Universal Turing Machines

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

BRNO UNIVERSITY OF TECHNOLOGY
FIT
FACULTY OF INFORMATION TECHNOLOGY

Advanced Topics of Theoretical Computer Science

FRVŠ MŠMT FR2581/2010/G1

Part I

Universal Turing Machines

There exists a Turing Machine acting as such a universal device, which simulates all machines in ${}_{TM}\Psi$.

Universal Turing Machine $U \in_{TM}\Psi$

Universal Turing Machine $U \in_{TM}\Psi$ simulates every $M \in_{TM}\Psi$ working on any input w.

- The input of any Turing Machine is always a string.
- How to encode every $M \in_{TM}\Psi$ as a string (denoted as $\langle M \rangle$)?

Pinciple

- U has the code of M followed by the code of w as its input (denoted as $\langle M, w \rangle$).
- U decodes M and w to simulate M working on w.
- U accepts $\langle M, w \rangle$ iff M accepts w.

Encoding Mathematically

The encoding should represent a total function *code* from ${}_{TM}\Psi$ to ϑ^* such that $code(M) = \langle M \rangle$ for all $M \in_{TM} \Psi$.
The decoding *decode* of Turing Machines is defined on an arbitrary but fixed $O \in_{TM} \Psi$, so

- for every $x \in range(code)$, $decode(x) = inverse(code(M))$.
- for every $y \in \vartheta^* - range(code)$, $decode(y) = O$ so $range(decode) =_{TM} \Psi$.

- *decode* is a total surjection (it maps every string in ϑ^*),
- *decode* may not be an injection (several strings in ϑ^* may be decoded to the same machine in ${}_{TM}\Psi$),
- *code* and *decode* are used to encode and decode the pairs consisting of Turing Machines and input strings.

We just require that the mechanical interpretation of both *code* and *decode* is relatively easily performable.

A Binary Code for Turing Machines

- Consider any $M \in_{TM} \Psi$.
- Rename states in Q to $q_1, q_2, q_3, q_4, \dots, q_m$ so $q_1 = ▶$, $q_2 = ■$, $q_3 = ♦$, where $m = card(Q)$.
- Rename the symbols of $\{\triangleright, \triangleleft\} \cup \Gamma$ to $a_1, a_2, \dots, a_n$ so $a_1 = \triangleright, a_2 = \triangleleft, a_3 = \square$, where $n = card(\Gamma)$.
- Introduce the homomorphism h from $Q \cup \Gamma$ to $\{0, 1\}^*$ as $h(q_i) = 10^i$, $1 \leq i \leq m$, and $h(a_j) = 110^j$, $1 \leq j \leq n$.
- Extend h so it is defined from $(\Gamma \cup Q)^*$ to $\{0, 1\}^*$
    - $h(\varepsilon) = \varepsilon$,
    - $h(X_1 \dots X_k) = h(X_1) \dots h(X_k)$, where $k \geq 1$, $X_l \in \Gamma \cup Q$, $1 \leq l \leq k$.
- Define the mapping $code$ from R to $\{0, 1\}^*$ so that for each rule $r : x \to y \in R$, $code(r) = h(xy)$.
- Write the rules of R in an order as $r_1, r_2, \dots, r_o$ with o = card(R) (for instance, order them lexicographically).
- Set $code(R) = code(r_1)111code(r_2)111code(r_o)111$.
- From $code(R)$, we obtain $code(M)$ by setting $code(M) = 0^m10^n1code(R)1$.

A Binary Code for Turing Machines

Let $code(M) = 0^m 1 0^n 1 code(R) 1$

- $0^m 1$ states that $m = card(Q)$,
- $0^n 1$ state that $n = card(\Gamma)$,
- $code(R)$ encodes the rules of R.

Mapping *code* is total, but *inverse*(*code*) is partial.

- Select an arbitrary but fixed $O \in_{TM} \Psi$,
- Extend *inverse*(*code*) to the total mapping *decode* so that for every $x \in \{0, 1\}^*$:
  - if x is a legal code of K in ${}_{TM}\Psi$, $decode(x) = K$,
  - otherwise, $decode(x) = O$.

For $w \in \triangle^*$, $code(w) = h(w)$

- Select an arbitrary but fixed $y \in \triangle^*$,
- Define the total surjection *decode* so for every $x \in \{0, 1\}^*$
  - if $x \in range(code)$, $decode(x) = inverse(code(w))$,
  - otherwise, $decode(z) = y$.

For every $(M, w) \in_{TM} \Psi \times \triangle^*$, define $code(M, w) = code(M)code(w)$

- *code* is a total function,
- Define the total surjection *decode* so
  - $decode(xy) = decode(x)decode(y)$,
  - where $decode(x) \in_{TM} \Psi$ and $decode(y) \in \triangle^*$.

Example

Consider Turing Machine $M = (\Sigma, R) \in_{TM} \Psi$, where
$\Sigma = Q \cup \Gamma \cup \{\triangleright, \triangleleft\}$, $Q = \{\blacktriangleright, \blacksquare, \blacklozenge, A, B, C, D\}$, $\Gamma = \triangle \cup \{\square\}, \triangle = \{b\}$,
and R contains these rules

$$
\begin{array}{rcl@{\qquad}rcl}
\blacktriangleright\triangleleft & \rightarrow & \blacksquare\triangleleft, & \blacktriangleright b & \rightarrow & bA, \\
Ab & \rightarrow & bB, & Bb & \rightarrow & bA, \\
A\triangleleft & \rightarrow & C\triangleleft, & B\triangleleft & \rightarrow & D\triangleleft, \\
bD & \rightarrow & D\square, & bC & \rightarrow & C\square, \\
\triangleright C & \rightarrow & \triangleright\blacklozenge, & \triangleright D & \rightarrow & \triangleright\blacksquare
\end{array}
$$

$$L(M) = \{bi | i \geq 0, i \text{ is even}\}$$

Homomorphism h from $Q \in \{\triangleright, \triangleleft\} \cup \Gamma$ to $\{0, 1\}^*$:

- $h(q_i) = 10^i, 1 \leq i \leq 7$, where $q_1, q_2, q_3, q_4, q_5, q_6$, and q_7 coincide with $\blacktriangleright, \blacksquare, \blacklozenge, A, B, C, D$, respectively,
- $h(a_i) = 110^j, 1 \leq j \leq 4$, where a_1, a_2, a_3, and a_4 coincide with $\triangleright, \triangleleft, \square$, and b, respectively.

Extend h so it is defined from $(Q \cup \{\triangleright, \triangleleft\} \cup \Gamma)^*$ to $\{0, 1\}^*$.

A Binary Code for Turing Machines

Example

Based on h, define the mapping *code* from R to $\{0, 1\}^*$ so for each rule $x \to y \in R$, $code(x \to y) = h(xy)$ (for example, $code(\blacktriangleright b \to bA) = 10110000110000010000$).

Take the above order of the rules from R, and set

$$code(R) = code(\blacktriangleright\triangleleft \to \blacksquare\triangleleft)111 \ldots code(\triangleright D \to \triangleright\blacksquare)111$$

Finally, $code(M) = 0^7 10^2 1 code(R) 1$.
For instance, take $w = bb$, and set $code(bb) = 110000110000$.
Thus, $code(M, w) = 0^7 10^2 1 code(R) 1111110000110000 = \ldots$

Convention

- We suppose there exist a fixed encoding and a fixed decoding of all Turing Machines in ${}_{TM}\Psi$.
- Both *code* and decode have to be uniquely and mechanically interpretable (not necessarily binary).

Construction of Universal Turing Machines

Universal Turing Machine ${}_{Accept}U$ simulates every $M \in_{TM} \Psi$ on $w \in \triangle^*$ so ${}_{Accept}U$ accepts $\langle M, w \rangle$ iff M accepts w.

Universal Turing Machine ${}_{Accept}U$

$$L({}_{Accept}M) = \{\langle M, w \rangle | M \in_{TM} \Psi, w \in \triangle^*, w \in L(M)\}$$

Universal Turing Machine ${}_{Halt}U$ simulates every $M \in_{TM} \Psi$ on $w \in \triangle^*$ so ${}_{Halt}U$ accepts $\langle M, w \rangle$ iff M halts on w.

Universal Turing Machine ${}_{Halt}U$

$$L({}_{Halt}M) = \{\langle M, w \rangle | M \in_{TM} \Psi, w \in \triangle^*, M \text{ halts on } w\}$$

Convention

${}_{Accept}U$ works on $\langle M, w \rangle$ so it first interprets $\langle M, w \rangle$ as M and w; then, it simulates the moves of M on w

is simplified to

${}_{Accept}U$ runs M on w.

Theorem

There exists ${}_{Accept}U \in_{TM} \Psi$ such that $L({}_{Accept}U) = {}_{Accept}L$.

Proof. On every input $\langle M, w\rangle$, ${}_{Accept}U$ works so it runs M on w. ${}_{Accept}U$ accepts $\langle M, w\rangle$ if and when it finds out that M accepts w; otherwise, ${}_{Accept}U$ keeps simulating the moves of M in this way.

Theorem

There exists ${}_{Halt}U \in_{TM} \Psi$ such that $L({}_{Halt}U) = {}_{Halt}L$.

Proof. On every input $\langle M, w\rangle$, ${}_{Halt}U$ works so it runs M on w. ${}_{Halt}U$ accepts $\langle M, w\rangle$ M if M halt w; which means that M either accepts or rejects w. Thus, ${}_{Halt}U$ loops on $\langle M, w\rangle$ iff M loops on w. Observe that $L({}_{Halt}U) = {}_{Halt}\ L$.

No Turing Machine can halt on every input and, simultaneously, act as a universal Turing Machine.

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