Decidable and Undecidable Problems

Idea of Reduction

- Assume we have an arbitrary problem and we don't know whether it is decidable

Idea of Reduction

- Assume we have an arbitrary problem and we don't know whether it is decidable
- Assume further we are able to find a transformation from HALT to our new problem

Idea of Reduction

- Assume we have an arbitrary problem and we don't know whether it is decidable
- Assume further we are able to find a transformation from HALT to our new problem
- Is our problem decidable or undecidable?

Idea of Reduction

- Assume we have an arbitrary problem and we don't know whether it is decidable
- Assume further we are able to find a transformation from HALT to our new problem
- Is our problem decidable or undecidable?
- Notice (important): known problem $\rightarrow$ unknown problem

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \mid \varepsilon \in L(\mathscr{A})\}$

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \mid \varepsilon \in L(\mathscr{A})\}$

- Is E decidable or not?

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \mid \varepsilon \in L(\mathscr{A})\}$

- Is E decidable or not?
- consider an instance of HALT: $\langle \mathscr{A}, x \rangle$

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \,|\, \varepsilon \in L(\mathscr{A})\}$

- Is E decidable or not?
- consider an instance of HALT: $\langle \mathscr{A}, x \rangle$
- define $\mathscr{A}'$ on input y

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \,|\, \varepsilon \in L(\mathscr{A})\}$

- Is E decidable or not?
- consider an instance of HALT: $\langle \mathscr{A}, x \rangle$
- define $\mathscr{A}'$ on input y
    1. erase y

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \,|\, \varepsilon \in L(\mathscr{A})\}$

- Is E decidable or not?
- consider an instance of HALT: $\langle \mathscr{A}, x \rangle$
- define $\mathscr{A}'$ on input y
  1. erase y
  2. write x

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \,|\, \varepsilon \in L(\mathscr{A})\}$

- Is E decidable or not?
- consider an instance of HALT: $\langle \mathscr{A}, x \rangle$
- define $\mathscr{A}'$ on input y
  1. erase y
  2. write x
  3. run $\mathscr{A}$ on x

Example for Reduction

Definition

$E = \{\langle \mathscr{A} \rangle \mid \varepsilon \in L(\mathscr{A})\}$

- Is E decidable or not?
- consider an instance of HALT: $\langle \mathscr{A}, x \rangle$
- define $\mathscr{A}'$ on input y
  1. erase y
  2. write x
  3. run $\mathscr{A}$ on x
  4. accept y if $\mathscr{A}$ halts on x

Cont. Decidability of E

- $L(\mathscr{A}') = \begin{cases} \Sigma^* & \text{if } \mathscr{A} \text{ halts on } x, \\ \emptyset & \text{otherwise} \end{cases}$

Cont. Decidability of E

- $L(\mathscr{A}') = \begin{cases} \Sigma^* & \text{if } \mathscr{A} \text{ halts on } x, \\ \emptyset & \text{otherwise} \end{cases}$
- Deciding HALT: Suppose E decidable

Cont. Decidability of E

- $L(\mathscr{A}') = \begin{cases} \Sigma^* & \text{if } \mathscr{A} \text{ halts on } x, \\ \emptyset & \text{otherwise} \end{cases}$
- Deciding HALT: Suppose E decidable
  - construct $\mathscr{A}'$

Cont. Decidability of E

- $L(\mathscr{A}') = \begin{cases} \Sigma^* & \text{if } \mathscr{A} \text{ halts on } x, \\ \emptyset & \text{otherwise} \end{cases}$
- Deciding HALT: Suppose E decidable
  - construct $\mathscr{A}'$
  - decide whether $\mathscr{A}'$ accepts ε

Cont. Decidability of E

- $L(\mathscr{A}') = \begin{cases} \Sigma^* & \text{if } \mathscr{A} \text{ halts on } x, \\ \emptyset & \text{otherwise} \end{cases}$
- Deciding HALT: Suppose E decidable
  - construct $\mathscr{A}'$
  - decide whether $\mathscr{A}'$ accepts ε
  - $\Rightarrow L(\mathscr{A}') \neq \emptyset \Rightarrow \mathscr{A}$ halts (Contradiction)

Undecidable Problems

- $\mathrm{Reg}_a = \{\langle \mathscr{A} \rangle \mid \mathscr{A} \text{ accepts regular set}\}$
- $\mathrm{CFL}_a = \{\langle \mathscr{A} \rangle \mid \mathscr{A} \text{ accepts CFL}\}$
- $\mathrm{Rec}_a = \{\langle \mathscr{A} \rangle \mid \mathscr{A} \text{ accepts recursive set}\}$

Reduction

How to decide Undecidability?

1. Diagonalisation (we have proven the correctness of this strategy)

How to decide Undecidability?

1. Diagonalisation (we have proven the correctness of this strategy)
2. Reduction (we saw an example)
    - Idea: reducing known undecidable problem to new one

Effectively Computable

Definition

function f effectively computable iff f computable by total Turing machine that outputs $f(x)$ on input x

Effectively Computable

Definition

function f effectively computable iff f computable by total Turing machine that outputs $f(x)$ on input x

Examples:

- addition, multiplication
- prime numbers

Reduction

Definition

$A \subseteq \Sigma^*, B \subseteq \Delta^*$

$\sigma : \Sigma^* \longrightarrow \Delta^*$ reduction of A to B iff

- σ total, effectively computable
- $\forall x \in \Sigma^* : \; x \in A \Leftrightarrow \sigma(x) \in B$

Reduction

Definition

$A \subseteq \Sigma^*, B \subseteq \Delta^*$

$\sigma : \Sigma^* \longrightarrow \Delta^*$ reduction of A to B iff

- σ total, effectively computable
- $\forall x \in \Sigma^* : \ x \in A \Leftrightarrow \sigma(x) \in B$

Definition

$A \subseteq \Sigma^*, B \subseteq \Delta^*$

A reducible to B iff reduction of A to B exists ($A \leq_\sigma B$)

Reducibility and (Recursively) Enumerable

Theorem

1. $A \leq_\sigma B$, B *(recursively) enumerable* $\Rightarrow A$ *(recursively) enumerable*
2. $A \leq_\sigma B$, A *not (recursively) enumerable* $\Rightarrow B$ *not (recursively) enumerable*

Reducibility and (Recursively) Enumerable

Theorem

1. $A \leq_\sigma B$, B *(recursively) enumerable* $\Rightarrow A$ *(recursively) enumerable*
2. $A \leq_\sigma B$, A *not (recursively) enumerable* $\Rightarrow B$ *not (recursively) enumerable*

Proof:

Reducibility and (Recursively) Enumerable

Theorem

1. $A \leq_\sigma B$, B *(recursively) enumerable* $\Rightarrow A$ *(recursively) enumerable*
2. $A \leq_\sigma B$, A *not (recursively) enumerable* $\Rightarrow B$ *not (recursively) enumerable*

Proof:

- $A \leq_\sigma B$ via σ, B (recursively) enumerable

Reducibility and (Recursively) Enumerable

Theorem

1. $A \leq_\sigma B$, B *(recursively) enumerable* $\Rightarrow$ A *(recursively) enumerable*
2. $A \leq_\sigma B$, A *not (recursively) enumerable* $\Rightarrow$ B *not (recursively) enumerable*

Proof:

- $A \leq_\sigma B$ via σ, B (recursively) enumerable
- M TM with $B = L(M)$

Cont.

- $A \leq_\sigma B$, B (recursively) enumerable, $\mathscr{A}$ TM with $B = L(\mathscr{A})$

Cont.

- $A \leq_\sigma B$, B (recursively) enumerable, $\mathscr{A}$ TM with $B = L(\mathscr{A})$
- define TM N on input x

Cont.

- $A \leq_\sigma B$, B (recursively) enumerable, $\mathscr{A}$ TM with $B = L(\mathscr{A})$
- define TM N on input x
  - compute $\sigma(x)$

Cont.

- $A \leq_\sigma B$, B (recursively) enumerable, $\mathscr{A}$ TM with $B = L(\mathscr{A})$
- define TM N on input x
  - compute $\sigma(x)$
  - run $\mathscr{A}$ on $\sigma(x)$

Cont.

- $A \leq_\sigma B$, B (recursively) enumerable, $\mathscr{A}$ TM with $B = L(\mathscr{A})$
- define TM N on input x
  - compute $\sigma(x)$
  - run $\mathscr{A}$ on $\sigma(x)$
  - accept if $\mathscr{A}$ accepts

Cont.

- $A \leq_\sigma B$, B (recursively) enumerable, $\mathscr{A}$ TM with $B = L(\mathscr{A})$
- define TM N on input x
  - compute $\sigma(x)$
  - run $\mathscr{A}$ on $\sigma(x)$
  - accept if $\mathscr{A}$ accepts
- N accepts $x \Leftrightarrow \mathscr{A}$ accepts $\sigma(x) \Leftrightarrow \sigma(x) \in B \Leftrightarrow x \in A$ $\square$

FIN and $\overline{\text{FIN}}$

Definition

$\text{FIN} = \{\langle \mathscr{A} \rangle \mid |L(\mathscr{A})| < \infty\}$

FIN and $\overline{\text{FIN}}$

Definition

$\text{FIN} = \{\langle \mathscr{A} \rangle \mid |L(\mathscr{A})| < \infty\}$

Lemma

FIN *and* $\overline{\text{FIN}}$ *are not recursively enumerable.*

FIN and $\overline{\text{FIN}}$

Definition

$\text{FIN} = \{\langle \mathscr{A} \rangle \mid |L(\mathscr{A})| < \infty\}$

Lemma

FIN *and* $\overline{\text{FIN}}$ *are not recursively enumerable.*

Proof:

FIN and $\overline{\text{FIN}}$

Definition

$\text{FIN} = \{\langle \mathscr{A} \rangle \mid |L(\mathscr{A})| < \infty\}$

Lemma

FIN *and* $\overline{\text{FIN}}$ *are not recursively enumerable.*

Proof:

- Plan: Reduction from $\overline{\text{HALT}}$

FIN and $\overline{\mathrm{FIN}}$

Definition

$\mathrm{FIN} = \{\langle \mathscr{A} \rangle \mid |L(\mathscr{A})| < \infty\}$

Lemma

FIN *and* $\overline{\mathrm{FIN}}$ *are not recursively enumerable.*

Proof:

- Plan: Reduction from $\overline{\mathrm{HALT}}$
- we need: σ with $\langle \mathscr{A}, x \rangle \in \overline{\mathrm{HALT}} \Leftrightarrow \sigma(\langle \mathscr{A}, x \rangle) \in \mathrm{FIN}$

FIN and $\overline{\text{FIN}}$

Definition

$\text{FIN} = \{\langle\mathscr{A}\rangle \mid |L(\mathscr{A})| < \infty\}$

Lemma

FIN *and* $\overline{\text{FIN}}$ *are not recursively enumerable.*

Proof:

- Plan: Reduction from $\overline{\text{HALT}}$
- we need: σ with $\langle\mathscr{A}, x\rangle \in \overline{\text{HALT}} \Leftrightarrow \sigma(\langle\mathscr{A}, x\rangle) \in \text{FIN}$
- i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)

Cont.

- Reduction from $\overline{\mathrm{HALT}}$, i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)
- define $\mathscr{A}'$

Cont.

- Reduction from $\overline{\text{HALT}}$, i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)
- define $\mathscr{A}'$
    1. erase input

Cont.

- Reduction from $\overline{\text{HALT}}$, i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)
- define $\mathscr{A}'$
  1. erase input
  2. write x on tape

Cont.

- Reduction from $\overline{\text{HALT}}$, i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)
- define $\mathscr{A}'$
  1. erase input
  2. write x on tape
  3. run $\mathscr{A}$ on x

Cont.

- Reduction from $\overline{\text{HALT}}$, i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)
- define $\mathscr{A}'$
  1. erase input
  2. write x on tape
  3. run $\mathscr{A}$ on x
  4. accept if $\mathscr{A}$ halts

Cont.

- Reduction from $\overline{\text{HALT}}$, i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)
- define $\mathscr{A}'$
  1. erase input
  2. write x on tape
  3. run $\mathscr{A}$ on x
  4. accept if $\mathscr{A}$ halts
- $\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite

Cont.

- Reduction from $\overline{\text{HALT}}$, i.e. we need TM $\mathscr{A}'$ with ($\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite)
- define $\mathscr{A}'$
  1. erase input
  2. write x on tape
  3. run $\mathscr{A}$ on x
  4. accept if $\mathscr{A}$ halts
- $\mathscr{A}$ does not halt on $x \Leftrightarrow L(\mathscr{A}')$ finite
- σ total, effectively computable since
  - $\sigma(\langle \mathscr{A}, x \rangle) = \mathscr{A}$ works

Rice's Theorem

Non-trivial Properties of Recursively Enumerable Sets

Definition

$\mathscr{R}$ set of all recursively enumerable sets over Σ

- $P : \mathscr{R} \to \{\texttt{true}, \texttt{false}\}$ non-trivial property iff P is surjective

(the property is neither universally true nor false)

Non-trivial Properties of Recursively Enumerable Sets

Definition

$\mathscr{R}$ set of all recursively enumerable sets over Σ

- $P : \mathscr{R} \to \{\texttt{true}, \texttt{false}\}$ non-trivial property iff P is surjective

(the property is neither universally true nor false)

Examples: finiteness, regularity, context-freedom, completeness

Rice's Theorem

Theorem (Rice)

Every non-trivial property of the recursively enumerable sets is undecidable.

Rice's Theorem

Theorem (Rice)

Every non-trivial property of the recursively enumerable sets is undecidable.

Proof:

- P non-trivial property (w.l.o.g. $P(\emptyset) = \texttt{false}$)

Rice's Theorem

Theorem (Rice)

Every non-trivial property of the recursively enumerable sets is undecidable.

Proof:

- P non-trivial property (w.l.o.g. $P(\emptyset) = \texttt{false}$)
- P non-trivial $\Rightarrow \exists A \in \mathcal{R} : P(A) = \texttt{true}$

Rice's Theorem

Theorem (Rice)

Every non-trivial property of the recursively enumerable sets is undecidable.

Proof:

- P non-trivial property (w.l.o.g. $P(\emptyset) = \texttt{false}$)
- P non-trivial $\Rightarrow \exists A \in \mathscr{R} : P(A) = \texttt{true}$
- K TM accepting A

Rice's Theorem

Theorem (Rice)

Every non-trivial property of the recursively enumerable sets is undecidable.

Proof:

- P non-trivial property (w.l.o.g. $P(\emptyset) = \texttt{false}$)
- P non-trivial $\Rightarrow \exists A \in \mathcal{R} : P(A) = \texttt{true}$
- K TM accepting A
- Plan: Reduction of HALT to $\{M | P(L(M)) = \texttt{true}\}$

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \mathtt{true}\}$

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \mathtt{true}\}$
- $M' = \sigma(\langle M, x\rangle)$ with input y

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \texttt{true}\}$
- $M' = \sigma(\langle M, x \rangle)$ with input y
    1. save y on an empty tape

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \texttt{true}\}$
- $M' = \sigma(\langle M, x \rangle)$ with input y
  1. save y on an empty tape
  2. write x on a tape

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \texttt{true}\}$
- $M' = \sigma(\langle M, x \rangle)$ with input y
    1. save y on an empty tape
    2. write x on a tape
    3. run M on x

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \texttt{true}\}$
- $M' = \sigma(\langle M, x \rangle)$ with input y
  1. save y on an empty tape
  2. write x on a tape
  3. run M on x
  4. M halts on $x \Rightarrow$ run K on y and accept iff K accepts

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \texttt{true}\}$
- $M' = \sigma(\langle M, x\rangle)$ with input y
  1. save y on an empty tape
  2. write x on a tape
  3. run M on x
  4. M halts on $x \Rightarrow$ run K on y and accept iff K accepts
- M halts on $x \Rightarrow L(M') = A \Rightarrow P(L(M')) = P(A) = \texttt{true}$

Cont. Proof Rice's Theorem

- Reduction: HALT to $\{M | P(L(M)) = \texttt{true}\}$
- $M' = \sigma(\langle M, x \rangle)$ with input y
  1. save y on an empty tape
  2. write x on a tape
  3. run M on x
  4. M halts on $x \Rightarrow$ run K on y and accept iff K accepts
- M halts on $x \Rightarrow L(M') = A \Rightarrow P(L(M')) = P(A) = \texttt{true}$
- M loops on $x \Rightarrow L(M') = \emptyset \Rightarrow P(L(M')) = P(\emptyset) = \texttt{false}$

Monotone Properties

Definition

assume: $\texttt{false} \leq \texttt{true}$

$P : \mathscr{R} \longrightarrow \{\texttt{false}, \texttt{true}\}$ monotone iff

$$\forall A, B \in \mathscr{R} : A \subseteq B \Rightarrow P(A) \leq P(B).$$

Monotone Properties

Definition

assume: $\texttt{false} \leq \texttt{true}$

$P : \mathscr{R} \to \{\texttt{false}, \texttt{true}\}$ monotone iff

$$\forall A, B \in \mathscr{R} : A \subseteq B \Rightarrow P(A) \leq P(B).$$

Examples:

- infinity, equality to Σ^* are monotone

Monotone Properties

Definition

assume: $\texttt{false} \leq \texttt{true}$

$P : \mathscr{R} \to \{\texttt{false}, \texttt{true}\}$ monotone iff

$$\forall A, B \in \mathscr{R} : \ A \subseteq B \Rightarrow P(A) \leq P(B).$$

Examples:

- infinity, equality to Σ^* are monotone
- finiteness, emptyness not

Rice's Theorem Part II

Theorem

No non-monotone propety of recursively enumerable sets is semidecidable

(i.e. P non-monotone property $\Rightarrow T_P = \{M \mid P(L(M)) = \texttt{true}\}$ not recusively enumerable)

Rice's Theorem Part II

Theorem

No non-monotone propety of recursively enumerable sets is semidecidable
(i.e. P non-monotone property $\Rightarrow T_P = \{M \mid P(L(M)) = \texttt{true}\}$ not recusively enumerable)

we omit the proof

Undecidable Problems about CFLs

Emptyness Problem for CFLs

- decidable or undecidable?

Emptyness Problem for CFLs

- decidable or undecidable?
- Pumping-Lemma $\Rightarrow$ if CFL not empty, then it contains short word

Emptyness Problem for CFLs

- decidable or undecidable?
- Pumping-Lemma $\Rightarrow$ if CFL not empty, then it contains short word
- CYK-Algo $\Rightarrow$ test all short words

Emptyness Problem for CFLs

- decidable or undecidable?
- Pumping-Lemma $\Rightarrow$ if CFL not empty, then it contains short word
- CYK-Algo $\Rightarrow$ test all short words
- Emptyness problem for CFLs decidable (procedure not nice)

Backward-Chaining

better algorithm for deciding $\text{EMPTY}_{\text{CFL}}$ by considering the grammar w.l.o.g. in CNF

1. mark all terminal symbols

Backward-Chaining

better algorithm for deciding $\text{EMPTY}_{\text{CFL}}$ by considering the grammar w.l.o.g. in CNF

1. mark all terminal symbols
2. `if` right hand side of production is completely marked `then` mark left-hand side and all occurrences of this left-hand side in the right-hand side

Backward-Chaining

better algorithm for deciding $\text{EMPTY}_{\text{CFL}}$ by considering the grammar w.l.o.g. in CNF

1. mark all terminal symbols
2. `if` right hand side of production is completely marked `then` mark left-hand side and all occurrences of this left-hand side in the right-hand side
3. `if` nothing newly marked
   `then return false` if S is marked, `true` otherwise
   `else goto` (2)

Chomsky Hierarchy in Grammars

Type 0	Type 1	Type 2	Type 3
unrestricted	context-sensitive	context-free	right-linear
TM		PDA	DFA/NFA

Chomsky Hierarchy in Grammars

Type 0	Type 1	Type 2	Type 3
unrestricted	context-sensitive	context-free	right-linear
TM		PDA	DFA/NFA

- context-sensitive $|LHS| \leq |RHS|$

Chomsky Hierarchy in Grammars

Type 0	Type 1	Type 2	Type 3
unrestricted	context-sensitive	context-free	right-linear
TM		PDA	DFA/NFA

- context-sensitive $|\mathrm{LHS}| \leq |\mathrm{RHS}|$
- right-linear $A \longrightarrow aB$

ToC

While-Programs

Syntax of While-Programs

$\mathrm{Var} = \{x, y, \dots\}$ ranging over $\mathbb{N}$ and $\circ \in \{<, >, \leq, \geq, =, \neq\}$

1. simple assignment $x := 0$, $x := y + 1$, $x := y$
2. sequential composition $p; q$
3. conditional $\texttt{if}\ x \circ y\ \texttt{then}\ p\ \texttt{else}\ q$
4. for loop $\texttt{for}\ y\ \texttt{do}\ p$
5. while loop $\texttt{for}\ x \circ y\ \texttt{do}\ \ p$

Semantics of While-Programs

Definition

- σ **state/environment**: $\sigma : \mathrm{Var} \longrightarrow \mathbb{N}$
- Env set of all environments
- $\sigma[x \leftarrow a](x) = a$ and $\sigma[x \leftarrow a](y) = \sigma(y)$ for $y \neq x$

Cont: Inductive Definition of Semantics

- $[\![x := 0]\!]_\sigma = \sigma[x \leftarrow 0]$

Cont: Inductive Definition of Semantics

- $[\![x := 0]\!]_\sigma = \sigma[x \leftarrow 0]$
- $[\![x := y]\!]_\sigma = \sigma[x \leftarrow \sigma(y)]$

Cont: Inductive Definition of Semantics

- $[\![x := 0]\!]_\sigma = \sigma[x \leftarrow 0]$
- $[\![x := y]\!]_\sigma = \sigma[x \leftarrow \sigma(y)]$
- $[\![x := y + 1]\!]_\sigma = \sigma[x \leftarrow \sigma(y) + 1]$

Cont: Inductive Definition of Semantics

- $[\![x := 0]\!]_\sigma = \sigma[x \leftarrow 0]$
- $[\![x := y]\!]_\sigma = \sigma[x \leftarrow \sigma(y)]$
- $[\![x := y + 1]\!]_\sigma = \sigma[x \leftarrow \sigma(y) + 1]$
- $[\![p; q]\!]_\sigma = [\![q]\!]_\sigma([\![p]\!]_\sigma)$

Cont: Inductive Definition of Semantics

- $[\![x := 0]\!]_\sigma = \sigma[x \leftarrow 0]$
- $[\![x := y]\!]_\sigma = \sigma[x \leftarrow \sigma(y)]$
- $[\![x := y + 1]\!]_\sigma = \sigma[x \leftarrow \sigma(y) + 1]$
- $[\![p; q]\!]_\sigma = [\![q]\!]_\sigma([\![p]\!]_\sigma)$
- $[\![\texttt{if}\ x \circ y\ \texttt{then}\ p\ \texttt{else}\ q]\!]_\sigma = \begin{cases} [\![p]\!]_\sigma & \text{if } \sigma(x) \circ \sigma(y), \\ [\![q]\!]_\sigma & \text{otherwise.} \end{cases}$

Cont: Inductive Definition of Semantics

- $\llbracket x := 0 \rrbracket_\sigma = \sigma[x \leftarrow 0]$
- $\llbracket x := y \rrbracket_\sigma = \sigma[x \leftarrow \sigma(y)]$
- $\llbracket x := y + 1 \rrbracket_\sigma = \sigma[x \leftarrow \sigma(y) + 1]$
- $\llbracket p; q \rrbracket_\sigma = \llbracket q \rrbracket_\sigma(\llbracket p \rrbracket_\sigma)$
- $\llbracket \texttt{if}\ x \circ y\ \texttt{then}\ p\ \texttt{else}\ q \rrbracket_\sigma = \begin{cases} \llbracket p \rrbracket_\sigma & \text{if } \sigma(x) \circ \sigma(y), \\ \llbracket q \rrbracket_\sigma & \text{otherwise.} \end{cases}$
- $\llbracket \texttt{for}\ y\ \texttt{do}\ p \rrbracket_\sigma = \llbracket p \rrbracket_\sigma^{\sigma(y)}$

Cont: Inductive Definition of Semantics

- $[\![x := 0]\!]_\sigma = \sigma[x \leftarrow 0]$
- $[\![x := y]\!]_\sigma = \sigma[x \leftarrow \sigma(y)]$
- $[\![x := y + 1]\!]_\sigma = \sigma[x \leftarrow \sigma(y) + 1]$
- $[\![p; q]\!]_\sigma = [\![q]\!]_\sigma([\![p]\!]_\sigma)$
- $[\![\texttt{if } x \circ y \texttt{ then } p \texttt{ else } q]\!]_\sigma = \begin{cases} [\![p]\!]_\sigma & \text{if } \sigma(x) \circ \sigma(y), \\ [\![q]\!]_\sigma & \text{otherwise.} \end{cases}$
- $[\![\texttt{for } y \texttt{ do } p]\!]_\sigma = [\![p]\!]_\sigma^{\sigma(y)}$
- $[\![\texttt{while } x \circ y \texttt{ do } p]\!]_\sigma =$
$$\begin{cases} [\![p]\!]_\sigma^n & \text{if } n = \min\{k \in \mathbb{N} \mid [\![p]\!]_\sigma^k \text{ defined and} \\ & \neg([\![p]\!]_\sigma^k(x) \circ [\![p]\!]_\sigma^k(y))\} \\ \text{undefined} & \text{otherwise.} \end{cases}$$

μ-recursion and while-programmes

Theorem

1. *μ-recursive functions are as powerful as while-programmes and v.v.*
2. *primitive recursive functions are as powerful as for-programmes and v.v.*