Logic and Theoretical Foundation of Computer Science

LaTFoCS

Pamela Fleischmann

fpa@informatik.uni-kiel.de

Winter Semester 2019

Kiel University
Dependable Systems Group

Turing Machines and Effective Computability

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?
- Several approaches:

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n | n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?
- Several approaches:
  - Turing Machines (Alan Turing 1936)

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?
- Several approaches:
  - Turing Machines (Alan Turing 1936)
  - Post systems (Emil Post 1936)

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?
- Several approaches:
  - Turing Machines (Alan Turing 1936)
  - Post systems (Emil Post 1936)
  - μ-recursive functions (Kurt Gödel, Jacques Herbrand 1931)

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?
- Several approaches:
  - Turing Machines (Alan Turing 1936)
  - Post systems (Emil Post 1936)
  - μ-recursive functions (Kurt Gödel, Jacques Herbrand 1931)
  - λ-calculus (Alonso Church 1933, Stephen C. Kleene 1935)

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?
- Several approaches:
  - Turing Machines (Alan Turing 1936)
  - Post systems (Emil Post 1936)
  - μ-recursive functions (Kurt Gödel, Jacques Herbrand 1931)
  - λ-calculus (Alonso Church 1933, Stephen C. Kleene 1935)
  - combinatory logic (Moses Schönfinkel 1924, Haskell B. Curry 1929)

New Approach

- Goal: Automaton that can recognize $\{a^n b^n c^n \mid n \in \mathbb{N}\}$
- or higher goal: What does effective computation mean?
- Several approaches:
  - Turing Machines (Alan Turing 1936)
  - Post systems (Emil Post 1936)
  - μ-recursive functions (Kurt Gödel, Jacques Herbrand 1931)
  - λ-calculus (Alonso Church 1933, Stephen C. Kleene 1935)
  - combinatory logic (Moses Schönfinkel 1924, Haskell B. Curry 1929)
- all models are equivalent

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs
- Reminder: 2DFA $\mathscr{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_a, q_r)$

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs
- Reminder: 2DFA $\mathscr{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_a, q_r)$
  - allowed to move the head to the right, to the left, or don't move it at all

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs
- Reminder: 2DFA $\mathscr{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_a, q_r)$
  - allowed to move the head to the right, to the left, or don't move it at all
  - **not** allowed to write

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs
- Reminder: 2DFA $\mathscr{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_a, q_r)$
  - allowed to move the head to the right, to the left, or don't move it at all
  - **not** allowed to write
- can we generalise it?

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs
- Reminder: 2DFA $\mathscr{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_a, q_r)$
  - allowed to move the head to the right, to the left, or don't move it at all
  - **not** allowed to write
- can we generalise it?
  1. we allow to write

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs
- Reminder: 2DFA $\mathscr{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_a, q_r)$
  - allowed to move the head to the right, to the left, or don't move it at all
  - **not** allowed to write
- can we generalise it?
  1. we allow to write
  2. where to write, can we store additional information?

Turing-Machines - Informal Approach

- when we examined the 2DFA we noticed that it is not more powerful than usual DFAs
- Reminder: 2DFA $\mathscr{A} = (Q, \Sigma, \vdash, \dashv, \delta, q_0, q_a, q_r)$
  - allowed to move the head to the right, to the left, or don't move it at all
  - **not** allowed to write
- can we generalise it?
  1. we allow to write
  2. where to write, can we store additional information?
  3. we allow to write at the right side of the word as much as we want

Turing-Machines - Informal Example

Let's have a look at $\{a^n b^n c^n \mid n \in \mathbb{N}\}$.
Input on the machine's tape: $a^n b^n c^n$ for one $n \in \mathbb{N}$

1. Reading start symbol $\vdash$: move head to the right, q_0
2. Reading a, state q_0: write x, move head to the right, q_1
3. Reading a, state q_1: move head to the right
4. Reading b, state q_1: write y, move head to the right, q_2
5. Reading b, state q_2: move head to the right
6. Reading c, state q_2: write z, move head to the left, q_3
7. Reading anything but $\vdash$, state q_3: move head to the left
8. Reading $\vdash$, state q_3: move head to the right, q_0

Turing-Machines - Informal Example - Cont.

9. Reading $\sqcup$, q_1: q_r
10. Reading $\sqcup$, q_2: q_r
11. Reading b or c, q_0: q_r
12. Reading $\sqcup$, state q_0: q_a
13. Reading x, state q_1: move head to the right
14. Reading y, state q_2: move head to the right
15. Reading z, state q_3: move head to the right
16. all other end in q_r

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathscr{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0
- finite input alphabet Σ

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0
- finite input alphabet Σ
- finite tape alphabet $\Gamma \supseteq \Sigma$

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0
- finite input alphabet Σ
- finite tape alphabet $\Gamma \supseteq \Sigma$
- blank symbol $\sqcup \in \Gamma \backslash \Sigma$

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0
- finite input alphabet Σ
- finite tape alphabet $\Gamma \supseteq \Sigma$
- blank symbol $\sqcup \in \Gamma \backslash \Sigma$
- left endmarker $\vdash \in \Gamma \backslash \Sigma$

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0
- finite input alphabet Σ
- finite tape alphabet $\Gamma \supseteq \Sigma$
- blank symbol $\sqcup \in \Gamma \backslash \Sigma$
- left endmarker $\vdash \in \Gamma \backslash \Sigma$
- transistion function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathscr{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0
- finite input alphabet Σ
- finite tape alphabet $\Gamma \supseteq \Sigma$
- blank symbol $\sqcup \in \Gamma \backslash \Sigma$
- left endmarker $\vdash \in \Gamma \backslash \Sigma$
- transistion function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- accepting state q_a

Formal Definition

Definition (1-tape, deterministic Turing Machine (1DTM))

$\mathscr{A} = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_a, q_r)$ 1DTM with

- finite set of states Q with start state q_0
- finite input alphabet Σ
- finite tape alphabet $\Gamma \supseteq \Sigma$
- blank symbol $\sqcup \in \Gamma \backslash \Sigma$
- left endmarker $\vdash \in \Gamma \backslash \Sigma$
- transistion function $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- accepting state q_a
- rejecting state $q_r \neq q_a$

Intuition for a TM's functioning

$\delta(q, a) = (q', b, D) \mathrel{\hat{=}}$

- I am in state q
- I read a from the tape
- I write b on the tape
- I go to state q'
- I move the read/write-head in direction D

Convenient Restrictions

- $\delta(p, \vdash) = (q, \vdash, R)$ for all $p, q \in Q$
- $\delta(q_a, b) = (q_a, c, D)$ for all $b, c \in \Gamma$, $D \in \{L, R\}$
- $\delta(q_r, b) = (q_r, c, D)$ for all $b, c \in \Gamma$, $D \in \{L, R\}$

Configuration of 1-DTM

Definition

configuration is element of $Q \times \{y\sqcup^{\omega} | y \in \Gamma^*\} \times \mathbb{N}$

- $\hat{=}$ state - tape-content - head-position

Configuration of 1-DTM

Definition

configuration is element of $Q \times \{y\sqcup^{\omega} | y \in \Gamma^{*}\} \times \mathbb{N}$

- $\hat{=}$ state - tape-content - head-position
- start configuration: $(q_0, \vdash x\sqcup^{\omega}, 0)$

Next Configuration Relation for 1-DTMs

Definition

- substituting $w[i]$ by b: $w[b|i]$
- next configuration relation

$$(p, z, n) \xrightarrow[\mathcal{A}]{1} \begin{cases} (q, z[b|n], n-1) & \text{if } \delta(p, z[n]) = (q, b, L) \\ (q, z[b|n], n+1) & \text{if } \delta(p, z[n]) = (w, b, R). \end{cases}$$

- $\xrightarrow[\mathcal{A}]{n}$ and $\xrightarrow[\mathcal{A}]{*}$ as usual

Acceptance and Rejection

Definition

- $\mathcal{A}$ accepts $w \in \Sigma^*$: $(q_0, \vdash w_\sqcup{}^\omega, 0) \xrightarrow[\mathcal{A}]{n} (q_a, y, n)$
- $\mathcal{A}$ rejects $w \in \Sigma^*$: $(q_0, \vdash w_\sqcup{}^\omega, 0) \xrightarrow[\mathcal{A}]{n} (q_r, y, n)$
- $\mathcal{A}$ halts on $w \in \Sigma^*$: $\mathcal{A}$ accepts or rejects w
- $\mathcal{A}$ loops on $w \in \Sigma^*$: $\mathcal{A}$ does not halt on w
- $L(\mathcal{A}) = \{w \in \Sigma^* \mid \mathcal{A} \text{ accepts } w\}$

Recursively Enumerable

Definition

- 1-DTM $\mathscr{A}$ total: $\mathscr{A}$ halts on all inputs
- $L \subseteq \Sigma^*$ recursively enumerable: $\exists$ 1-DTM $\mathscr{A} : L(\mathscr{A}) = L$
- $L \subseteq \Sigma^*$ co-recursively enumerable: $\exists$ 1-DTM $\mathscr{A} : L(\mathscr{A}) = \overline{L}$
- $L \subseteq \Sigma^*$ recursive: $\exists$ total 1-DTM $\mathscr{A} : L(\mathscr{A}) = L$

Decidability and Semidecidability

Definition

$P \subseteq \Sigma^*$

- P is decidable $\Leftrightarrow$ P is recursive
- P is semidecidable $\Leftrightarrow$ P is recursively enumerable

Decidability and Semidecidability

Definition

$P \subseteq \Sigma^*$

- P is decidable $\Leftrightarrow$ P is recursive
- P is semidecidable $\Leftrightarrow$ P is recursively enumerable

at the beginning of the semester, I told you that HALT is not decidable

$\Rightarrow$ at some point we need to prove that no 1-DTM recognising HALT exists

Addition - we can count!

Is the language $L_+ = \{a^r\#_1 a^s\#_2 a^t \in \{a, \#_1, \#_2\}^* | \, r + s = t\}$ recognizable by a 1-DTM?

Addition - we can count!

Is the language $L_+ = \{a^r\#_1a^s\#_2a^t \in \{a, \#_1, \#_2\}^* \,|\, r + s = t\}$ recognizable by a 1-DTM?

- ○ $\delta(q_0, \vdash) = (q_0, \vdash, R)$
- ○ $\delta(q_0, a) = (q_1, \sqcup, R)$
- ○ $\delta(q_0, \#_1) = (q_0, \#_1, R)$
- ○ $\delta(q_0, \#_2) = (q_4, \#_2, R)$
- ○ $\delta(q_1, x) = (q_1, x, R)$ for $x \neq \#_2$
- ○ $\delta(q_1, \#_2) = (q_2, \#_2, R)$
- ○ $\delta(q_2, \sqcup') = (q_2, \sqcup', R)$
- ○ $\delta(q_2, a) = (q_3, \sqcup', L)$
- ○ $\delta(q_3, x) = (q_3, x, L)$ for $x \neq \sqcup$
- ○ $\delta(q_3, \sqcup) = (q_0, \sqcup, R)$
- ○ $\delta(q_4, \sqcup') = (q_4, \sqcup', R)$
- ○ $\delta(q_4, \sqcup) = (q_4, \sqcup, R)$

$$\delta(q, y) = (q_r, y, R) \text{ for all others}$$

Addition - we can count!

Is the language $L_+ = \{a^r \#_1 a^s \#_2 a^t \in \{a, \#_1, \#_2\}^* \mid r + s = t\}$ recognizable by a 1-DTM?

- $\delta(q_0, \vdash) = (q_0, \vdash, R)$
- $\delta(q_0, a) = (q_1, \sqcup, R)$
- $\delta(q_0, \#_1) = (q_0, \#_1, R)$
- $\delta(q_0, \#_2) = (q_4, \#_2, R)$
- $\delta(q_1, x) = (q_1, x, R)$ for $x \neq \#_2$
- $\delta(q_1, \#_2) = (q_2, \#_2, R)$
- $\delta(q_2, \sqcup') = (q_2, \sqcup', R)$
- $\delta(q_2, a) = (q_3, \sqcup', L)$
- $\delta(q_3, x) = (q_3, x, L)$ for $x \neq \sqcup$
- $\delta(q_3, \sqcup) = (q_0, \sqcup, R)$
- $\delta(q_4, \sqcup') = (q_4, \sqcup', R)$
- $\delta(q_4, \sqcup) = (q_4, \sqcup, R)$

$$\delta(q, y) = (q_r, y, R) \text{ for all others}$$

we proved that the usual addition is decidable!

Extensions of TMs

Multiple Tapes

we have n tapes:

Definition (n-DTM)

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash^n, \sqcup, \delta, q_0, q_a, q_r)$ with

- $Q, q_0, \Sigma, \sqcup, \vdash, q_a, q_r$ as in 1-DTM
- $\delta : Q \times \Gamma^n \to Q \times \Gamma^n \times \{L, R\}^n$

Multiple Tapes

we have n tapes:

Definition (n-DTM)

$\mathcal{A} = (Q, \Sigma, \Gamma, \vdash^n, \sqcup, \delta, q_0, q_a, q_r)$ with

- $Q, q_0, \Sigma, \sqcup, \vdash, q_a, q_r$ as in 1-DTM
- $\delta : Q \times \Gamma^n \to Q \times \Gamma^n \times \{L, R\}^n$

n-DTMs are not more powerful than 1-DTMs

More Extensions

- Counter-Automaton: two-way read-only head and k integer counters (inc, dec, test on 0)
  - 2-counter automaton is equivalent to PDA,
  - 4-counter-automaton equivalent to DTM

More Extensions

- Counter-Automaton: two-way read-only head and k integer counters (inc, dec, test on 0)
  - 2-counter automaton is equivalent to PDA,
  - 4-counter-automaton equivalent to DTM
- Enumeration Machine: 1 alphabet Σ, two tapes (2-way read/write working tape, 1-way write output tape
  - special enumeration state : machine in it $\Rightarrow$ current content of output tape added to language, output tape erased
  - enumeration machines are equivalent to Turing machines

Universal Machines and Diagonalisation

Processing TMs with TMs?!?

Is this idea stupid?

Processing TMs with TMs?!?

Is this idea stupid?

- Does there exist a washing machine washing washing machines?

Processing TMs with TMs?!?

Is this idea stupid?

- Does there exist a washing machine washing washing machines?
- Does there exist a shredder shreddering shredders?

Processing TMs with TMs?!?

Is this idea stupid?

- Does there exist a washing machine washing washing machines?
- Does there exist a shredder shreddering shredders?
- Does there exist a programme processing other programmes?

Processing TMs with TMs?!?

Is this idea stupid?

- Does there exist a washing machine washing washing machines?
- Does there exist a shredder shreddering shredders?
- Does there exist a programme processing other programmes?

Perhaps the idea is not completely stupid.

How to feed a TM with a TM?

- Problem:
  - $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \sqcup, \vdash, q_a, q_r)$
  - expected input: $w \in \Sigma^*$

How to feed a TM with a TM?

- Problem:
  - $\mathscr{A} = (Q, \Sigma, \Gamma, q_0, \sqcup, \vdash, q_a, q_r)$
  - expected input: $w \in \Sigma^*$
- how can we get $\mathscr{A}$ as an input?

How to feed a TM with a TM?

- Problem:
  - $\mathcal{A} = (Q, \Sigma, \Gamma, q_0, \sqcup, \vdash, q_a, q_r)$
  - expected input: $w \in \Sigma^*$
- how can we get $\mathcal{A}$ as an input?
- Convenience: $\Sigma = \{0, 1\}$

Encoding a Turing Machine

- states $Q = \{q_0, \ldots, q_n\}$: 0^i represents q_i

Encoding a Turing Machine

- states $Q = \{q_0, \ldots, q_n\}$: 0^i represents q_i
- alphabet $\Sigma = \{a_1, \ldots, a_m\}$: 0^i represents a_i

Encoding a Turing Machine

- states $Q = \{q_0, \ldots, q_n\}$: 0^i represents q_i
- alphabet $\Sigma = \{a_1, \ldots, a_m\}$: 0^i represents a_i
- tape alphabet Γ: since $\Sigma \subseteq \Gamma$ holds, assume $\Gamma = \{a_1, \ldots, a_k\}$ for $k \geq m \rightsquigarrow 0^i$ represents a_i

Encoding a Turing Machine

- states $Q = \{q_0, \ldots, q_n\}$: 0^i represents q_i
- alphabet $\Sigma = \{a_1, \ldots, a_m\}$: 0^i represents a_i
- tape alphabet Γ: since $\Sigma \subseteq \Gamma$ holds, assume $\Gamma = \{a_1, \ldots, a_k\}$ for $k \geq m \rightsquigarrow 0^i$ represents a_i
- the representation of the special states is given by Q

Encoding a Turing Machine

- states $Q = \{q_0, \ldots, q_n\}$: 0^i represents q_i
- alphabet $\Sigma = \{a_1, \ldots, a_m\}$: 0^i represents a_i
- tape alphabet Γ: since $\Sigma \subseteq \Gamma$ holds, assume $\Gamma = \{a_1, \ldots, a_k\}$ for $k \geq m \rightsquigarrow 0^i$ represents a_i
- the representation of the special states is given by Q
- the symbols $\vdash$ and $\sqcup$ are given by Γ

Encoding a Turing Machine

- states $Q = \{q_0, \ldots, q_n\}$: 0^i represents q_i
- alphabet $\Sigma = \{a_1, \ldots, a_m\}$: 0^i represents a_i
- tape alphabet Γ: since $\Sigma \subseteq \Gamma$ holds, assume $\Gamma = \{a_1, \ldots, a_k\}$ for $k \geq m \rightsquigarrow 0^i$ represents a_i
- the representation of the special states is given by Q
- the symbols $\vdash$ and $\sqcup$ are given by Γ

How to distinguish between q_i and a_i, both are 0^i

Encoding a Turing Machine

- states $Q = \{q_0, \dots, q_n\}$: 0^i represents q_i
- alphabet $\Sigma = \{a_1, \dots, a_m\}$: 0^i represents a_i
- tape alphabet Γ: since $\Sigma \subseteq \Gamma$ holds, assume $\Gamma = \{a_1, \dots, a_k\}$ for $k \geq m \rightsquigarrow 0^i$ represents a_i
- the representation of the special states is given by Q
- the symbols $\vdash$ and $\sqcup$ are given by Γ

How to distinguish between q_i and a_i, both are 0^i

$$\underbrace{0^n}_{Q}\ 1\ \underbrace{0^m}_{\Sigma}\ 1\ \underbrace{0^k}_{\Gamma}\ 1\ \underbrace{0^a}_{q_a}\ 1\ \underbrace{0^r}_{q_r}\ 1\ \underbrace{0^u}_{\vdash}\ 1\ \underbrace{0^v}_{\sqcup}\ 1$$

Encoding the Transitions

- $\delta(q, a) = (q', b, D)$

Encoding the Transitions

- $\delta(q, a) = (q', b, D)$
- there exists $i, i' \in [n]$ with $q = 0^i$, $q' = 0^{i'}$

Encoding the Transitions

- $\delta(q, a) = (q', b, D)$
- there exists $i, i' \in [n]$ with $q = 0^i$, $q' = 0^{i'}$
- there exists $j, j' \in [k]$ with $a = 0^j$, $b = 0^{j'}$

Encoding the Transitions

- $\delta(q, a) = (q', b, D)$
- there exists $i, i' \in [n]$ with $q = 0^i$, $q' = 0^{i'}$
- there exists $j, j' \in [k]$ with $a = 0^j$, $b = 0^{j'}$
- $L = 0$, $R = 00$

Encoding the Transitions

- $\delta(q, a) = (q', b, D)$
- there exists $i, i' \in [n]$ with $q = 0^i$, $q' = 0^{i'}$
- there exists $j, j' \in [k]$ with $a = 0^j$, $b = 0^{j'}$
- $L = 0$, $R = 00$
- thus we get

$$\underbrace{0^i}_{q} \; 1 \; \underbrace{0^j}_{a} \; 1 \; \underbrace{0^{i'}}_{q'} \; 1 \; \underbrace{0^{j'}}_{b} \; 1 \; \underbrace{0^{\ell}}_{D}$$

Encoding the Transitions

- $\delta(q, a) = (q', b, D)$
- there exists $i, i' \in [n]$ with $q = 0^i$, $q' = 0^{i'}$
- there exists $j, j' \in [k]$ with $a = 0^j$, $b = 0^{j'}$
- $L = 0$, $R = 00$
- thus we get

$$\underbrace{0^i}_{q} \; 1 \; \underbrace{0^j}_{a} \; 1 \; \underbrace{0^{i'}}_{q'} \; 1 \; \underbrace{0^{j'}}_{b} \; 1 \; \underbrace{0^\ell}_{D}$$

we encoded the complete machine and denote it by $\langle \mathscr{A} \rangle$:-)

Universal Turing Machine

Definition

Universal Turing Machine U defined by

$$L(U) = \{\langle \mathscr{A}, w\rangle \mid w \in L(\mathscr{A}), \mathscr{A} \text{ TM}\}.$$

Universal Turing Machine

Definition

Universal Turing Machine U defined by

$$L(U) = \{\langle \mathcal{A}, w\rangle \mid w \in L(\mathcal{A}), \mathcal{A} \text{ TM}\}.$$

Description of U

Universal Turing Machine

Definition

Universal Turing Machine U defined by

$$L(U) = \{\langle \mathscr{A}, w\rangle \mid w \in L(\mathscr{A}), \mathscr{A} \text{ TM}\}.$$

Description of U

1. checking if $\mathscr{A}$ is a valid TM

Universal Turing Machine

Definition

Universal Turing Machine U defined by

$$L(U) = \{\langle \mathscr{A}, w\rangle \mid w \in L(\mathscr{A}), \mathscr{A} \text{ TM}\}.$$

Description of U

1. checking if $\mathscr{A}$ is a valid TM
2. checking if w is in $\{0,1\}^*$

Universal Turing Machine

Definition

Universal Turing Machine U defined by

$$L(U) = \{\langle \mathcal{A}, w \rangle \mid w \in L(\mathcal{A}), \mathcal{A} \text{ TM}\}.$$

Description of U

1. checking if $\mathcal{A}$ is a valid TM
2. checking if w is in $\{0, 1\}^*$
3. U simulates $\mathcal{A}$

Universal Turing Machine

Definition

Universal Turing Machine U defined by

$$L(U) = \{\langle \mathcal{A}, w\rangle \mid w \in L(\mathcal{A}), \mathcal{A} \text{ TM}\}.$$

Description of U

1. checking if $\mathcal{A}$ is a valid TM
2. checking if w is in $\{0, 1\}^*$
3. U simulates $\mathcal{A}$
   3.1 keep track of state and actual letter

Universal Turing Machine

Definition

Universal Turing Machine U defined by

$$L(U) = \{\langle \mathcal{A}, w \rangle \mid w \in L(\mathcal{A}), \mathcal{A} \text{ TM}\}.$$

Description of U

1. checking if $\mathcal{A}$ is a valid TM
2. checking if w is in $\{0, 1\}^*$
3. U simulates $\mathcal{A}$
   3.1 keep track of state and actual letter
   3.2 if $\mathcal{A}$ accepts/rejects, U does the same

Halting- and Membership-Problem

Definition

- $\mathrm{HALT} = \{\langle A, w\rangle \mid \mathscr{A} \text{ halts on } w\}$

Halting- and Membership-Problem

Definition

- $\text{HALT} = \{\langle A, w\rangle \mid \mathscr{A} \text{ halts on } w\}$
- $\text{MEM} = \{\langle A, w\rangle \mid w \in L(\mathscr{A})\}.$

Halting- and Membership-Problem

Definition

- HALT = $\{\langle A, w\rangle \mid \mathscr{A} \text{ halts on } w\}$
- MEM = $\{\langle A, w\rangle \mid w \in L(\mathscr{A})\}$.

we want to proof that both languages are not recursive, i.e. these problems are undecidable for TMs

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \to \mathscr{P}(A)$ is surjective (and thus not bijective).

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \to \mathscr{P}(A)$ is surjective (and thus not bijective).

Proof:

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \longrightarrow \mathscr{P}(A)$ is surjective (and thus not bijective).

Proof:

- Suppose: $f : A \longrightarrow \mathscr{P}(A)$ is surjective

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \longrightarrow \mathcal{P}(A)$ is surjective (and thus not bijective).

Proof:

- Suppose: $f : A \longrightarrow \mathcal{P}(A)$ is surjective
- $B := \{x \in A \mid x \notin f(x)\}$

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \longrightarrow \mathscr{P}(A)$ is surjective (and thus not bijective).

Proof:

- Suppose: $f : A \longrightarrow \mathscr{P}(A)$ is surjective
- $B := \{x \in A | \, x \notin f(x)\}$
- $\Rightarrow B \subseteq A \Rightarrow B \in \mathscr{P}(A)$

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \longrightarrow \mathscr{P}(A)$ is surjective (and thus not bijective).

Proof:

- Suppose: $f : A \longrightarrow \mathscr{P}(A)$ is surjective
- $B := \{x \in A |\, x \notin f(x)\}$
- $\Rightarrow B \subseteq A \Rightarrow B \in \mathscr{P}(A)$
- f surjective $\Rightarrow \exists y \in A : \; f(y) = B$

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \longrightarrow \mathscr{P}(A)$ is surjective (and thus not bijective).

Proof:

- Suppose: $f : A \longrightarrow \mathscr{P}(A)$ is surjective
- $B := \{x \in A |\, x \notin f(x)\}$
- $\Rightarrow B \subseteq A \Rightarrow B \in \mathscr{P}(A)$
- f surjective $\Rightarrow \exists y \in A :\ f(y) = B$
- case 1 $y \in f(y)$: $y \in B \Rightarrow y \notin f(y)$

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \longrightarrow \mathscr{P}(A)$ is surjective (and thus not bijective).

Proof:

- Suppose: $f : A \longrightarrow \mathscr{P}(A)$ is surjective
- $B := \{x \in A | \, x \notin f(x)\}$
- $\Rightarrow B \subseteq A \Rightarrow B \in \mathscr{P}(A)$
- f surjective $\Rightarrow \exists y \in A : \; f(y) = B$
- case 1 $y \in f(y)$: $y \in B \Rightarrow y \notin f(y)$
- case 2 $y \notin f(y)$: $y \in B \Rightarrow y \in f(y)$

Cantor's Diagonalisation

Theorem

For any set A, no function $f : A \longrightarrow \mathscr{P}(A)$ is surjective (and thus not bijective).

Proof:

- Suppose: $f : A \longrightarrow \mathscr{P}(A)$ is surjective
- $B := \{x \in A | \, x \notin f(x)\}$
- $\Rightarrow B \subseteq A \Rightarrow B \in \mathscr{P}(A)$
- f surjective $\Rightarrow \exists y \in A : \; f(y) = B$
- case 1 $y \in f(y)$: $y \in B \Rightarrow y \notin f(y)$
- case 2 $y \notin f(y)$: $y \in B \Rightarrow y \in f(y)$
- Contradiction

HALT is undecidable

Theorem

HALT *is undecidable*

HALT is undecidable

Theorem

HALT *is undecidable*

Proof:

HALT is undecidable

Theorem

HALT *is undecidable*

Proof:

- $\mathscr{A}_x$ TM whose encoding is x (if not valid description, $\mathscr{A}_x$ is a trivial machine)

HALT is undecidable

Theorem

HALT *is undecidable*

Proof:

- $\mathscr{A}_x$ TM whose encoding is x (if not valid description, $\mathscr{A}_x$ is a trivial machine)
- $\Rightarrow \mathscr{A}_\varepsilon, \mathscr{A}_0, \mathscr{A}_1, \mathscr{A}_{00}, \mathscr{A}_{01}, \dots$

HALT is undecidable

Theorem

HALT *is undecidable*

Proof:

- $\mathscr{A}_x$ TM whose encoding is x (if not valid description, $\mathscr{A}_x$ is a trivial machine)
- $\Rightarrow \mathscr{A}_\varepsilon, \mathscr{A}_0, \mathscr{A}_1, \mathscr{A}_{00}, \mathscr{A}_{01}, \ldots$
- $M = (m_{ij})$ matrix over $\mathscr{A}_x$ for $x \in \Sigma^*$ and Σ^* defined by

$$m_{ij} = \begin{cases} H & \text{if } \mathscr{A}_i \text{ halts with input } j, \\ L & \text{if } \mathscr{A}_i \text{ loops with input } j. \end{cases}$$

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts
- a column j describes all machines halting for j

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts
- a column j describes all machines halting for j
- Suppose: existence of total machine K accepting HALT

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts
- a column j describes all machines halting for j
- Suppose: existence of total machine K accepting HALT
  - K accepts x, if $\mathscr{A}$ halts on x

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts
- a column j describes all machines halting for j
- Suppose: existence of total machine K accepting HALT
  - K accepts x, if $\mathscr{A}$ halts on x
  - K rejects x, if $\mathscr{A}$ loops on x

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts
- a column j describes all machines halting for j
- Suppose: existence of total machine K accepting HALT
  - K accepts x, if $\mathscr{A}$ halts on x
  - K rejects x, if $\mathscr{A}$ loops on x
  - Consider machine N with input x

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts
- a column j describes all machines halting for j
- Suppose: existence of total machine K accepting HALT
  - K accepts x, if $\mathscr{A}$ halts on x
  - K rejects x, if $\mathscr{A}$ loops on x
  - Consider machine N with input x
    1. builds $\mathscr{A}_x$ and writes $\langle \mathscr{A}_x, x \rangle$

Proof cont.

- $\mathscr{A}_i$ TM whose encoding is i
- row i of M describes for every word whether $\mathscr{A}_i$ halts
- a column j describes all machines halting for j
- Suppose: existence of total machine K accepting HALT
  - K accepts x, if $\mathscr{A}$ halts on x
  - K rejects x, if $\mathscr{A}$ loops on x
  - Consider machine N with input x
    1. builds $\mathscr{A}_x$ and writes $\langle \mathscr{A}_x, x \rangle$
    2. run K on this and accept if K rejects and loop somehow otherwise

Proof cont.

- $\mathscr{A}_x$ TM whose encoding is x
- Suppose: existence of total machine K accepting HALT
  - Consider machine N with input x, accepting if K rejects and looping otherwise
  - K accepts x, if $\mathscr{A}$ halts on x
  - K rejects x, if $\mathscr{A}$ loops on x
  - N halts on $x \Leftrightarrow K$ rejects $\langle \mathscr{A}_x, x \rangle \Leftrightarrow \mathscr{A}_x$ loops on x

Proof cont.

- $\mathscr{A}_x$ TM whose encoding is x
- Suppose: existence of total machine K accepting HALT
  - Consider machine N with input x, accepting if K rejects and looping otherwise
  - K accepts x, if $\mathscr{A}$ halts on x
  - K rejects x, if $\mathscr{A}$ loops on x
  - N halts on $x \Leftrightarrow K$ rejects $\langle \mathscr{A}_x, x \rangle \Leftrightarrow \mathscr{A}_x$ loops on x
  - N is different from all $\mathscr{A}_x$

Proof cont.

- $\mathscr{A}_x$ TM whose encoding is x
- Suppose: existence of total machine K accepting HALT
  - Consider machine N with input x, accepting if K rejects and looping otherwise
  - K accepts x, if $\mathscr{A}$ halts on x
  - K rejects x, if $\mathscr{A}$ loops on x
  - N halts on $x \Leftrightarrow K$ rejects $\langle \mathscr{A}_x, x \rangle \Leftrightarrow \mathscr{A}_x$ loops on x
  - N is different from all $\mathscr{A}_x$
  - $\mathscr{A}_x$ for all $x \in \Sigma^*$ are all TMs over $\{0,1\}^*$ (Contradiction) $\square$