Computability, Complexity and Logic (Theory 3)

2024/25

Detlef Plump and Simon Foster

Part I

Organization and Literature

Organization

Two parts

- Week 1 - 6: Computability and complexity (Detlef Plump)
- Week 7 - 11: Logic and verification (Simon Foster)

Weekly work

- Attending lectures and reading in textbooks
- Working on exercises both in problem classes/practicals and privately
- Week 1 - 10: Problem classes (supervised)
- Week 7 - 9 & 11: Practicals in the software labs (supervised)

Teaching assistants

- Jeremy Jacob and Christian Pardillo Laursen

Assessments

- Formative tests in Week 6 and Week 10 which will be marked
- 4-hour closed/in-person exam

Literature

Main textbooks (∗∗∗∗)

- John C. Martin: *Introduction to Languages and the Theory of Computation*, McGraw-Hill, 4th edition, 2010
- Michael Huth and Mark Ryan: *Logic in Computer Science: Modelling and Reasoning About Systems*, Cambridge University Press, 2nd edition, 2004

Additional reading

- James L. Hein: *Discrete Structures, Logic, and Computability*, Jones & Bartlett Learning, 4th edition, 2015 (∗∗∗)
- Michael Sipser: *Introduction to the Theory of Computation*, Cengage Learning, 3rd edition, 2012 (∗∗)
- Elaine Rich: *Automata, Computability and Complexity*, Pearson, 2007 (∗∗)

Part II

Formal Languages

Formal languages (Section 1.4 in ILTC)

- An *alphabet* is a nonempty finite set (denoted by Σ, Γ, ...) of *symbols*.

  Examples: $\Sigma = \{0, 1\}$ and $\Gamma = \{a, b, \ldots, z\}$.

- The *set of all strings* over an alphabet Σ is denoted by Σ^*. The *empty string* is denoted by Λ (other authors use λ or ϵ).

- A *formal language* over Σ is a subset of Σ^*.

  Examples:

  $\{\Lambda, 0, 00, 001\}$ and $\{w \in \{0,1\}^* \mid |w| = 2^n \text{ for some } n \geq 0\}$

  are languages over $\{0, 1\}$.

Formal languages (cont'd)

Notation

For $a \in \Sigma$, $w \in \Sigma^*$, $L \subseteq \Sigma^*$ and $k \geq 1$,

$$a^k = aa \cdots a, \quad w^k = ww \cdots w, \quad L^k = L \cdots L$$

where in each case there are k factors on the right-hand side.

Special case $k = 0$: $a^0 = w^0 = \Lambda$ and $L^0 = \{\Lambda\}$.

Kleene star: For a language L,

$$L^* = \bigcup_{i=0}^{\infty} L^i \quad \text{and} \quad L^+ = \bigcup_{i=1}^{\infty} L^i.$$

Note: $L^* = L^+$ if and only if $\Lambda \in L$.

Grammars (Section 8.3 in ILTC)

A *grammar* (or *unrestricted grammar*) is a 4-tuple $G = (V, \Sigma, S, P)$ where

- V is the finite set of *nonterminal symbols* (or *variables*),
- Σ is the alphabet of *terminal symbols*, where $\Sigma \cap V = \emptyset$,
- $S \in V$ is the *start symbol*,
- P is the finite set of *productions* $\alpha \to \beta$, where $\alpha \in (V \cup \Sigma)^+$ and $\beta \in (V \cup \Sigma)^*$.

Remark

- ILTC requires that the left-hand side α of a production contains at least one variable. Our definition is more flexible.

Grammars (Section 8.3 in ILTC)

A *grammar* (or *unrestricted grammar*) is a 4-tuple $G = (V, \Sigma, S, P)$ where

- V is the finite set of *nonterminal symbols* (or *variables*),
- Σ is the alphabet of *terminal symbols*, where $\Sigma \cap V = \emptyset$,
- $S \in V$ is the *start symbol*,
- P is the finite set of *productions* $\alpha \to \beta$, where $\alpha \in (V \cup \Sigma)^+$ and $\beta \in (V \cup \Sigma)^*$.

Remark

- ILTC requires that the left-hand side α of a production contains at least one variable. Our definition is more flexible.

The *language generated by* G is the set of all terminal strings derivable from the start symbol:

$$\mathrm{L}(G) = \{w \in \Sigma^* \mid S \Rightarrow^* w\}.$$

Example (unrestricted grammar)

$G = (\{\mathtt{S}\}, \{\mathtt{a}, \mathtt{b}\}, \mathtt{S}, P)$ with

$$P = \begin{cases} \mathtt{S} & \to & \Lambda \mid \mathtt{aaSb} \\ \mathtt{ab} & \to & \mathtt{ba} \end{cases}$$

Example (unrestricted grammar)

$G = (\{\mathtt{S}\}, \{\mathtt{a}, \mathtt{b}\}, \mathtt{S}, P)$ with

$$P = \begin{cases} \mathtt{S} & \to & \Lambda \mid \mathtt{aaSb} \\ \mathtt{ab} & \to & \mathtt{ba} \end{cases}$$

- Derivation of baaaab:
$$\begin{aligned} \mathtt{S} &\Rightarrow \mathtt{aaSb} \\ &\Rightarrow \mathtt{aaaaSbb} \\ &\Rightarrow \mathtt{aaaabb} \\ &\Rightarrow \mathtt{aaabab} \\ &\Rightarrow \mathtt{aabaab} \\ &\Rightarrow^2 \mathtt{baaaab} \end{aligned}$$
- $\mathrm{L}(G) = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \text{ contains twice as many a's as b's}\}$

The Chomsky hierarchy (Section 8.4 in ILTC)

Type	Grammars/ Languages	Grammar productions	Machines
0	*Unrestricted/ semidecidable*	$\alpha \to \beta$ $[\alpha \in (V \cup \Sigma)^+,$ $\beta \in (V \cup \Sigma)^*]$	*Turing machine* (deterministic or nondeterministic)
1	*Context-sensitive*	$\alpha \to \beta$ $[\alpha, \beta \in (V \cup \Sigma)^+,$ $\lvert\alpha\rvert \leq \lvert\beta\rvert]$	*Linear-bounded automaton*
2	*Context-free*	$A \to \beta$ $[A \in V,\ \beta \in (V \cup \Sigma)^*]$	*Pushdown automaton*
3	*Regular*	$A \to aB,\ A \to \Lambda$ $[A, B \in V,\ a \in \Sigma]$	*Finite automaton* (deterministic or nondeterministic)

Noam Chomsky (born 1928)

Part III

Computability

Hilbert's Entscheidungsproblem

David Hilbert
(1862 - 1943)

In 1928, the famous mathematician David Hilbert formulated the *Entscheidungsproblem* (German for "decision problem"): is there an algorithm that takes as input a formula of first-order logic and correctly decides whether the formula is universally valid or not.

This was negatively answered by Alan Turing (and others) in 1936. To do this, Turing needed to formalize the notion of an *algorithm*.

In the second part of this module, we will prove that the Entscheidungsproblem is unsolvable.

Turing machines (Chapter 7 in ILTC)

Introduced by Alan Turing in "On Computable Numbers with an Application to the Entscheidungsproblem", *Proceedings of the London Mathematical Society (Series 2)* 42, pages 230–265, 1936

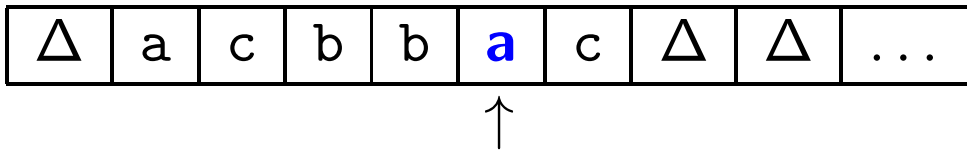
A *Turing machine* (TM) is a 5-tuple $(Q, \Sigma, \Gamma, q_0, \delta)$ where

- Q is the finite set of *states*, including $\mathrm{h_a}$ and $\mathrm{h_r}$, the *halting states*,
- Σ is the *input alphabet*,
- Γ is the *tape alphabet*, including the *blank symbol* Δ, where $\Sigma \subseteq \Gamma - \{\Delta\}$,
- $q_0 \in Q$ is the *initial state*,
- $\delta\colon (Q - \{\mathrm{h_a}, \mathrm{h_r}\}) \times \Gamma \to Q \times \Gamma \times \{\mathrm{L}, \mathrm{R}, \mathrm{S}\}$ is the *transition function*.

(The definition in ILTC is slightly different but equivalent.)

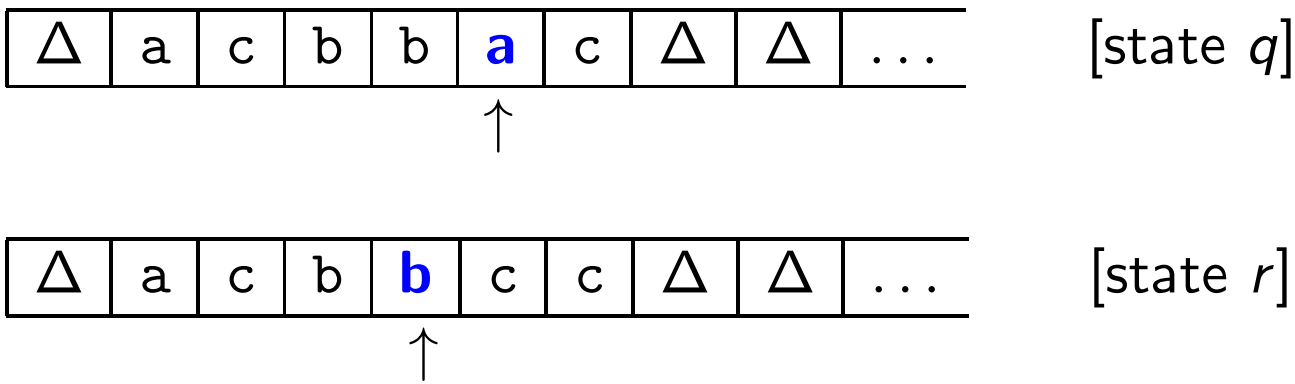
Tape and tape head

Intuitively, a TM moves a tape head on a infinite tape of squares:



A move $\delta(q, a) = (r, b, D)$ takes place if the machine is in state q and reads symbol a: the machine enters state r, overwrites a with b, and moves its head one square to the left ($D = \mathrm{L}$), to the right ($D = \mathrm{R}$), or leaves it stationary ($D = \mathrm{S}$).

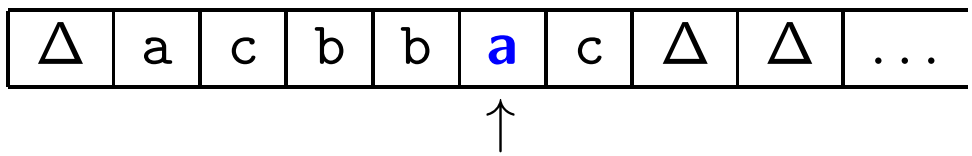
For example, if $\delta(q, \mathtt{a}) = (r, \mathtt{c}, \mathrm{L})$:



Configurations and transitions

A *configuration* is a string uqv such that $u \in \Gamma^*$, $q \in Q$ and $v \in \Gamma^+$, where we assume $Q \cap \Gamma = \emptyset$. It describes the situation in which uv is the contents of an initial portion of the tape, the TM is in state q, the tape head reads the first symbol of v, and all squares to the right of uv are blank.

For example,



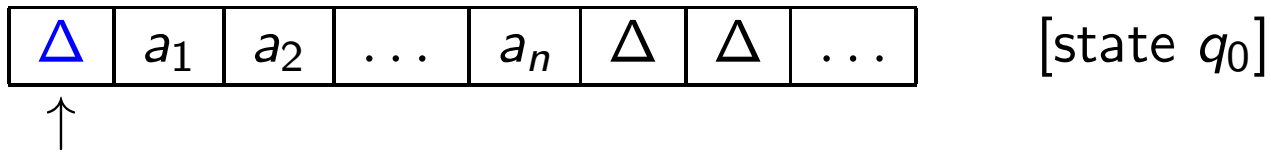
is represented by the configuration $\Delta\mathtt{acbb}q\mathtt{ac}$.

The configurations uqv, $uqv\Delta$, $uqv\Delta\Delta$, ... are considered to be the same; that is, rightmost blanks can be ignored.

A *transition* $uqv \vdash xry$ describes the effect of a single move of a TM. We write $uqv \vdash^* xry$ for a sequence of transitions.

Special configurations

The *initial configuration* for an input $w \in \Sigma^*$ is $q_0 \Delta w$:



where $w = a_1 a_2 \ldots a_n$.

A configuration $u\,\mathrm{h_a} v$ is an *accepting configuration* and $u\,\mathrm{h_r} v$ is a *rejecting configuration*. Both are *halting configurations*.

Crash at the start of the tape: we define $qav \vdash \mathrm{h_r} av$ if $\delta(q, a) = (r, b, \mathrm{L})$ for some $r \in Q$ and $b \in \Gamma$.

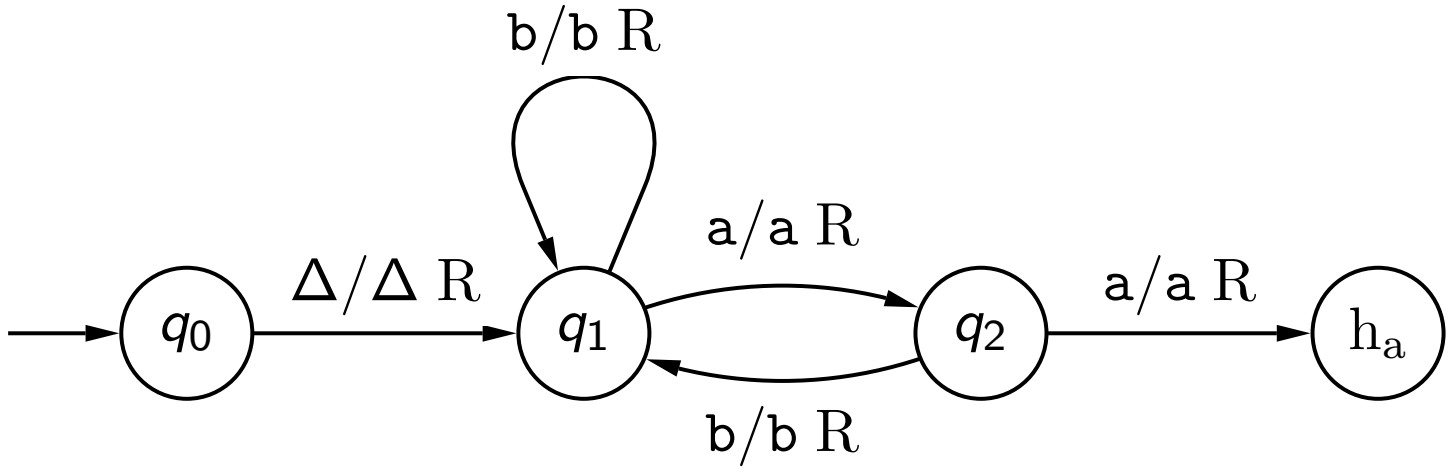
Language accepted by a Turing machine

The *language accepted by* a TM M is

$$\mathrm{L}(M) = \{w \in \Sigma^* \mid q_0 \Delta w \vdash^* u\,\mathrm{h_a} v \text{ for some } u, v \in \Gamma^*\}.$$

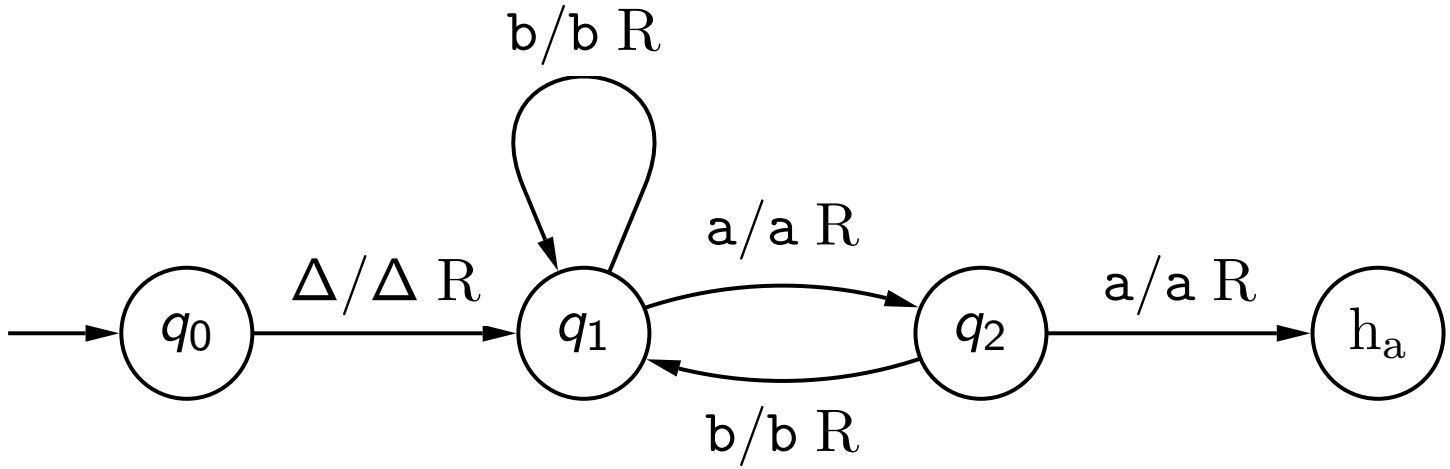
Example (Turing machine)

M:



Example (Turing machine)

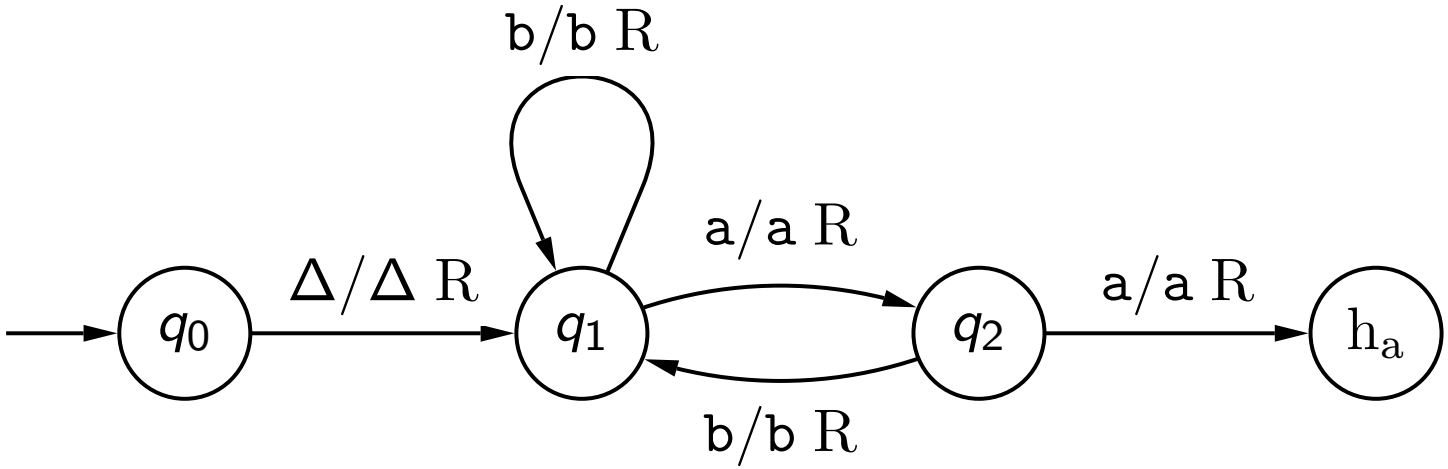
M:



$$\begin{aligned} L(M) &= (\{b\}^*\{ab\}^*)^*\{aa\}\{a,b\}^* \quad \text{for } \Sigma = \{a,b\} \\ &= \{a,b\}^*\{aa\}\{a,b\}^* \end{aligned}$$

Example (Turing machine)

M:

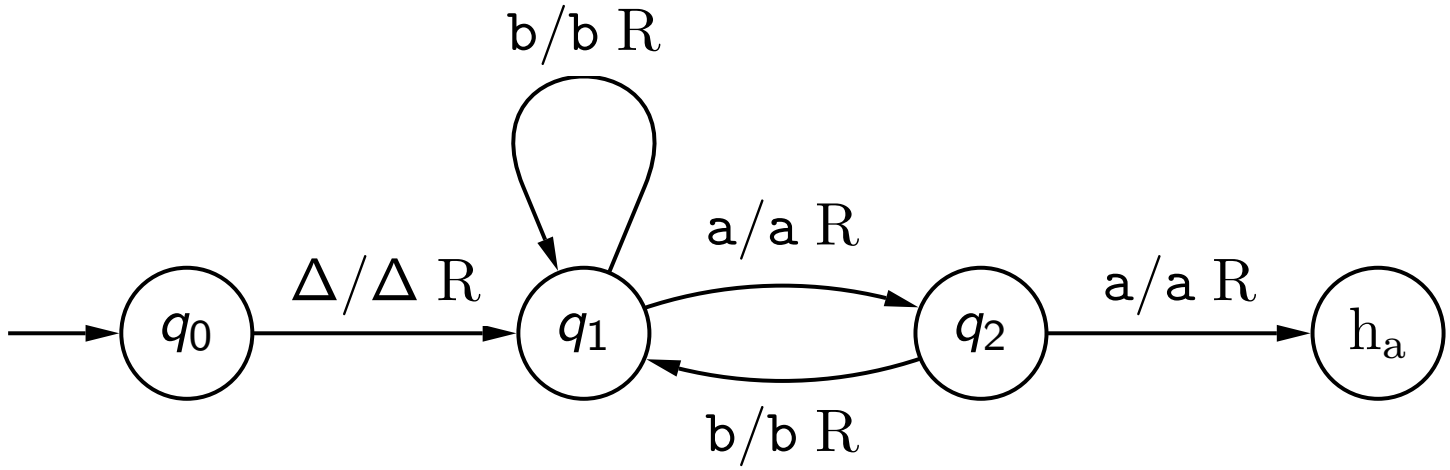


$$\begin{aligned} L(M) &= (\{b\}^*\{ab\}^*)^*\{aa\}\{a,b\}^* \quad \text{for } \Sigma = \{a,b\} \\ &= \{a,b\}^*\{aa\}\{a,b\}^* \end{aligned}$$

$$L(M) = \{a,b\}^*\{aa\}\{a,b,c\}^* \quad \text{for } \Sigma = \{a,b,c\}$$

Example (Turing machine)

M:



$$\begin{aligned} \mathrm{L}(M) &= (\{\mathtt{b}\}^*\{\mathtt{ab}\}^*)^*\{\mathtt{aa}\}\{\mathtt{a},\mathtt{b}\}^* \quad \text{for } \Sigma = \{\mathtt{a},\mathtt{b}\} \\ &= \{\mathtt{a},\mathtt{b}\}^*\{\mathtt{aa}\}\{\mathtt{a},\mathtt{b}\}^* \end{aligned}$$

$$\mathrm{L}(M) = \{\mathtt{a},\mathtt{b}\}^*\{\mathtt{aa}\}\{\mathtt{a},\mathtt{b},\mathtt{c}\}^* \quad \text{for } \Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$$

$$\mathrm{L}(M) = \{\mathtt{a},\mathtt{b}\}^*\{\mathtt{aa}\}\Sigma^*$$

Example (cont'd)

Transition sequence of M on input baabab:

$$
\begin{aligned}
q_0\Delta\mathtt{baabab} &\vdash \Delta q_1\mathtt{baabab} \\
&\vdash \Delta\mathtt{b}q_1\mathtt{aabab} \\
&\vdash \Delta\mathtt{ba}q_2\mathtt{abab} \\
&\vdash \Delta\mathtt{baa}\mathrm{h_a}\mathtt{bab}
\end{aligned}
$$

Rejecting transitions

To keep transition diagrams and transition tables small, we agree that transitions to the reject state need not be specified:

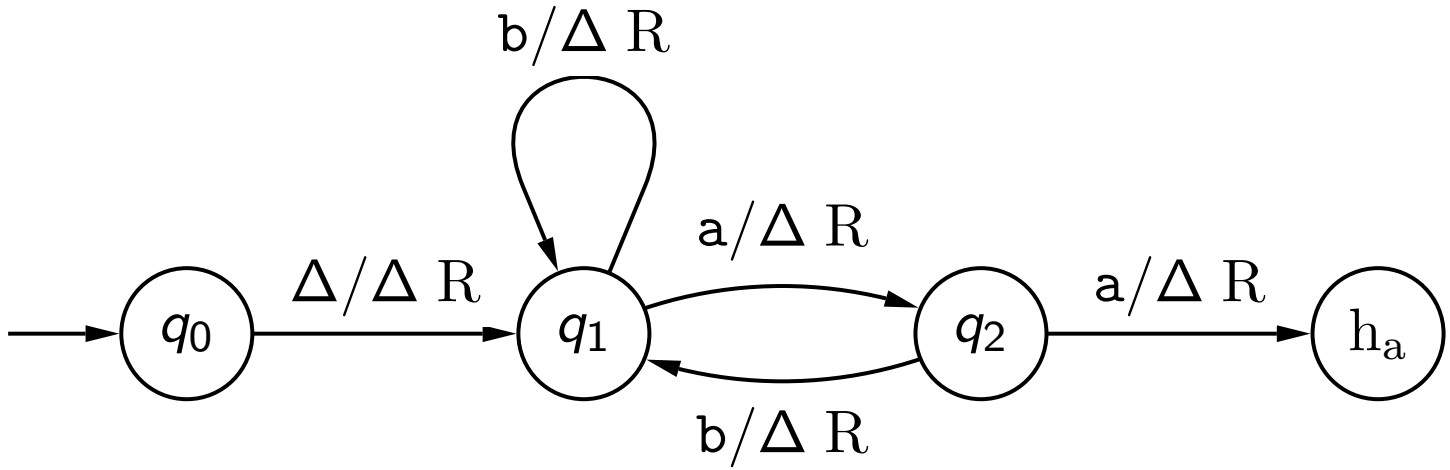
Convention

In specifications of Turing machines, if there is no transition for some pair $(q, a) \in (Q - \{\mathrm{h_a}, \mathrm{h_r}\}) \times \Gamma$, then

$$\delta(q, a) = (\mathrm{h_r}, a, \mathrm{S}).$$

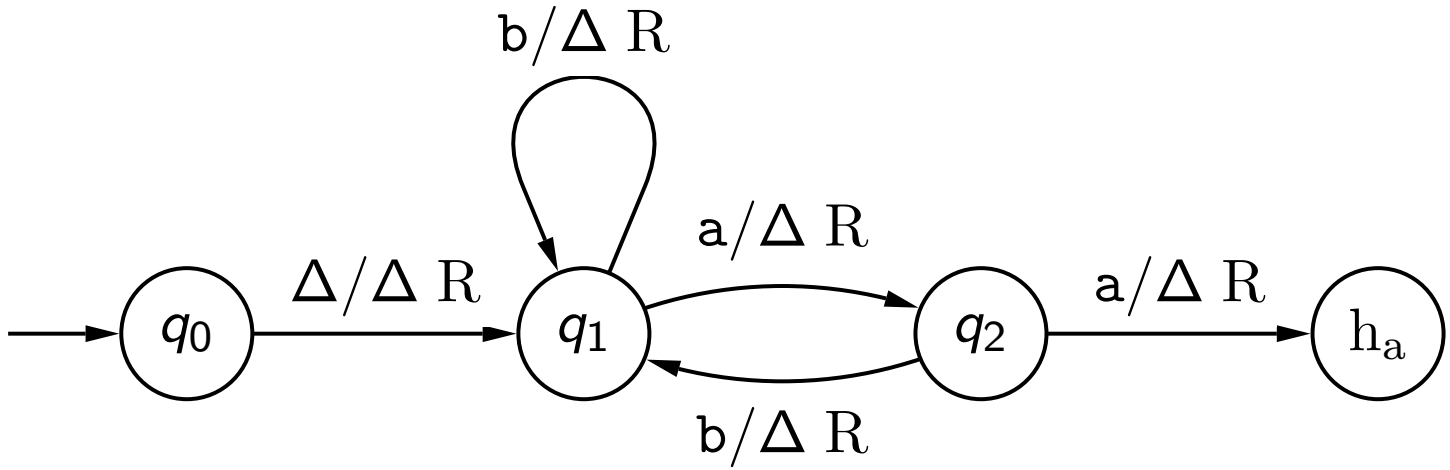
Example (cont'd)

M':



Example (cont'd)

M':



$$\mathrm{L}(M') = \mathrm{L}(M)$$

Alan Mathison Turing (1912 – 1954)