TDDD14/TDDD85
Slides for Lecture 14, 2017

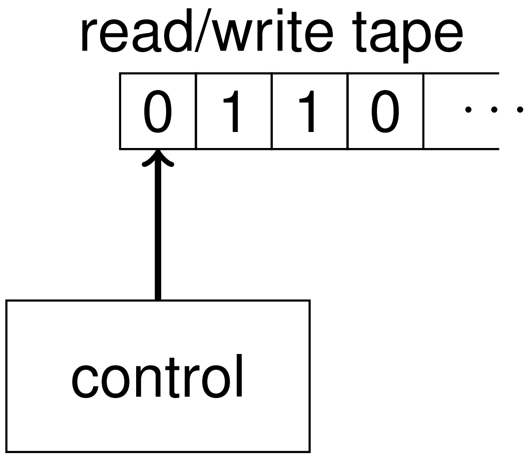
Slides originally for TDDD65 by Gustav Nordh

Some differences to Kozen:

- Kozen uses a predefined left-end marker symbol for TMs. One can instead assume that nothing happens if trying to move left at the first position.
- "Turing recognizable" is called "recursively enumerable" (or "semi-decidable") in Kozen.
- "Decidable" is primarily called "recursive" in Kozen (but both terms are used).

Computability

What can be computed?



Turing machine

Alan Turing (1912-1954)

Definition of a Turing machine

Definition

A Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where

- Q is the finite set of states
- Σ is the finite input alphabet not containing the blank symbol B
- Γ is the finite tape alphabet where $B \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state

Comparision with finite automata

- A Turing machine can both write on the tape and read from it
- The read-write head can move both to the left and right
- The tape is infinite
- The special states for rejecting and accepting take effect immediately

Turing machine computation

- Initially the machine recieves the input on the leftmost part of the tape
- Computation proceeds according to the transition function
- The computation continues until the machine enters the accept or reject states at which point it halts.
- The machine may continue forever without entering the accept or reject states, in which case we say that the machine loops.

Turing-recognizable, decidable

Definition

The collection of strings that a Turing machine M accepts is the language recognized by M, denoted $L(M)$

Turing-recognizable, decidable

Definition

The collection of strings that a Turing machine M accepts is the language recognized by M, denoted $L(M)$

Definition

A language is Turing-recognizable if some Turing machine recognizes it

Turing-recognizable, decidable

Definition

The collection of strings that a Turing machine M accepts is the language recognized by M, denoted $L(M)$

Definition

A language is Turing-recognizable if some Turing machine recognizes it

Definition

A language is decidable if some Turing machine recognizes it and rejects all strings that are not in the language

Turing machines, decidable

Definition

A language is decidable if some Turing machine recognizes it and rejects all strings that are not in the language

Example

Consider a Turing machine *M* with $\Sigma = \{0, 1\}$ that works as follows: *M* accept all strings of even length and loop on all strings of odd length.

Turing machines, decidable

Definition

A language is decidable if some Turing machine recognizes it and rejects all strings that are not in the language

Example

Consider a Turing machine M with $\Sigma = \{0, 1\}$ that works as follows: M accept all strings of even length and loop on all strings of odd length.
Is $L(M)$ decidable?

Turing machines, decidable

Definition

A language is decidable if some Turing machine recognizes it and rejects all strings that are not in the language

Example

Consider a Turing machine M with $\Sigma = \{0, 1\}$ that works as follows: M accept all strings of even length and loop on all strings of odd length.
Is $L(M)$ decidable?
YES! For example by the Turing machine M' which accept all strings of even length and reject all strings of odd length.

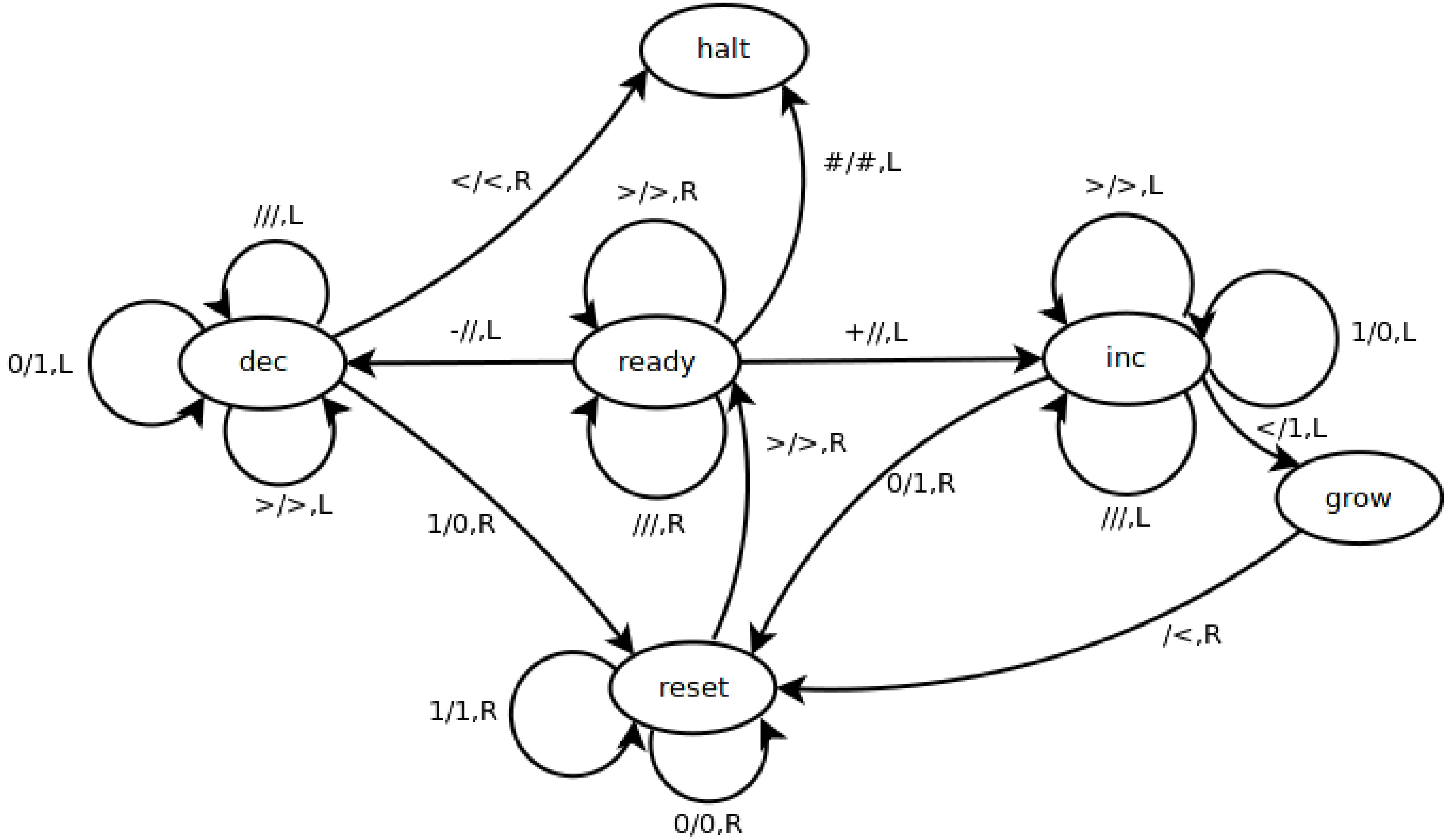
Describing Turing machines

- Machine code
- Assembly code
- Java code
- Pseudo code
- Algorithm description

Describing Turing machines

- Formal description (e.g., State diagram)
- Implementation level description
- Algorithm description

Describing Turing machines: State diagram



Describing Turing machines: Implementation level

Example

Describe a Turing machine that recognizes the language $L = \{0^n 1^n 2^n \mid n \geq 0\}$.

1. Scan the input from left to right and make sure it is of the form $0^*1^*2^*$ (if it is not, then reject)
2. Return the head to the left end of the tape
3. If there is no 0 on the tape, then scan right and check that there are no 1's and 2's on the tape and accept (should a 1 or 2 be on the tape, then reject)
4. Otherwise, cross of the first 0 and continue to the right crossing of the first 1 and the first 2 that is found (should there be no 1 or no 2 on the tape, then reject)
5. Go to Step 2

Describing Turing machines: Algorithm description

- An algorithm description is a list of simple instructions for solving/computing some task.

Describing Turing machines: Algorithm description

- An algorithm description is a list of simple instructions for solving/computing some task.
- If the goal of the algorithm description is to convince the reader that the task can be solved/computed, then “simple instructions” means “can be carried out by a Turing machine”.

Describing Turing machines: Algorithm description

- An algorithm description is a list of simple instructions for solving/computing some task.
- If the goal of the algorithm description is to convince the reader that the task can be solved/computed, then “simple instructions” means “can be carried out by a Turing machine”.
- Algorithm descriptions are similar to mathematical proofs
  - The goal of a mathematical proof is to convince the reader that the truth of a mathematical statement follows from the basic axioms.
  - The goal of an algorithm description is to convince the reader that a task/problem can be solved by Turing machines/computers.

Describing Turing machines: Algorithm description

Example

Describe an algorithm for recognizing the language $L = \{0^n 1^n 2^n \mid n \geq 0\}$.

Describing Turing machines: Algorithm description

Example

Describe an algorithm for recognizing the language $L = \{0^n 1^n 2^n \mid n \geq 0\}$.

1. Check that the input is of the form $0^*1^*2^*$. Then count the number of 0's, 1's, and 2's. If they are the same, accept. Otherwise, reject.

Algorithm

Muhammad ibn Musa al-Khwarizmi (780-850)

Alternatives to Turing machines?

- Why are Turing machines a good model for computation?

Alternatives to Turing machines?

- Why are Turing machines a good model for computation?
- There should be more powerful machines, right?

Alternatives to Turing machines?

Alonzo Church (1903-1995)

Church-Turing thesis

Intuitive notion of computation

equals

Turing machine computation

Consequences of the Church-Turing thesis

- The details of the computational model are not important

Consequences of the Church-Turing thesis

- The details of the computational model are not important
- Opens up the possibility to prove that some problems are not solvable by computers/Turing machines

Consequences of the Church-Turing thesis

- The details of the computational model are not important
- Opens up the possibility to prove that some problems are not solvable by computers/Turing machines
- Humans can be simulated by Turing machines!?
- The universe is a Turing machine!?

Testing the Church-Turing thesis

Theorem

Nondeterministic Turing machines can be simulated by deterministic Turing machines

Definition of a nondeterministic Turing machine

Definition

A nondeterministic Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where

- Q is the finite set of states
- Σ is the finite input alphabet not containing the blank symbol B
- Γ is the finite tape alphabet where $B \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is the transition function
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state

Definition of a nondeterministic Turing machine

Definition

A nondeterministic Turing machine is a 7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, q_{accept}, q_{reject})$ where

- Q is the finite set of states
- Σ is the finite input alphabet not containing the blank symbol B
- Γ is the finite tape alphabet where $B \in \Gamma$ and $\Sigma \subseteq \Gamma$
- $\delta : Q \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{L, R\})$ is the transition function
- $q_0 \in Q$ is the start state
- $q_{accept} \in Q$ is the accept state
- $q_{reject} \in Q$ is the reject state

A nondeterministic Turing machine accepts its input w if at least one of the states explored is an accept state.

Configuration of a Turing machine

Given a Turing machine M operating on an input w: the current state, current tape contents, and current position of the read/write head is the current configuration of M.

Configuration of a Turing machine

Given a Turing machine M operating on an input w: the current state, current tape contents, and current position of the read/write head is the current configuration of M.

Example

$00q_5 10$ represent the configuration where the tape contents is 0010, the state is q_5, and the position of the head is over the 1.

Configuration of a Turing machine

Given a Turing machine M operating on an input w: the current state, current tape contents, and current position of the read/write head is the current configuration of M.

Example

$00q_5 10$ represent the configuration where the tape contents is 0010, the state is q_5, and the position of the head is over the 1.

- The start configuration is $q_0 w$
- An accept configuration is one where the state is q_{accept}
- A reject configuration is one where the state is q_{reject}

Testing the Church-Turing thesis

Theorem

Nondeterministic Turing machines can be simulated by deterministic Turing machines

Proof.

Given a nondeterministic Turing machine N we construct a deterministic Turing machine D such that D accepts input w if and only if N accepts w. D works as follows: $\square$

Testing the Church-Turing thesis

Theorem

Nondeterministic Turing machines can be simulated by deterministic Turing machines

Proof.

Given a nondeterministic Turing machine N we construct a deterministic Turing machine D such that D accepts input w if and only if N accepts w. D works as follows:

1. Given input w. Beginning with the start configuration of N on input w, D explores the computation tree of N on input w.
2. If D encounters an accept configuration of N, then D accepts w.
3. If D has explored the whole computational tree of N without finding an accept configuration, then D rejects w.

□

Testing the Church-Turing thesis

Theorem

Nondeterministic Turing machines can be simulated by deterministic Turing machines

Proof.

Given a nondeterministic Turing machine N we construct a deterministic Turing machine D such that D accepts input w if and only if N accepts w. D works as follows:

1. Given input w. Beginning with the start configuration of N on input w, D explores the computation tree of N on input w in breadth first manner (i.e., level by level).
2. If D encounters an accept configuration of N, then D accepts w.
3. If D has explored the whole computational tree of N without finding an accept configuration, then D rejects w.

□

Decidable languages

$$A_{DFA} = \{\langle B, w \rangle \mid B \textit{ is a DFA that accepts input string } w\}$$

Decidable languages

$$A_{DFA} = \{\langle B, w\rangle \mid B \textit{ is a DFA that accepts input string } w\}$$

Theorem

A_{DFA} is decidable

Decidable languages

$$A_{DFA} = \{\langle B, w\rangle \mid B \textit{ is a DFA that accepts input string } w\}$$

Theorem

A_{DFA} is decidable

Proof.

Let M be a Turing machine that works as follows:

1. Check that the input $\langle B, w\rangle$ is a legal encoding of a DFA B and string w (otherwise reject)
2. Simulate B on input w
3. If the simulation ends in an accept state (of B), then M accepts the input $\langle B, w\rangle$
4. If the simulation ends in a nonaccepting state (of B), then M rejects the input $\langle B, w\rangle$

$\square$

Diagonalization

Diagonalization

Georg Cantor (1845-1918)

Diagonalization

Theorem

Some languages are not Turing-recognizable

Diagonalization

Theorem

Some languages are not Turing-recognizable

Proof idea.

1. The set of all Turing machines is countable
2. The set of all languages is uncountable
3. Since each Turing machine recognize exactly one language, there are languages that are not recognized by any Turing machine

□

Diagonalization

Lemma

The set of all Turing machines is countable

Diagonalization

Lemma

The set of all Turing machines is countable

Proof.

1. The set of all strings Σ^* (for any alphabet Σ) is countable
   - A list of all strings in Σ^* can be written down by listing all strings of length 0, length 1, length 2, ...
2. Each Turing machine M can be encoded as a string $\langle M \rangle$ over Σ
3. By omitting those strings in Σ^* which are not legal encodings of Turing machines, we get a list of all Turing machines

$\square$

Diagonalization

Lemma

The set of all languages over Σ is uncountable

Diagonalization

Lemma

The set of all languages over Σ is uncountable

Proof idea.

1. Each language over Σ can be represented by its characteristic sequence (an infinite binary sequence).

Diagonalization

Lemma

The set of all languages over Σ is uncountable

Proof idea.

1. Each language over Σ can be represented by its characteristic sequence (an infinite binary sequence).
2. Each infinite binary sequence can be seen as a characteristic sequence for a language over Σ. Hence, there is a one-to-one correspondence between infinite binary sequences and languages over Σ.

Diagonalization

Lemma

The set of all languages over Σ is uncountable

Proof idea.

1. Each language over Σ can be represented by its characteristic sequence (an infinite binary sequence).
2. Each infinite binary sequence can be seen as a characteristic sequence for a language over Σ. Hence, there is a one-to-one correspondence between infinite binary sequences and languages over Σ.
3. The set of infinite binary sequences is uncountable (by a simple diagonalization proof)

Diagonalization

Lemma

The set of all languages over Σ is uncountable

Proof idea.

1. Each language over Σ can be represented by its characteristic sequence (an infinite binary sequence).
2. Each infinite binary sequence can be seen as a characteristic sequence for a language over Σ. Hence, there is a one-to-one correspondence between infinite binary sequences and languages over Σ.
3. The set of infinite binary sequences is uncountable (by a simple diagonalization proof)
4. Hence, the set of all languages over Σ is uncountable

□

Diagonalization

Theorem

Some languages are not Turing-recognizable

Proof.

1. The set of all Turing machines is countable
2. The set of all languages is uncountable
3. Since each Turing machine recognize exactly one language, there are languages that are not recognized by any Turing machine

$\square$

Undecidability

Undecidability

David Hilbert (1862-1943)

Undecidability

Hilbert's 10th problem:
Construct an algorithm that given a polynomial, determine whether the polynomial has an integral root

Undecidability

Hilbert's 10th problem:
Construct an algorithm that given a polynomial, determine whether the polynomial has an integral root

Example

$6x^3yz^2 + 3xy^2 - x^3 - 10$

Undecidability

Hilbert's 10th problem:
Construct an algorithm that given a polynomial, determine whether the polynomial has an integral root

Example

$6x^3yz^2 + 3xy^2 - x^3 - 10$
root: $x = 5, y = 3, z = 0$

Undecidability

Hilbert's 10th problem:
Construct an algorithm that given a polynomial, determine whether the polynomial has an integral root

Example

$6x^3yz^2 + 3xy^2 - x^3 - 10$
root: $x = 5, y = 3, z = 0$

1970: Hilbert's 10th problem is undecidable!

Undecidability

Undecidability

- Software verification: Given a computer program and a specification of how the program is supposed to work, verify that the program performs as specified.

Undecidability

- Software verification: Given a computer program and a specification of how the program is supposed to work, verify that the program performs as specified.
  Impossible! This problem is not solvable by computers

Undecidability

- Construct a tool/debugger that can tell whether a Java program will go into an infinite loop on input *w*.

Undecidability

- Construct a tool/debugger that can tell whether a Java program will go into an infinite loop on input w.
  Impossible! This problem is undecidable

Undecidability

- Construct a tool/debugger that can tell whether a Java program will go into an infinite loop on input w.
  Impossible! This problem is undecidable
- There is no Turing machine U that given a Turing machine M and string w can determine whether M will loop on input w

Undecidability

$$A_{TM} = \{\langle M, w\rangle \mid M \textit{ is a Turing machine that accepts } w\}$$

Undecidability

$$A_{TM} = \{\langle M, w\rangle \mid M \textit{ is a Turing machine that accepts } w\}$$

Theorem

A_{TM} is Turing-recognizable

Undecidability

$$A_{TM} = \{\langle M, w \rangle \mid M \textit{ is a Turing machine that accepts } w\}$$

Theorem

A_{TM} is Turing-recognizable

Proof.

Let U be a Turing machine that works as follows:

1. Check that the input $\langle M, w \rangle$ is a legal encoding of a Turing machine M and string w (otherwise reject)
2. Simulate M on input w
3. If M enters the accept state, then U accepts $\langle M, w \rangle$
4. If M enters the reject state, then U rejects $\langle M, w \rangle$

Undecidability

$$A_{TM} = \{\langle M, w\rangle \mid M \textit{ is a Turing machine that accepts } w\}$$

Theorem

A_{TM} is Turing-recognizable

Proof.

Let U be a Turing machine that works as follows:

1. Check that the input $\langle M, w\rangle$ is a legal encoding of a Turing machine M and string w (otherwise reject)
2. Simulate M on input w
3. If M enters the accept state, then U accepts $\langle M, w\rangle$
4. If M enters the reject state, then U rejects $\langle M, w\rangle$

Note that U does not decide A_{TM}! □