Advanced Topics in Theoretical Computer Science

Part 1: Turing Machines and Turing Computability (2)

7.11.2013

Viorica Sofronie-Stokkermans

Universität Koblenz-Landau

e-mail: sofronie@uni-koblenz.de

Turing Machines

Overview: Turing Machines

- Accept languages of type 0.
- First memory: state (finite)
- Second memory: tape
  unlimited size; access at arbitrary place.
- Have a read/write head which can move left/right over the tape.
- Input word: initially on the tape.
  The machine can read it arbitrarily often.

Last time

- Deterministic Turing Machine (DTM)
- Configuration, transition between configurations, computation
  To halt, to hang

- Representation of Turing machines
  - as in definition
  - diagram (flow-chart) representation

Last time

- Definitions: TM-computable function

> - TM^{part} is the set of all partial TM-computable functions $f : \mathbb{N}^k \to \mathbb{N}$
> - TM is the set of all total TM-computable functions $f : \mathbb{N}^k \to \mathbb{N}$

Remark: Restrictions when defining TM and TM^{part}:

- Only functions over $\mathbb{N}$
- Only functions with values in $\mathbb{N}$ (not in $\mathbb{N}^m$)

This is not a real restriction:

Words from other domains can be encoded as natural numbers.

Last time

Types of Turing machines:

– Standard deterministic Turing Machines (Standard DTM)

– Other types of Turing machines:

- Tape infinite on both sides
- Several tapes
- Non-deterministic Turing machines

- For every TM with both sides infinite tape which computes a function f or accepts a language L, there exists a standard DTM $\mathcal{M}'$ which also computes f (resp. accepts L).
- For every k-DTM which computes a function f (or accepts a language L) there exists a DTM $\mathcal{M}'$ which computes f (resp. accepts L).

Last time

Universal Turing machines: TM which simulates other Turing machines

- Universal Turing machine $\mathcal{U}$ receives as input
  (i) the rules of an arbitrary TM $\mathcal{M}$ and
  (ii) a word w.
- $\mathcal{U}$ simulates $\mathcal{M}$, by always changing the configurations (according to the transition function δ) the way $\mathcal{M}$ would change them.

Problem: Turing machines take words (or numbers) as inputs. Can we encode an arbitraty Turing machine as a number or as a word?

Solution: Gödelisation
Method for assigning with every Turing machine a number or a word (Gödel number or Gödel word) such that the Turing machine can be effectively reconstructed from that number (or word).

Last time

- Acceptable language
- Recursively enumerable language
- Enumerable language
- Decidable language

relationships between these notions.

Last time

A DTM $\mathcal{M}$ decides a language L if

- for every input word $w \in L$, $\mathcal{M}$ halts with band contents Y (yes)
- for every input word $w \notin L$, $\mathcal{M}$ halts with band contents N (no)

L is called decidable if there exists a DTM which decides L.

Let L be a language over Σ_0 with $\#, Y, N \notin \Sigma_0$.
Let $\mathcal{M} = (K, \Sigma, \delta, s)$ be a DTM with $\Sigma_0 \subseteq \Sigma$.

- $\mathcal{M}$ enumerates L if there exists a state $q_B \in K$ (the blink state) such that: $L = \{w \in \Sigma_0^* \mid \exists u \in \Sigma^*; s, \underline{\#} \vdash_{\mathcal{M}}^* q_B, \#w\underline{\#}u\}$
- L is called recursively enumerable if there exists a DTM $\mathcal{M}$ which enumerates L.

Acceptable/Recursively enumerable/Decidable

Theorem (Acceptable = Recursively enumerable)
A language is recursively enumerable iff it is acceptable.

Proposition
Every decidable language is acceptable.

Proposition
The complement of any decidable language is decidable.

Proposition (Characterisation of decidability)
A language L is decidable iff L and its complement are acceptable.

Recursively enumerable = Type 0

Formal languages are of type 0 if they can be generated by arbitrary grammars (no restrictions).

Proposition

The recursively enumerable languages (i.e. the languages acceptable by DTMs) are exactly the languages generated by arbitrary grammars (i.e. languages of type 0).

Today

- Undecidable problems
- Ways of proving undecidability

Undecidability of the halting problem

$\mathcal{M}$ Turing machine $\mapsto G(\mathcal{M})$ Gödelisation

$$HALT = \{G(\mathcal{M})w \mid \mathcal{M} \text{ halts on input } w\}$$

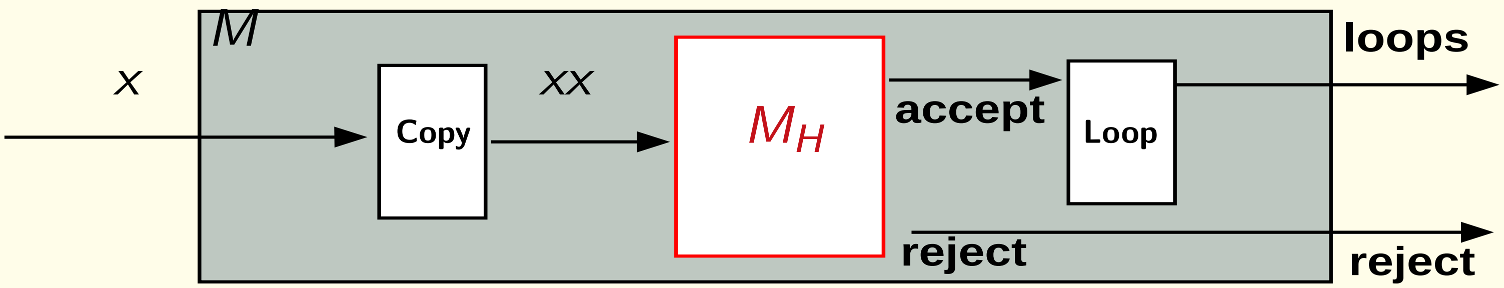
Is $HALT$ decidable?

Undecidability of the halting problem

Proposition: $HALT = \{G(\mathcal{M})w \mid \mathcal{M}$ halts on input $w\}$ is not decidable.

Proof: Assume, in order to derive a contradiction, that there exists a TM M_H which halts on every input and accepts only inputs in $HALT$.

We construct the following TM:



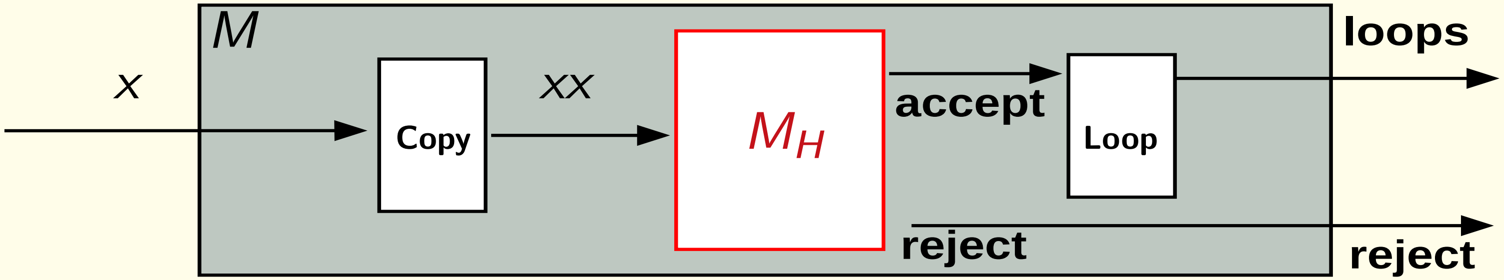
1. Let x be the input.
2. Copy the input. Let xx be the result.
3. Decide using M_H if $xx \in HALT$
4. If yes: write infinitely many 1s to the right.
5. If no: halt

Undecidability of the halting problem

Proposition: $HALT = \{G(\mathcal{M})w \mid \mathcal{M}$ halts on input $w\}$ is not decidable.

Proof: Assume, in order to derive a contradiction, that there exists a TM M_H which halts on every input and accepts only inputs in $HALT$.

What happens when we start M with input $G(M)$?



Case 1: M started with $G(M)$ halts: Then $G(M)G(M) \notin HALT$ Contradiction!

Fall 2: M started with $G(M)$ does not halt: Then $G(M)G(M) \in HALT$ Contradiction!

Undecidability proofs: Example

Theorem. $K = \{G(M) \mid M \text{ halts for input } G(M)\}$
is acceptable but undecidable.

Proof: Similar to the undecidability proof for the halting problem.

Exercise

Undecidability proofs: Example

Theorem. $K = \{G(M) \mid M \text{ halts for input } G(M)\}$
is acceptable but undecidable.

Proof: Similar to the undecidability proof for the halting problem.

Reformulation using numbers instead of words:

Gödelization $\mapsto$ Gödel numbers

Let $M_0, M_1, \ldots, M_n, \ldots$ be an enumeration of all Turing Machines

M_n is the TM with Gödel number n.

$$K = \{n \mid M_n \text{ halts on input } n\}$$

Undecidability proofs

Proof via reduction

Given:	L_1, L_2 languages
	L_1 known to be undecidable
To show:	L_2 undecidable

Idea:
Assume L_2 decidable. Let M_2 be a TM which decides L_2. Show that then we can construct a TM which decides L_1.

For this, we have to find a computable function f which transforms an instance of L_1 into an instance of L_2

$$\forall w(w \in L_1 \text{ iff } f(w) \in L_2)$$

Let M_f be the TM which computes f. Construct $M_1 = M_f M_2$. Then M_1 decides L_1.

Undecidability proofs: Example

Theorem. $H_0 = \{n \mid M_n \text{ halts for input } 0\}$ is undecidable.

Proof: We show that K can be reduced to H_0, i.e. that there exists a TM computable function $f : \mathbb{N} \to \mathbb{N}$ such that

$$i \in K \quad \text{iff} \quad f(i) \in H_0.$$

Only main idea here, we will come back to this example later

Undecidability proofs: Example

Theorem. $H_0 = \{n \mid M_n \text{ halts for input } 0\}$ is undecidable.

Proof: We show that K can be reduced to H_0, i.e. that there exists a TM computable function $f : \mathbb{N} \to \mathbb{N}$ such that $\quad i \in K \quad$ iff $\quad f(i) \in H_0$.

Want: $f(i) = j \quad$ iff $\quad$ (M_i halts for input i iff M_j halts for input 0).

For every i there exists a TM A_i s.t.: $s, \#\underline{\#} \vdash^*_{A_i} h, \#|^i\underline{\#}$.
Let M_K be the TM which accepts K.

We define $f(i) := j$ where $\;j$ is the Gödel number of $M_j = A_i M_K$.
f is TM computable. We show that f has the desired property:

$$
\begin{aligned}
f(i) = j \in H_0 \quad & \text{iff} \quad M_j = A_i M_K \text{ halts for input } 0 \ (\#\underline{\#}) \\
& \text{iff} \quad M_K \text{ halts for input } i \quad \text{iff} \quad i \in K.
\end{aligned}
$$