Church-Turing Thesis and Turing Machine

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
FACULTY OF INFORMATION TECHNOLOGY

Advanced Topics of Theoretical Computer Science

FRVŠ MŠMT FR2581/2010/G1

Part I

Church-Turing Thesis and Turing Machine

Church-Turing Thesis

The intuitive notion of a procedure is functionally identifiable with the formal notion of a Turing Machine.

Church-Turing Thesis (Alonzo Church[1] in 1936)

- makes Turing Machine exceptionally significant,
- makes effective procedure central to computation as a whole,
- needs formalization of effective procedure to a formal model,
- assures Turing Machine as suitable model.

Turning Machine

- is relatively simple language-defining model,
- obviously constitutes a procedure,
- formalizes every procedure in the intuitive sense.

[1] (⋆ June 14, 1903 - † August 11, 1995)

Church-Turing Thesis is really a thesis

Church-Turing Thesis

Church-Turing Thesis is not a theorem because it cannot be proved.

Why?

- Proof necessitate rigorous comparsion of a procedure with a Turing Machine.
- A formalization of the notion of a procedure is necessary.
- Is this newly formalized notion equivalent to the intuitive notion of a procedure?
- Attempt to prove this thesis ends up with an infinite regression.

The evidence supporting the Church-Turing thesis:

- Formalization of the notion of a procedure in the intuitive sense by other mathematical models equivalent with Turing Machines.
- Nobody has ever come with a procedure in the intuitive sense and demonstrated that no Turing Machine can formalize it.

Part II

Turing Machines and Their Languages

The Turing Machine generalizes the finite automaton in three ways

- it can read and write on its tape,
- its head can move both to the right and to the left on the tape,
- the tape can be limitlessly extended to the right.

Turing Machine

Turing Machine is a rewriting system $M = (\Sigma, R)$, where:

- Σ contains subalphabets $Q, F, \Gamma, \triangle, \{\rhd, \lhd, \Box\}$ such that $\Sigma = Q \cup \Gamma \cup \{\rhd, \lhd\}$, $F \subseteq Q$, $\triangle \subset \Gamma$, $\Box \in \Gamma - \triangle$, and $\{\rhd, \lhd\}$, Q, Γ are pairwise disjoint,
- R is a finite set of rules of the form $x \rightarrow y$ satisfying
  - $\{x, y\} \subseteq \{\rhd\}Q$, or
  - $\{x, y\} \subseteq \Gamma Q \cup Q\Gamma$, or
  - $x \in \{Q\}\{\lhd\}$ and $y \in \{Q\}\{\Box\lhd, \lhd\}$.

Q, F, Γ and $\triangle$ are referred to as the *set of states*, the *set of final states*, the *alphabet of tape symbols*, and the *alphabet of input symbols*, respectively. Q containst the *start state* denoted by $\blacktriangleright$.

- Relations $\Rightarrow$, $\Rightarrow^n$ for $n \geq 0$, $\Rightarrow^+$, and $\Rightarrow^*$ are defined as usual.
- *M* accepts $w \in \triangle^*$ if $\triangleright \blacktriangleright w \triangleleft \Rightarrow^* \triangleright ufv \triangleleft$ in M, where $u, v \in \Gamma^*$, $f \in F$.

Language of a Turing Machine

$L(M) = \{w | w \in \triangle^*, \triangleright \blacktriangleright w \triangleleft \Rightarrow^* \triangleright ufv \triangleleft, u, v \in \Gamma^*, f \in F\}$

Informally, $L(M)$ is defined as the set of all strings that M accepts.
Notation:

- *Configuration* of M is a string of the form $\triangleright uqv \triangleleft$, $u, v \in \Gamma^*, q \in Q$,
- ${}_M X$ denote the set of all configurations of M,
- $\triangleright, \triangleleft$ are referred to as the *left* and *right* bounders, respectively.

How to understand $\triangleright uqv \triangleleft$ in M

- uv is on the tape of M,
- q is the current state of M,
- *head* of M is over the leftmost symbol of $v \triangleleft$.

How to understand $\beta \Rightarrow \chi$ in M

- $\beta, \chi \in_M X$,
- M makes a *move* or a *computational step* from β to χ.

How to understand $\beta \Rightarrow^* \chi$ in M

- $\beta, \chi \in_M X$,
- M makes a *computation* from β to χ.

How to understand $q \triangleright \rightarrow p \square \triangleleft \in R$

- $p, q \in Q$,
- extend the tape by inserting $\square$, called a *blank*, in front of $\triangleleft$,
- formally, $\triangleright uq \triangleleft \Rightarrow \triangleright up \square \triangleleft$, $u \in \Gamma^*$.

Let $L = \{x | x \in \{a, b, c\}^*, occur(x, a) = occur(x, b) = occur(x, c)\}$.

Example (Turing Machine M such that $L(M) = L$)

$M = (\Sigma, R)$, where

- $\Sigma = Q \cup \Gamma \cup \{\triangleright, \triangleleft\}$, with $\Gamma = \triangle \cup \{\square\}$
- $\triangle = \{a, b, c\}$
- $Q = \{\blacktriangleright, \lrcorner, \blacksquare\} \cup W$, with $W = \{\langle O \rangle | O \subseteq \{a, b, c\}\}$
- $F = \{\blacksquare\}$

Construct R by performing

1. add $\triangleright\blacktriangleright \rightarrow \triangleright\langle\{\}\rangle$ to R,
2. for every $\langle O \rangle \in W$ and every $d \in \triangle \cup \{\square\}$, add $\langle O \rangle d \rightarrow d\langle O \rangle$ and $d\langle O \rangle \rightarrow \langle O \rangle d$ to R,
3. for every $\langle O \rangle \in W$ such that $O \subset \{a, b, c\}$ and every $d \in \triangle - O$, add $\langle O \rangle d \rightarrow \langle O \cup \{d\}\square$ to R,
4. add $\langle\{a, b, c\}\rangle d \rightarrow \langle\{\}\rangle d$ to R, where $d \in \triangle \cup \{\square, \triangleleft\}$,
5. add $\langle\{\}\rangle\triangleleft \rightarrow \lrcorner\triangleleft, \square\lrcorner \rightarrow \lrcorner\square$, and $\triangleright\lrcorner \rightarrow \triangleright\blacksquare$ to R.

Principle of computation

- M starts every computation by (1),
- M moves on its tape by (2),
- M adds the input symbol into its current state from *power*($\triangle$) and changes the symbol to $\Box$ on the tape by (3),
- M empties $\{a, b, c\}$ so it changes this state to the state equal to the empty set by (4),
- M makes a final scan of the tape from $\lhd$ to $\rhd$, if the tape is completely blank, M accepts by (5).

Example (M accepts *babcca*)

$\rhd\blacktriangleright babcca\lhd \Rightarrow \rhd\langle\{\}\rangle babcca\lhd \Rightarrow^* \rhd babc\langle\{\}\rangle ca\lhd \Rightarrow$
$\rhd babc\langle\{c\}\rangle\Box a\lhd \Rightarrow^* \rhd ba\langle\{c\}bc\Box a\lhd \Rightarrow \rhd ba\langle\{b, c\}\rangle\Box c\Box a\lhd \Rightarrow^*$
$\rhd ba\Box c\Box\langle\{b, c\}\rangle a\lhd \Rightarrow \rhd ba\Box c\Box\langle\{a, b, c\}\rangle\Box\lhd \Rightarrow^*$
$\rhd b\langle\{a, b, c\}\rangle a\Box c\Box\Box\lhd \Rightarrow \rhd b\langle\{\}\rangle a\Box c\Box\Box\lhd \Rightarrow^* \rhd\Box\Box\Box\Box\Box\Box\langle\{\}\rangle\lhd \Rightarrow^*$
$\rhd\Box\Box\Box\Box\Box\Box\hookleftarrow\lhd \Rightarrow \rhd\Box\Box\Box\Box\Box\hookleftarrow\Box\lhd \Rightarrow^* \rhd\hookleftarrow\Box\Box\Box\Box\Box\Box\lhd \Rightarrow^* \rhd\blacksquare\Box\Box\Box\Box\Box\Box\lhd$

M accepts the same string in many other ways, thus M is *non-deterministic* rewriting system.

Turing Machines and Their Languages

Strictly formal definition of a Turing Machine

- the most detailed and rigorous description,
- tends to be lenghty and tedious,
- difficult and time consuming to figure out the way the Turing Machine accepts its language.

Informal description of a Turing Machine

- describes Turing Machines as procedures,
- omites various details concerning their components.

Formal vs. informal description of Turing Machines

Turing-Church thesis makes both ways of description perfectly legitimate because it assures us every procedure is identifiable with a Turing Machine defined in a strictly mathematical way.

The translation from informal description to the corresponding strictly formal description

- is a straightforward task,
- is usually lengthy and tedious.

Turing Machine as a Pascal-like procedure (explain the changes,omit the states and rules).

Example (Turing Machine M, $L(M) = \{a^i | i \text{ is a prime number}\}$)

INPUT: a^i with $i \in \mathbb{N}$
if $i \leq 1$ **then**
REJECT
end if
change a^i to AAa^{i-2}
while $A^k a^h$ occurs with $k \leq h$ and $i = k + h$ **do**
change $A^k a^h$ to the unique string y with $i = |y|$ and $y \in A^k\{a^k A^k\}^* z$ with $z \in prefix(a^k A^{k-1})$;
if $|z| = 0$ or $|z| = k$ **then**
REJECT
else
change y to $A^{k+1} a^{h-1}$
end if
end while
ACCEPT.

Idea of computation

- i is not prime iff $y = A^k a^k A^k \ldots a^k A^k$ (i is divisible by k, so M rejects a^i),
- if $y = A^k a^k A^k \ldots a^k A^k z$ such that $z \in prefix(a^k A^{k-1}) - \{\varepsilon, a^k\}$, then i is a prime and M accepts.

Notes

- The test $A^k a^h$ with $k \leq h$ and $i = k + h$ can be reformulated to its strictly formal description, but it is a tedious task.
- A strictly mathematical definition of the other parts of M is lengthy as well.
- We just use English prose to describe procedures representing Turing Machines.

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