Theory of Computatio

Martin Čermák, Jiří Koutný and Alexander Meduna

Deparment of Information Systems
Faculty of Informatin Technology
Brno University of Technology, Faculty of Information Technology
Božetěchova 2, Brno 612 00, Czech Republic

Advanced Topics of Theoretical Computer Science

FRVŠ MŠMT FR2581/2010/G1

Part I

Theory of Computation

Recapitulation

- Usage of Turing Machines to demonstrate theoretical limits of computation.
- By the Turing-Church thesis, every procedure can be realized as a Turing Machine.
- The notion of a Turing Machine is exactly the right model of computation.

Basic idea

Some relatively simple functions and problems are beyond the limits of computation.

Definition

Let $M \in_{TM} \Psi$. The *function computed by M*, symbolically denoted by M–f, is defined over $\triangle^*$ as
$M\text{–}f = \{(x, y) | x, y \in \triangle^*, \rhd\blacktriangleright x\lhd \Rightarrow^* \rhd\blacksquare yu\lhd \text{ in } M, u \in \{\square\}^*\}$.

- Consider M–f, where $M \in_{TM} \Psi$, and an argument $x \in \triangle^*$.
- In a general, M–f is partial, so M–$f(x)$ may or may not be defined.
- If $M\text{–}f(x) = y$ is defined, M computes $\rhd\blacktriangleright x\lhd \Rightarrow^* \rhd\blacksquare yu\lhd$, where $u \in \{\square\}^*$.
- If M–$f(x)$ is undefined, M, starting from $\rhd\blacktriangleright x\lhd$, never reaches a configuration of the form $\rhd\blacksquare vu\lhd$, where $v \in \triangle^*$ and $u \in \{\square\}^*$, so it either rejects x or loops on x.

Definition

A function f is a *computable function* if there exists $M \in_{TM} \Psi$ such that $f = M$–f; otherwise, f is an incomputable function.

Integer Functions

- For every $M \in_{TM} \Psi$, $M\text{–}f$ is defined over $\triangle^*$, where $\triangle$ is an alphabet.
- We usually study numeric functions defined over sets of infinitely many numbers (such as $\mathbb{N}$).
- For Turing Machines, we need to represent numbers by strings over $\triangle$.
- We represent i in unary as $unary(i) = a^i$ for all $i \geq 0$.
- We automatically assume that $\triangle = \{a\}$ (because a is the only input symbol we need).

Definition

- Let g be a function over ${}_0\mathbb{N}$ and $M \in_{TM} \Psi$. *M computes g* iff $unary(g) = M\text{–}f$.
- A function h over ${}_0\mathbb{N}$ is a *computable function* if there is $M \in_{TM} \Psi$ such that M *computes* h; otherwise, h is an *incomputable function*.
- M computes an integer function g over ${}_0\mathbb{N}$ if this equivalence holds: $g(x) = y$ iff $(unary(x), unary(y)) \in M\text{–}f$, for all $x, y \in_0 \mathbb{N}$.

Convention

Whenever $M \in_{TM} \Psi$ works on an integer $x \in {}_0\mathbb{N}$, x is expressed as $unary(x)$. Instead of stating that M works on x represented as $unary(x)$, we just state that M works on x.

Example

Let g be the successor function defined as $g(i) = i + 1$ for all $i \geq 0$. Construct a Turing Machine M that computes $\triangleright\blacktriangleright a^i \triangleleft \Rightarrow^* \triangleright\blacksquare a^{i+1} \triangleleft$ so it moves across a^i to the right bounder $\triangleleft$, replaces it with $a\triangleleft$, and returns to the left to finish its computation in $\triangleright\blacksquare a^{i+1}\triangleleft$. As a result, M increases the number of as. Thus, M computes g.

- Function in the example is total.
- Suppose g is a function over ${}_0\mathbb{N}$, which is undefined for some arguments and let $M \in_{TM} \Psi$ compute g.
- For any $x \in {}_0\mathbb{N}$, $g(x)$ is undefined iff $(unary(x), unary(y)) \notin M\text{–}f$ for all $y \in {}_0\mathbb{N}$.

Example

Let g over $\mathbb{N}$ be a partial function as

- $g(x) = 2x$ if $x = 2n$, for some $n \in \mathbb{N}$,
- otherwise, $g(x)$ is undefined.

Construct $M \in_{TM} \Psi$ that computes g as follows.

INPUT: $\triangleright\blacktriangleright a^i\triangleleft$ for some $i \in \mathbb{N}$
change $\triangleright\blacktriangleright a^i\triangleleft$ to $\triangleright\blacktriangleright a^i A\triangleleft$
while current configuration $\triangleright\blacktriangleright a^i A^j\triangleleft$ satisfies $j \leq i$ **do**
  if $i = j$ **then**
    ACCEPT by computing $\triangleright\blacktriangleright a^i A^j\triangleleft \Rightarrow^* \triangleright\blacksquare a^i a^i\triangleleft$ (because $i = j = 2^m$ for some $m \in \mathbb{N}$)
  else
    compute $\triangleright\blacktriangleright a^i A^j\triangleleft \Rightarrow^* \triangleright\blacktriangleright a^i A^{2j}\triangleleft$ by changing each A to AA
  end if
end while
REJECT by computing $\triangleright\blacktriangleright a^i A^j\triangleleft \Rightarrow^* \triangleright\blacklozenge a^i \Box^j\triangleleft$ (because $j > i$, so $i \neq 2^m$ for any $m \in \mathbb{N}$).

Incomputable Functions

- The set of all rewriting systems is countable because each definition of a rewriting system is finite, so this set can be put into a bijection with $\mathbb{N}$.
- The set of all Turing Machines, which are defined as rewriting systems, is countable.
- The set of all functions is uncountable.
- Thus, there are more functions than Turing Machines.
- Some functions are incomputable.
- Even some simple total well-defined functions over $\mathbb{N}$ are incomputable.

Example

For every $k \in \mathbb{N}$, set
${}_kX = \{M \in {}_{TM}\Psi | card({}_MQ) = k + 1, L(M) \subseteq \{a\}^*\}$

Informally

- ${}_kX$ denotes the set of all Turing Machines in ${}_{TM}\Psi$ with $k+1$ states such that their languages are over $\{a\}$.
- Suppose that ${}_MQ = \{q_0, q_1, \ldots, q_k\}$ with $\blacktriangleright = q_0$ and $\blacksquare = q_k$.
- Let g be the function over $\mathbb{N}$ defined for every $i \in \mathbb{N}$ so $g(i)$ equals the greatest integer $j \in \mathbb{N}$ satisfying $\triangleright q_0 a \triangleleft \Rightarrow^* \triangleright q_i a^j u \triangleleft$ in M with $M \in {}_jX$ where $u \in \{\square\}^*$.
- For every $i \in \mathbb{N}$, ${}_iX$ is finite.
- ${}_iX$ always contains $M \in {}_{TM}\Psi$ such that $\triangleright q_0 a \triangleleft \Rightarrow^* q_i a^j u \triangleleft$ in M with $j \in \mathbb{N}$, so g is total.
- $g(i)$ is defined quite rigorously because each Turing Machine in ${}_iX$ is deterministic.
- But, g is incomputable.

Proof idea (based upon diagonalization)

- Assume that g is computable.
- Thus, ${}_{TM}\Psi$ contains a Turing Machine M that computes g.
- Convert M to a Turing Machine N, which we subsequently transform to a Turing Machine O.
- Demonstrate that O performs a computation that contradicts the definition of g.
- So our assumption that g is computable is incorrect.
- Thus, g is incomputable.

Convention

In the sequel, ζ denotes some fixed enumeration of all possible Turing Machines,

$$\zeta =_1 M, {}_2M, \ldots$$

Regarding ζ, we just require the existence of two algorithms

- Translation of every $i \in \mathbb{N}$ to ${}_iM$,
- Translation of every $M \in_{TM} \Psi$ to i so $M =_i M$, where $i \in (N)$.

Let

$$\xi = \_1 M\text{–}f, {}_2M\text{–}f, \ldots$$

That is, ξ corresponds to ζ so ξ denotes the enumeration of the functions computed by the machines listed in ζ. The positive integer i of ${}_iM\text{–}f$ is referred to as the index of ${}_iM\text{–}f$; in terms of ζ, i is referred to as the index of ${}_iM$.

References

- Wayne Goddard.
  Introducing the Theory of Computation.
  Jones Bartlett Publishers, 2008.
- Jeffrey D. Ullman John E. Hopcroft, Rajeev Motwani.
  Introduction to Automata Theory, Languages, and Computation.
  Addison Wesley, 2006.
- Dexter C. Kozen.
  Automata and Computability.
  Springer, 2007.
- Dexter C. Kozen.
  Theory of Computation.
  Springer, 2010.
- John C. Martin.
  Introduction to Languages and the Theory of Computation.
  McGraw-Hill Science/Engineering/Math, 2002.

Thank you for your attention!

End