Theoretical Computer Science
Decidability and Computability

Jonas Hübotter

Outline

- Turing Machine (TM)
  - Encoding
  - k-tape TM
- Computability
- Decidability
  - Problem
  - Reduction
  - Decidability
  - Theorem of Rice
  - Semi-Decidability
  - Theorem of Rice-Shapiro
- Computation Models
- Ackermann Function

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) tape alphabet Γ

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) tape alphabet Γ;
- a (partial) transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) tape alphabet Γ;
- a (partial) transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$;
- an initial state $q_0 \in Q$

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) tape alphabet Γ;
- a (partial) transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$;
- an initial state $q_0 \in Q$;
- an empty tape element $\Box \in \Gamma \setminus \Sigma$

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) tape alphabet Γ;
- a (partial) transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$;
- an initial state $q_0 \in Q$;
- an empty tape element $\Box \in \Gamma \setminus \Sigma$; and
- a set of terminal (accepting) states $F \subseteq Q$.

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) tape alphabet Γ;
- a (partial) transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$;
- an initial state $q_0 \in Q$;
- an empty tape element $\Box \in \Gamma \setminus \Sigma$; and
- a set of terminal (accepting) states $F \subseteq Q$.

We assume $\delta(q, a) = \perp$ (is undefined) for any $q \in F$, i.e. as soon as we reach a final state the TM halts.

TM

Definition 1

A Turing machine (TM) $M = (Q, \Sigma, \Gamma, \delta, q_0, \Box, F)$ consists of

- a finite set of states Q;
- a (finite) input alphabet Σ;
- a (finite) tape alphabet Γ;
- a (partial) transition function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R, N\}$;
- an initial state $q_0 \in Q$;
- an empty tape element $\Box \in \Gamma \setminus \Sigma$; and
- a set of terminal (accepting) states $F \subseteq Q$.

We assume $\delta(q, a) = \bot$ (is undefined) for any $q \in F$, i.e. as soon as we reach a final state the TM halts.
Graphically, transitions are denoted as $\alpha/\beta, \xi$ where $\alpha \in \Gamma$ is the current tape element which is replaced by $\beta \in \Gamma$ and the head moves in the direction $\xi \in \{L, R, N\}$.

TM

A Turing machine can be interpreted as a read-write-head operating on an infinite tape initialized with $\square$. L, R, and N denote the movement of the head on the tape in the direction left, right, and none, respectively.

TM

A Turing machine can be interpreted as a read-write-head operating on an infinite tape initialized with $\square$. L, R, and N denote the movement of the head on the tape in the direction left, right, and none, respectively.

Definition 2

A nondeterministic TM has the transition function $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R,N\}}$, similarly to nondeterministic PDAs.

TM

A Turing machine can be interpreted as a read-write-head operating on an infinite tape initialized with $\square$. L, R, and N denote the movement of the head on the tape in the direction left, right, and none, respectively.

Definition 2

A nondeterministic TM has the transition function $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R,N\}}$, similarly to nondeterministic PDAs.

Theorem 3

For every nondeterministic TM N there exists a deterministic TM M such that $L(N) = L(M)$.

TM

A Turing machine can be interpreted as a read-write-head operating on an infinite tape initialized with $\Box$. L, R, and N denote the movement of the head on the tape in the direction left, right, and none, respectively.

Definition 2

A nondeterministic TM has the transition function $\delta : Q \times \Gamma \to 2^{Q \times \Gamma \times \{L,R,N\}}$, similarly to nondeterministic PDAs.

Theorem 3

For every nondeterministic TM N there exists a deterministic TM M such that $L(N) = L(M)$.

Idea: M uses breadth-first search to emulate N (see *dovetailing*).

TM

Definition 4

The configuration of a TM M is a triple (α, q, β) where $q \in Q$ is its state, $\alpha \in \Gamma^*$ is the tape content left-to-right up to the position of the head, and $\beta \in \Gamma^*$ is the tape content left-to-right from the element at the position of the head.

TM

Definition 4

The configuration of a TM M is a triple (α, q, β) where $q \in Q$ is its state, $\alpha \in \Gamma^*$ is the tape content left-to-right up to the position of the head, and $\beta \in \Gamma^*$ is the tape content left-to-right from the element at the position of the head.
Given configuration $(\alpha, q\beta)$, M can be graphically represented as $\cdots \Box \alpha \beta \Box \cdots$ where M is in state q and its head is at the leftmost symbol of β.

TM

Definition 4

The configuration of a TM M is a triple (α, q, β) where $q \in Q$ is its state, $\alpha \in \Gamma^*$ is the tape content left-to-right up to the position of the head, and $\beta \in \Gamma^*$ is the tape content left-to-right from the element at the position of the head.
Given configuration $(\alpha, q\beta)$, M can be graphically represented as $\cdots \Box\alpha\beta\Box \cdots$ where M is in state q and its head is at the leftmost symbol of β.

The initial configuration of M on input $w \in \Sigma^*$ is (ϵ, q_0, w).

TM

Definition 4

The configuration of a TM M is a triple (α, q, β) where $q \in Q$ is its state, $\alpha \in \Gamma^*$ is the tape content left-to-right up to the position of the head, and $\beta \in \Gamma^*$ is the tape content left-to-right from the element at the position of the head.
Given configuration $(\alpha, q\beta)$, M can be graphically represented as $\cdots \Box \alpha\beta \Box \cdots$ where M is in state q and its head is at the leftmost symbol of β.

The initial configuration of M on input $w \in \Sigma^*$ is (ϵ, q_0, w).The run of a Turing machine on input w is denoted by $M[w]$.

TM

Definition 4

The configuration of a TM M is a triple (α, q, β) where $q \in Q$ is its state, $\alpha \in \Gamma^*$ is the tape content left-to-right up to the position of the head, and $\beta \in \Gamma^*$ is the tape content left-to-right from the element at the position of the head.
Given configuration $(\alpha, q\beta)$, M can be graphically represented as $\cdots \Box \alpha\beta \Box \cdots$ where M is in state q and its head is at the leftmost symbol of β.

The initial configuration of M on input $w \in \Sigma^*$ is (ϵ, q_0, w).The run of a Turing machine on input w is denoted by $M[w]$.

Definition 5

A TM terminates when it reaches a configuration $(\alpha, q, a\beta)$ where $\delta(q, a) = \bot$ or $\delta(q, a) = \emptyset$.

Definition 4

The configuration of a TM M is a triple (α, q, β) where $q \in Q$ is its state, $\alpha \in \Gamma^*$ is the tape content left-to-right up to the position of the head, and $\beta \in \Gamma^*$ is the tape content left-to-right from the element at the position of the head.
Given configuration $(\alpha, q\beta)$, M can be graphically represented as $\cdots \Box \alpha\beta \Box \cdots$ where M is in state q and its head is at the leftmost symbol of β.

The initial configuration of M on input $w \in \Sigma^*$ is (ϵ, q_0, w).The run of a Turing machine on input w is denoted by $M[w]$.

Definition 5

A TM terminates when it reaches a configuration $(\alpha, q, a\beta)$ where $\delta(q, a) = \bot$ or $\delta(q, a) = \emptyset$. This is denoted by $M[w]\downarrow$.

TM

Definition 6

A run of a TM M is modeled as the relation $\rightarrow_M$. Given $\delta(q, \text{first}(\beta)) = (q', c, D)$

$$\alpha, q\beta) \rightarrow_M \begin{cases} (\alpha, q', c\ \text{rest}(\beta)) & D = N \\ (\alpha c, q', \text{rest}(\beta)) & D = R \\ (\text{butlast}(\alpha), q', \text{last}(\alpha)\ c\ \text{rest}(\beta)) & D = L \end{cases}$$

where for $w = w_1 \cdots w_n$, $\text{first}(w) = w_1$, $\text{rest}(w) = w_2 \cdots w_n$, $\text{last}(w) = w_n$, and $\text{butlast}(w) = w_1 \cdots w_{n-1}$.

TM

Definition 7

A TM M accepts the language

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F.\ \alpha, \beta \in \Gamma^*.\ (\epsilon, q_0, w) \to_M^* (\alpha, q, \beta)\}.$$

TM

Definition 7

A TM M accepts the language

$$L(M) = \{w \in \Sigma^* \mid \exists q \in F.\ \alpha, \beta \in \Gamma^*.\ (\epsilon, q_0, w) \rightarrow^*_M (\alpha, q, \beta)\}.$$

The languages accepted by a TM are precisely the type-0 grammars in the Chomsky-Hierarchy (i.e. semi-decidable languages).

Encoding

A TM can be encoded using words over the alphabet $\{0, 1\}$.

Encoding

A TM can be encoded using words over the alphabet $\{0, 1\}$.

Definition 8

M_w denotes the Turing machine represented by the encoding $w \in \{0, 1\}^*$.

k-tape TM

Definition 9

A k-tape TM is a TM that operates on k tapes simultaneously.

k-tape TM

Definition 9

A k-tape TM is a TM that operates on k tapes simultaneously.

Theorem 10

Every k-tape TM can be simulated by a 1-tape TM.

Turing-Computability

Definition 11

A function $f : \Sigma^* \to \Sigma^*$ (Σ is a finite set) is Turing-computable if there exists a TM M such that $\forall u, v \in \Sigma^*$

$$f(u) = v \iff \exists q \in F.\ (\epsilon, q_0, u) \to_M^* (\epsilon, q, v).$$

Turing-Computability

Definition 11

A function $f : \Sigma^* \to \Sigma^*$ (Σ is a finite set) is Turing-computable if there exists a TM M such that $\forall u, v \in \Sigma^*$

$$f(u) = v \iff \exists q \in F.\ (\epsilon, q_0, u) \to_M^* (\epsilon, q, v).$$

In particular, any TM computes a function. φ_w denotes the function computed by M_w.

Turing-Computability

Definition 11

A function $f : \Sigma^* \to \Sigma^*$ (Σ is a finite set) is Turing-computable if there exists a TM M such that $\forall u, v \in \Sigma^*$

$$f(u) = v \iff \exists q \in F.\ (\epsilon, q_0, u) \to_M^* (\epsilon, q, v).$$

In particular, any TM computes a function. φ_w denotes the function computed by M_w.

Thus, Turing-computability is a property of functions operating on discrete sets (i.e. functions implemented by a computer).

Turing-Computability

Definition 11

A function $f : \Sigma^* \to \Sigma^*$ (Σ is a finite set) is Turing-computable if there exists a TM M such that $\forall u, v \in \Sigma^*$

$$f(u) = v \iff \exists q \in F.\ (\epsilon, q_0, u) \to_M^* (\epsilon, q, v).$$

In particular, any TM computes a function. φ_w denotes the function computed by M_w.

Thus, Turing-computability is a property of functions operating on discrete sets (i.e. functions implemented by a computer).

The Church-Turing (hypo-)thesis states that any such function can be computed by a *computer* (or effective method) iff it is Turing-computable (i.e. can be computed by a Turing machine).

Problem

Definition 12

A problem is a language $A = \{x \in \Sigma^* \mid P(x)\} \subseteq \Sigma^*$ for some predicate $P : \Sigma^* \to \{0, 1\}$.

Problem

Definition 12

A problem is a language $A = \{x \in \Sigma^* \mid P(x)\} \subseteq \Sigma^*$ for some predicate $P : \Sigma^* \to \{0, 1\}$.

Given problem $A \subseteq \Sigma^*$.

Problem

Definition 12

A problem is a language $A = \{x \in \Sigma^* \mid P(x)\} \subseteq \Sigma^*$ for some predicate $P : \Sigma^* \to \{0, 1\}$.

Given problem $A \subseteq \Sigma^*$.

- x is an instance of A if $x \in \Sigma^*$.

Problem

Definition 12

A problem is a language $A = \{x \in \Sigma^* \mid P(x)\} \subseteq \Sigma^*$ for some predicate $P : \Sigma^* \to \{0, 1\}$.

Given problem $A \subseteq \Sigma^*$.

- x is an instance of A if $x \in \Sigma^*$.
- x is a solution to A if $x \in A$.

Reduction

Definition 13

A problem $A \subseteq \Sigma^*$ is reducable to a problem $B \subseteq \Gamma^*$ (denoted $A \leq B$) if there is a total and computable function $f : \Sigma^* \to \Gamma^*$ such that

$$\forall w \in \Sigma^*.\ w \in A \iff f(w) \in B.$$

Reduction

Definition 13

A problem $A \subseteq \Sigma^*$ is reducable to a problem $B \subseteq \Gamma^*$ (denoted $A \leq B$) if there is a total and computable function $f : \Sigma^* \to \Gamma^*$ such that

$$\forall w \in \Sigma^*.\ w \in A \iff f(w) \in B.$$

Example 14

To show that a function f is a valid reduction from A to B we need to prove three properties:

Reduction

Definition 13

A problem $A \subseteq \Sigma^*$ is reducable to a problem $B \subseteq \Gamma^*$ (denoted $A \leq B$) if there is a total and computable function $f : \Sigma^* \to \Gamma^*$ such that

$$\forall w \in \Sigma^*.\ w \in A \iff f(w) \in B.$$

Example 14

To show that a function f is a valid reduction from A to B we need to prove three properties:

- f is *total* on Σ^*

Reduction

Definition 13

A problem $A \subseteq \Sigma^*$ is reducable to a problem $B \subseteq \Gamma^*$ (denoted $A \leq B$) if there is a total and computable function $f : \Sigma^* \to \Gamma^*$ such that

$$\forall w \in \Sigma^*.\ w \in A \iff f(w) \in B.$$

Example 14

To show that a function f is a valid reduction from A to B we need to prove three properties:

- f is *total* on Σ^*;
- f is *computable*

Reduction

Definition 13

A problem $A \subseteq \Sigma^*$ is reducable to a problem $B \subseteq \Gamma^*$ (denoted $A \leq B$) if there is a total and computable function $f : \Sigma^* \to \Gamma^*$ such that

$$\forall w \in \Sigma^*.\ w \in A \iff f(w) \in B.$$

Example 14

To show that a function f is a valid reduction from A to B we need to prove three properties:

- f is *total* on Σ^*;
- f is *computable*; and
- f is *correct*, i.e. $\forall w \in \Sigma^*.\ w \in A \iff f(w) \in B$.

Decidability

Decidability can be interpreted as computability in the context of problems instead of functions.

Decidability

Decidability can be interpreted as computability in the context of problems instead of functions.

Definition 15

The characteristic function of a problem A is given as

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}.$$

Decidability

Decidability can be interpreted as computability in the context of problems instead of functions.

Definition 15

The characteristic function of a problem A is given as

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}.$$

Definition 16

A problem A is decidable if its characteristic function is computable.

Decidability

Decidability can be interpreted as computability in the context of problems instead of functions.

Definition 15

The characteristic function of a problem A is given as

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}.$$

Definition 16

A problem A is decidable if its characteristic function is computable.

Given a reduction $A \leq B$,

Decidability

Decidability can be interpreted as computability in the context of problems instead of functions.

Definition 15

The characteristic function of a problem A is given as

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}.$$

Definition 16

A problem A is decidable if its characteristic function is computable.

Given a reduction $A \leq B$,

- B decidable $\implies$ A decidable

Decidability

Decidability can be interpreted as computability in the context of problems instead of functions.

Definition 15

The characteristic function of a problem A is given as

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A \end{cases}.$$

Definition 16

A problem A is decidable if its characteristic function is computable.

Given a reduction $A \leq B$,

- B decidable $\implies$ A decidable; and
- A undecidable $\implies$ B undecidable.

Theorem of Rice

Theorem 17 (Theorem of Rice)

Let $\mathcal{F}$ be a set of computable functions.

Theorem of Rice

Theorem 17 (Theorem of Rice)

Let $\mathcal{F}$ be a set of computable functions.
If $\mathcal{F}$ is non-trivial, i.e. $\mathcal{F} \neq \emptyset$ and $\mathcal{F} \neq \{f \mid f \text{ computable}\}$,

Theorem of Rice

Theorem 17 (Theorem of Rice)

*Let $\mathcal{F}$ be a set of computable functions.
If $\mathcal{F}$ is non-trivial, i.e. $\mathcal{F} \neq \emptyset$ and $\mathcal{F} \neq \{f \mid f \text{ computable}\}$,
then deciding if $\varphi_w \in \mathcal{F}$ is undecidable.*

Theorem of Rice

Theorem 17 (Theorem of Rice)

Let $\mathcal{F}$ be a set of computable functions.
If $\mathcal{F}$ is non-trivial, i.e. $\mathcal{F} \neq \emptyset$ and $\mathcal{F} \neq \{f \mid f \text{ computable}\}$,
then deciding if $\varphi_w \in \mathcal{F}$ is undecidable.

In other words,

$$C_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \varphi_w \in \mathcal{F}\}$$

is undecidable.

Theorem of Rice

Example 18

When using the theorem of Rice to prove that a problem $A = \{w \in \{0,1\}^* \mid P(w)\}$ is undecidable, we must complete two steps:

Theorem of Rice

Example 18

When using the theorem of Rice to prove that a problem $A = \{w \in \{0,1\}^* \mid P(w)\}$ is undecidable, we must complete two steps:

1. construct the set of computable functions $\mathcal{F}$ that fulfill the same property P as functions φ_w whose w are in A

Theorem of Rice

Example 18

When using the theorem of Rice to prove that a problem $A = \{w \in \{0,1\}^* \mid P(w)\}$ is undecidable, we must complete two steps:

1. construct the set of computable functions $\mathcal{F}$ that fulfill the same property P as functions φ_w whose w are in A; and
2. show that $\mathcal{F}$ is non-trivial by giving an example of a computable function $g \in \mathcal{F}$ and a computable function $h \notin \mathcal{F}$.

Theorem of Rice

Example 18

When using the theorem of Rice to prove that a problem $A = \{w \in \{0,1\}^* \mid P(w)\}$ is undecidable, we must complete two steps:

1. construct the set of computable functions $\mathcal{F}$ that fulfill the same property P as functions φ_w whose w are in A; and
2. show that $\mathcal{F}$ is non-trivial by giving an example of a computable function $g \in \mathcal{F}$ and a computable function $h \notin \mathcal{F}$.

Note that for step 1, P must not depend directly on the encoding w but only on φ_w, otherwise the theorem of Rice cannot be applied.

Semi-Decidability

Definition 19

A problem A is semi-decidable if

$$\chi'_A(x) = \begin{cases} 1 & x \in A \\ \bot & x \notin A \end{cases}.$$

is computable.

Semi-Decidability

Definition 19

A problem A is semi-decidable if

$$\chi'_A(x) = \begin{cases} 1 & x \in A \\ \perp & x \notin A \end{cases}.$$

is computable.

- Given a reduction $A \leq B$, B semi-decidable $\implies$ A semi-decidable

Semi-Decidability

Definition 19

A problem A is semi-decidable if

$$\chi'_A(x) = \begin{cases} 1 & x \in A \\ \bot & x \notin A \end{cases}.$$

is computable.

- Given a reduction $A \leq B$, B semi-decidable $\implies$ A semi-decidable; and
- A decidable $\iff$ A semi-decidable and $\bar{A}$ semi-decidable.

Recursive Enumerability

Definition 20

A problem A is recursively enumerable if $A = \emptyset$ or there exists a computable function $f : \mathbb{N}_0 \to A$ such that $A = \{f(0), f(1), \dots\}$.

Recursive Enumerability

Definition 20

A problem A is recursively enumerable if $A = \emptyset$ or there exists a computable function $f : \mathbb{N}_0 \to A$ such that $A = \{f(0), f(1), \dots\}$.

Theorem 21

A problem A is semi-decidable iff A is recursively enumerable.

Theorem of Rice-Shapiro

Theorem 22 (Theorem of Rice-Shapiro)

Let $\mathcal{F}$ be a set of computable functions.

Theorem of Rice-Shapiro

Theorem 22 (Theorem of Rice-Shapiro)

Let $\mathcal{F}$ be a set of computable functions.
If $C_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \varphi_w \in \mathcal{F}\}$ is semi-decidable,

Theorem of Rice-Shapiro

Theorem 22 (Theorem of Rice-Shapiro)

Let $\mathcal{F}$ be a set of computable functions.
If $C_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \varphi_w \in \mathcal{F}\}$ is semi-decidable,
then $f \in \mathcal{F}$ iff there exists a finite and partial function $g \subseteq f$ with $f \in \mathcal{F}$.

Theorem of Rice-Shapiro

Theorem 22 (Theorem of Rice-Shapiro)

Let $\mathcal{F}$ be a set of computable functions.
If $C_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \varphi_w \in \mathcal{F}\}$ is semi-decidable,
then $f \in \mathcal{F}$ iff there exists a finite and partial function $g \subseteq f$ with $f \in \mathcal{F}$.

Often the contrapositive statement is useful:

Theorem of Rice-Shapiro

Theorem 22 (Theorem of Rice-Shapiro)

Let $\mathcal{F}$ be a set of computable functions.
If $C_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \varphi_w \in \mathcal{F}\}$ is semi-decidable,
then $f \in \mathcal{F}$ iff there exists a finite and partial function $g \subseteq f$ with $f \in \mathcal{F}$.

Often the contrapositive statement is useful:
If there exists an $f \in \mathcal{F}$ such there exists no finite and partial function $g \subseteq f$ with $g \in \mathcal{F}$

Theorem of Rice-Shapiro

Theorem 22 (Theorem of Rice-Shapiro)

Let $\mathcal{F}$ be a set of computable functions.
If $C_{\mathcal{F}} = \{w \in \{0,1\}^* \mid \varphi_w \in \mathcal{F}\}$ is semi-decidable,
then $f \in \mathcal{F}$ iff there exists a finite and partial function $g \subseteq f$ with $f \in \mathcal{F}$.

Often the contrapositive statement is useful:
If there exists an $f \in \mathcal{F}$ such there exists no finite and partial function $g \subseteq f$ with $g \in \mathcal{F}$, then $C_{\mathcal{F}}$ is not semi-decidable.

Computation Models

We have mainly focused on Turing machines to model computability. There are, however, other models for computability that are commonly used:

Computation Models

We have mainly focused on Turing machines to model computability. There are, however, other models for computability that are commonly used:

- WHILE, programs using **while** $x \neq 0$ **do** $\cdots$ **end while** and **if** $x = 0$ **then** $\cdots$ **else** $\cdots$ **end if** for control flow

Computation Models

We have mainly focused on Turing machines to model computability. There are, however, other models for computability that are commonly used:

- WHILE, programs using **while** $x \neq 0$ **do** $\cdots$ **end while** and **if** $x = 0$ **then** $\cdots$ **else** $\cdots$ **end if** for control flow;
- GOTO, programs running with a program counter using conditionals (**if**), commands to jump to a specific line (**goto**), and commands to terminate (**halt**) for control flow

Computation Models

We have mainly focused on Turing machines to model computability. There are, however, other models for computability that are commonly used:

- WHILE, programs using **while** $x \neq 0$ **do** $\cdots$ **end while** and **if** $x = 0$ **then** $\cdots$ **else** $\cdots$ **end if** for control flow;
- GOTO, programs running with a program counter using conditionals (**if**), commands to jump to a specific line (**goto**), and commands to terminate (**halt**) for control flow;
- LOOP, programs using conditionals (**if**) and loops of a pre-determined fixed length (**loop**) for control flow;

Computation Models

- primitively recursive (PR), functions of the shape

$$f(0, \bar{x}) = t_0$$
$$f(m+1, \bar{x}) = t$$

  where t_0 is a term that is only using x_i and other PR functions and t is a term that may use $f(m, \bar{x})$, x_i, and other PR functions

Computation Models

- primitively recursive (PR), functions of the shape

$$f(0, \bar{x}) = t_0$$
$$f(m+1, \bar{x}) = t$$

  where t_0 is a term that is only using x_i and other PR functions and t is a term that may use $f(m, \bar{x})$, x_i, and other PR functions; and

- μ-recursive (μR), an extension of PR where programs are allowed to use the μ-operator which is defined as

$$\mu f(\bar{x} = \text{find}(0, \bar{x})$$
$$\text{find}(n, \bar{x}) = \begin{cases} n & f(n, \bar{x}) = 0 \\ \text{find}(n+1, \bar{x}) & \text{otherwise.} \end{cases}$$

Computation Models

Turing-computable functions are functionally equivalent to

- Turing machines;
- WHILE programs;
- GOTO programs; and
- μ-recursive programs.

Computation Models

Turing-computable functions are functionally equivalent to

- Turing machines;
- WHILE programs;
- GOTO programs; and
- μ-recursive programs.

LOOP and PR programs are also able to express the same set of functions, but this set is a true subset of all Turing-computable functions.

Computation Models

Turing-computable functions are functionally equivalent to

- Turing machines;
- WHILE programs;
- GOTO programs; and
- μ-recursive programs.

LOOP and PR programs are also able to express the same set of functions, but this set is a true subset of all Turing-computable functions.
In other words, there exist Turing-computable functions that are not primitively recursive (or computable by a LOOP program), for example the Ackermann function which is discussed next.

Ackermann Function

The Ackermann function can be used to show that a function f is not primitively recursive.

Ackermann Function

The Ackermann function can be used to show that a function f is not primitively recursive.

Definition 23

The Ackermann function a is not primitively recursive and is defined as

$$
\begin{aligned}
a(0, n) &= n + 1 \\
a(m + 1, 0) &= a(m, 1) \\
a(m + 1, n + 1) &= a(m, a(m + 1, n))
\end{aligned}
$$

Ackermann Function

The Ackermann function can be used to show that a function f is not primitively recursive.

Definition 23

The Ackermann function a is not primitively recursive and is defined as

$$a(0, n) = n + 1$$
$$a(m + 1, 0) = a(m, 1)$$
$$a(m + 1, n + 1) = a(m, a(m + 1, n))$$

Theorem 24

For every primitively recursive function $f : \mathbb{N}^k \to \mathbb{N}$ *there exists a* $t \in \mathbb{N}$ *such that* $\forall \bar{x} \in \mathbb{N}^k.\ f(\bar{x}) < a(t, \max \bar{x})$.