scientifically explicable) can be simulated by a Turing machine.*

Lots of philosophers of mind cite this as an established fact, but it is actually false! (Again, depending on "describable", though.)

C: *f can be generated by a machine that is physically possible / implementable $\iff$ f is Turing computable.*

We *may* come back to this in a few weeks (if time permits).

Instead let's now discuss what Copeland calls the "simulation thesis":

> *Any process that can be given a mathematical description (or that is scientifically describable or scientifically explicable) can be simulated by a Turing machine.*

Lots of philosophers of mind cite this as an established fact, but it is actually false! (Again, depending on "describable", though.)

To see that, let's recall the mathematical proof (from the final section of the Turing Machines chapter) that there is a Turing uncomputable real number.

Theorem. There are Turing uncomputable real numbers.

Theorem. There are Turing uncomputable real numbers.

Proof. By comparing the natural-number codes of Turing machines, let's enumerate, as in

$$M_0, M_1, M_2, \ldots,$$

all the Turing machines M_n that compute a real number x_n.

Theorem. There are Turing uncomputable real numbers.

Proof. By comparing the natural-number codes of Turing machines, let's enumerate, as in

$$M_0, M_1, M_2, \dots,$$

all the Turing machines M_n that compute a real number x_n.

(It may even be physically possible to line up all these machines M_n!)

Theorem. There are Turing uncomputable real numbers.

Proof. By comparing the natural-number codes of Turing machines, let's enumerate, as in

$$M_0, M_1, M_2, \ldots,$$

all the Turing machines M_n that compute a real number x_n.

(It may even be physically possible to line up all these machines M_n!)

Then write $x_{n,0}$ for the integer part of x_n and $x_{n,m}$ ($m > 0$) for the digit in the mth decimal place of x_n. Visually, we have the following table:

$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Now define a real number d by $d_m = x_{m,m} \pm 1$. Then d differs from every x_n. Therefore d is not Turing computable. $\square$

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Now define a real number d by $d_m = x_{m,m} \pm 1$. Then d differs from every x_n. Therefore d is not Turing computable. □

- But we have just described d mathematically, have we not???

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Now define a real number d by $d_m = x_{m,m} \pm 1$. Then d differs from every x_n. Therefore d is not Turing computable. $\square$

- But we have just described d mathematically, have we not???
  — Yes, we have. Therefore mathematical describability does not imply Turing computablity.

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & \cdots \\ \hline x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\ x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\ x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\ x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

Now define a real number d by $d_m = x_{m,m} \pm 1$. Then d differs from every x_n. Therefore d is not Turing computable. $\square$

- But we have just described d mathematically, have we not???
  — Yes, we have. Therefore mathematical describability does not imply Turing computablity.
- But, but, can we not design a new Turing machine that computes

$$f(m) = (\text{the output of } M_m \text{ on input } m) \pm 1 \quad ???$$

$$
\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Now define a real number d by $d_m = x_{m,m} \pm 1$. Then d differs from every x_n. Therefore d is not Turing computable. □

- But we have just described d mathematically, have we not???
  — Yes, we have. Therefore mathematical describability does not imply Turing computablity.
- But, but, can we not design a new Turing machine that computes

$$f(m) = (\text{the output of } M_m \text{ on input } m) \pm 1 \quad ???$$

— No. No Turing machine can compute such an f,

which brings us to . . .

Computability Theory

Advice

This is a very technical part of the course.

- The concepts and facts (theorems, etc.) covered here may be relevant to both midterm and final exams.
- The proofs of the facts may be tough, but do not worry too much: you will not be tested for the understanding of them, except maybe in extra-credit problems in the midterm exam.

If you find the material hard,

- *Come to see me in office hours & appointments!*

Theorem. There are Turing uncomputable real numbers.

We proved this by mathematically constructing a Turing uncomputable number d.

Question. But why can't a Turing machine compute this d by simply tracing its construction? Where would the attempt go wrong?

Theorem. There are Turing uncomputable real numbers.

We proved this by mathematically constructing a Turing uncomputable number d.

Question. But why can't a Turing machine compute this d by simply tracing its construction? Where would the attempt go wrong?

Proof of the theorem. By comparing the natural-number codes of Turing machines, let's enumerate, as in

$$M_0, M_1, M_2, \ldots,$$

all the Turing machines M_n that compute a real number x_n.

Theorem. There are Turing uncomputable real numbers.

We proved this by mathematically constructing a Turing uncomputable number d.

Question. But why can't a Turing machine compute this d by simply tracing its construction? Where would the attempt go wrong?

Proof of the theorem. By comparing the natural-number codes of Turing machines, let's enumerate, as in

$$M_0, M_1, M_2, \ldots,$$

all the Turing machines M_n that compute a real number x_n. [...]

Answer. Mathematicians have no problem saying this enumeration exists, but it is something that Turing machines can never do!

Computability theory is the mathematical theory of what functions are not Turing computable / partial recursive / λ-definable.

Computability theory is the mathematical theory of what functions are not Turing computable / partial recursive / λ-definable.

Therefore, in this part of the course, we adopt:

Terminology. We (assume the Church-Turing thesis and) use "computable" to mean "Turing computable" / "partial recursive" / "λ-definable", and "algrorithm" to mean a Turing computable one.

Computability theory is the mathematical theory of what functions are not Turing computable / partial recursive / λ-definable.

Therefore, in this part of the course, we adopt:

Terminology. We (assume the Church-Turing thesis and) use "computable" to mean "Turing computable" / "partial recursive" / "λ-definable", and "algrorithm" to mean a Turing computable one.

In addition, we also adopt:

Notation. We write U for the function computed by a universal Turing machine, and M_n for the Turing machine whose code is n, so that

$$U(n, m) = \text{the output of the Turing machine } M_n \text{ on the input } m.$$

Let’s summarize the proof that there are uncomputable numbers:

Let's summarize the proof that there are uncomputable numbers:

1. There is an enumeration

$$M_0, M_1, M_2, \ldots, M_n, \ldots$$

of Turing machines M_n that compute real numbers x_n.

Let's summarize the proof that there are uncomputable numbers:

1. There is an enumeration

$$M_0, M_1, M_2, \ldots, M_n, \ldots$$

of Turing machines M_n that compute real numbers x_n.

2. This gives a table whose rows list all the computable numbers.

$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

Let's summarize the proof that there are uncomputable numbers:

1. There is an enumeration

$$M_0, M_1, M_2, \ldots, M_n, \ldots$$

   of Turing machines M_n that compute real numbers x_n.

2. This gives a table whose rows list all the computable numbers.

$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

3. Define a new number d by shifting the numbers on the diagonal.

Let's summarize the proof that there are uncomputable numbers:

1. There is an enumeration

$$M_0, M_1, M_2, \ldots, M_n, \ldots$$

of Turing machines M_n that compute real numbers x_n.

2. This gives a table whose rows list all the computable numbers.

$$\begin{array}{c|ccccc}
 & 0 & 1 & 2 & 3 & \cdots \\
\hline
x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\
x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\
x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\
x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}$$

3. Define a new number d by shifting the numbers on the diagonal.

4. d differs from every row of the table, and hence is uncomputable.

Let's summarize the proof that there are uncomputable numbers:

1. There is an enumeration

$$M_0, M_1, M_2, \ldots, M_n, \ldots$$

of Turing machines M_n that compute real numbers x_n.

2. This gives a table whose rows list all the computable numbers.

$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & \cdots \\ \hline x_0 & x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & \cdots \\ x_1 & x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & \cdots \\ x_2 & x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & \cdots \\ x_3 & x_{3,0} & x_{3,1} & x_{3,2} & x_{3,3} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{array}$$

3. Define a new number d by shifting the numbers on the diagonal.
4. d differs from every row of the table, and hence is uncomputable.
5. It follows that the enumeration above is not computable — because if it were, d would be computable, too.

But what exactly does it mean for there to be a computable enumeration of *such-and-such* Turing machines?

But what exactly does it mean for there to be a computable enumeration of *such-and-such* Turing machines?

Because our models of computation concern functions of numbers,

- instead of Turing machines, we must enumerate their ID numbers;
- we must represent an enumeration by a function.

But what exactly does it mean for there to be a computable enumeration of *such-and-such* Turing machines?

Because our models of computation concern functions of numbers,

- instead of Turing machines, we must enumerate their ID numbers;
- we must represent an enumeration by a function.

Definition. We say that a function $f : \mathbb{N} \to \mathbb{N}$ enumerates (all the) *such-and-such* Turing machines, as in

$$f(0), f(1), f(2), \ldots, f(n), \ldots,$$

if f is a total function s.th.

- every $f(n)$ is the ID of a *such-and-such* Turing machine,
- every *such-and-such* Turing machine has its ID appear as some $f(n)$.

But what exactly does it mean for there to be a computable enumeration of *such-and-such* Turing machines?

Because our models of computation concern functions of numbers,

- instead of Turing machines, we must enumerate their ID numbers;
- we must represent an enumeration by a function.

Definition. We say that a function $f : \mathbb{N} \to \mathbb{N}$ enumerates (all the) *such-and-such* Turing machines, as in

$$f(0), f(1), f(2), \ldots, f(n), \ldots,$$

if f is a total function s.th.

- every $f(n)$ is the ID of a *such-and-such* Turing machine,
- every *such-and-such* Turing machine has its ID appear as some $f(n)$.

Then, by a computable enumeration of *such-and-such* Turing machines, we mean a computable function f enumerating them.