Computability

voorjaar 2024

`https://liacs.leidenuniv.nl/~vlietrvan1/computability/`

college 4, 1 maart 2024

7.7. Nondeterministic Turing Machines
7.6. The Church-Turing Thesis
7.8. Universal Turing Machines
8.1. Recursively Enumerable and Recursive

Huiswerkopgave

voor 0.4pt

individueel

inleveren: vrijdag 8 maart 2024, 23.59 uur

Laatste hoor/werkcollege

vrijdag 22 maart, 11.00-15.00 ipv 09.00-12.45

Example 7.30. The Language of Prefixes of Elements of L.

Let $L = L(T)$. Then

$$P(L) = \{x \in \Sigma^* \mid \; xy \in L \text{ for some } y \in \Sigma^*\}$$

$L = \{abba\}\ldots$

Example 7.30. The Language of Prefixes of Elements of L.

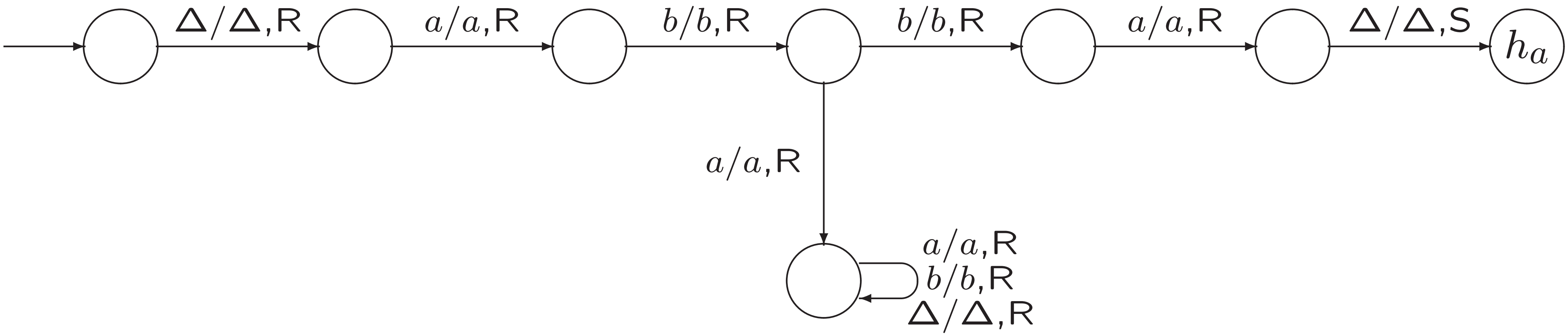
Let $L = L(T)$. Then

$$P(L) = \{x \in \Sigma^* \mid \; xy \in L \text{ for some } y \in \Sigma^*\}$$

Deterministic TM accepting $P(L)$ may execute following algorithm for input x:

$y = \Lambda$;
while (T does not accept xy)
    y is next string in Σ^* (in canonical order);
accept;

but...



Is $x = a \in P(L)$?

Example 7.30. The Language of Prefixes of Elements of L.

Let $L = L(T)$. Then

$$P(L) = \{x \in \Sigma^* \mid \; xy \in L \text{ for some } y \in \Sigma^*\}$$

$$\textit{NB} \to \textit{G} \to \textit{Delete} \to \textit{PB} \to T$$

Theorem 7.31.
For every nondeterministic TM $T = (Q, \Sigma, \Gamma, q_0, \delta)$, there is an ordinary (deterministic) TM $T_1 = (Q_1, \Sigma, \Gamma_1, q_1, \delta_1)$ with $L(T_1) = L(T)$.

Moreover, if there is no input on which T can loop forever, then T_1 also halts on every input.

The proof of this result does not have to be known for the exam.

N.B.

- NTM is not directly useful as algorithm to test membership of string x

- acceptance of string x:
  - there exists a run of NTM for x that leads to acceptance
  - not: repeat running NTM for x until it accepts

Nondeterminism

- TMs
- PDAs
- FAs

NP completeness / complexity

- nondeterminism
- size of input

Complexity

- size of input

bool prime (int n)
$\{$
    $p = 2$;
    while ($p < n$ and p is not divisor of n)
        $p++$;

    if ($p == n$)
        return true;
    else
        return false;
$\}$

A slide from lecture 3

Example 7.21. Erasing the Tape

From the current position to the right

A slide from exercise class 3

Exercise 7.13.

Suppose T is a TM that accepts every input. We might like to construct a TM R_T such that for every input string x, R_T halts in the accepting state with exactly the same tape contents as when T halts on input x, but with the tape head positioned at the rightmost nonblank symbol on the tape.

Show that there is no fixed TM T_0 such that $R_T = TT_0$ for every T. (In other words, there is no TM capable of executing the instruction "move the tape head to the rightmost nonblank tape symbol" in every possible situation.)

Suggestion: Assume there is such a TM T_0, and try to find two other TMs T_1 and T_2 such that if $R_{T_1} = T_1T_0$ then R_{T_2} cannot be T_2T_0.

Assume that the tape contains at least one nonblank symbol, when T halts.

7.6. The Church-Turing Thesis

Turing machine is general model of computation.

Any algorithmic procedure that can be carried out at all
(by human computer, team of humans, electronic computer)
can be carried out by a TM.
(Alonzo Church, 1930s)

Evidence for Church-Turing thesis:

1. Nature of the model.

2. Various enhancements of TM do not change computing power.

3. Other theoretical models of computation have been proposed. Various notational systems have been suggested as ways of describing computations. All of them equivalent to TM.

4. No one has suggested any type of computation that ought to be considered 'algorithmic procedure' and cannot be implemented on TM.

Once we adopt Church-Turing thesis,

- we have definition of algorithmic procedure

- we may omit details of TMs

7.8. Universal Turing Machines

Definition 7.32. Universal Turing Machines

A *universal* Turing machine is a Turing machine T_u that works as follows. It is assumed to receive an input string of the form $e(T)e(z)$, where
- T is an arbitrary TM,
- z is a string over the input alphabet of T,
- and e is an encoding function whose values are strings in $\{0,1\}^*$.

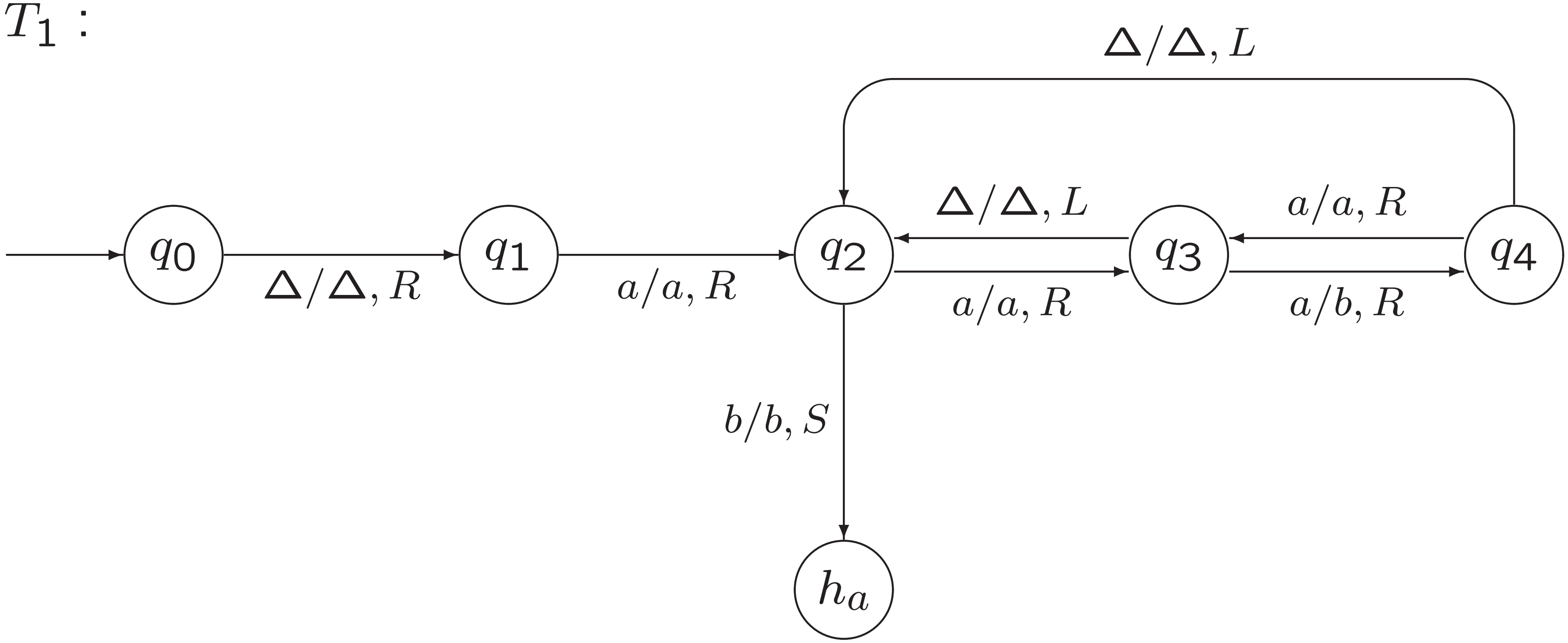
The computation performed by T_u on this input string satisfies these two properties:
1. T_u accepts the string $e(T)e(z)$ if and only if T accepts z.
2. If T accepts z and produces output y, then T_u produces output $e(y)$.

Some Crucial features of any encoding function e:

1. It should be possible to decide algorithmically, for any string $w \in \{0, 1\}^*$, whether w is a legitimate value of e.
2. A string w should represent at most one Turing machine, or at most one string z.
3. If $w = e(T)$ or $w = e(z)$, there should be an algorithm for *decoding* w.

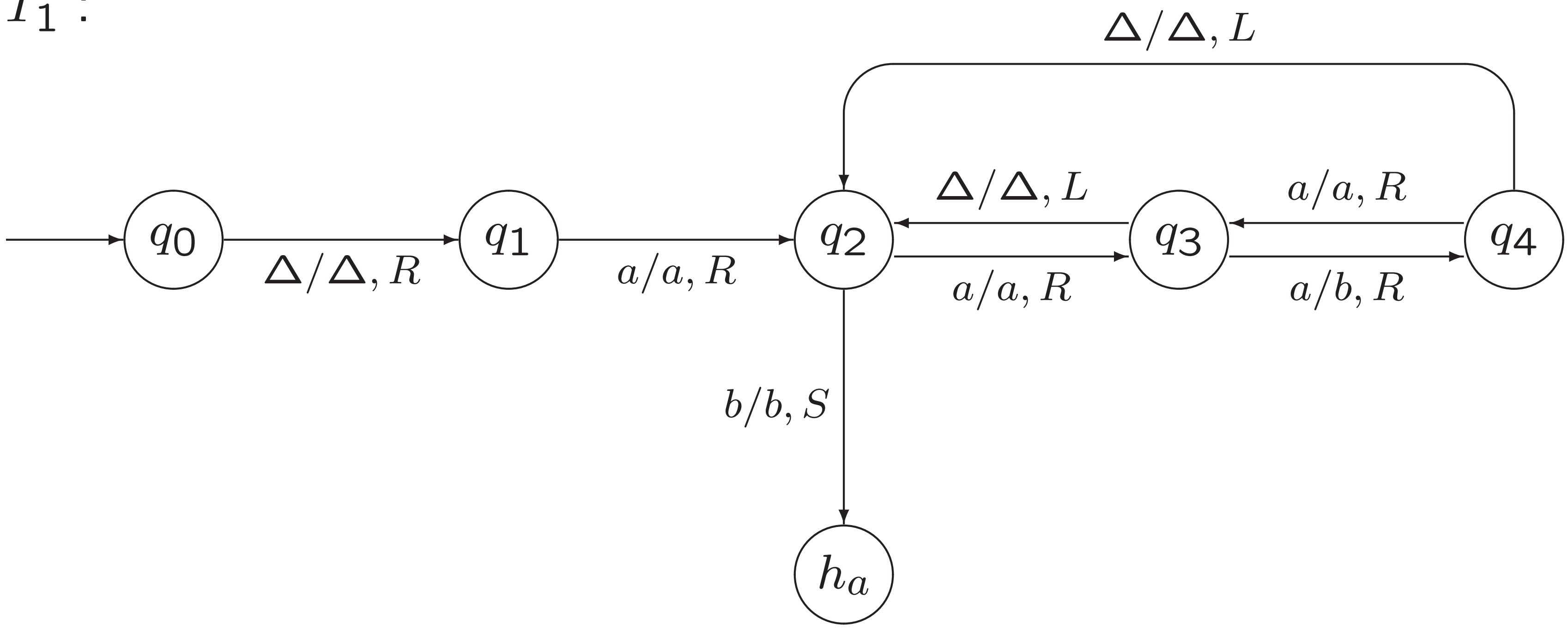
Computability e itself...

T_1 :

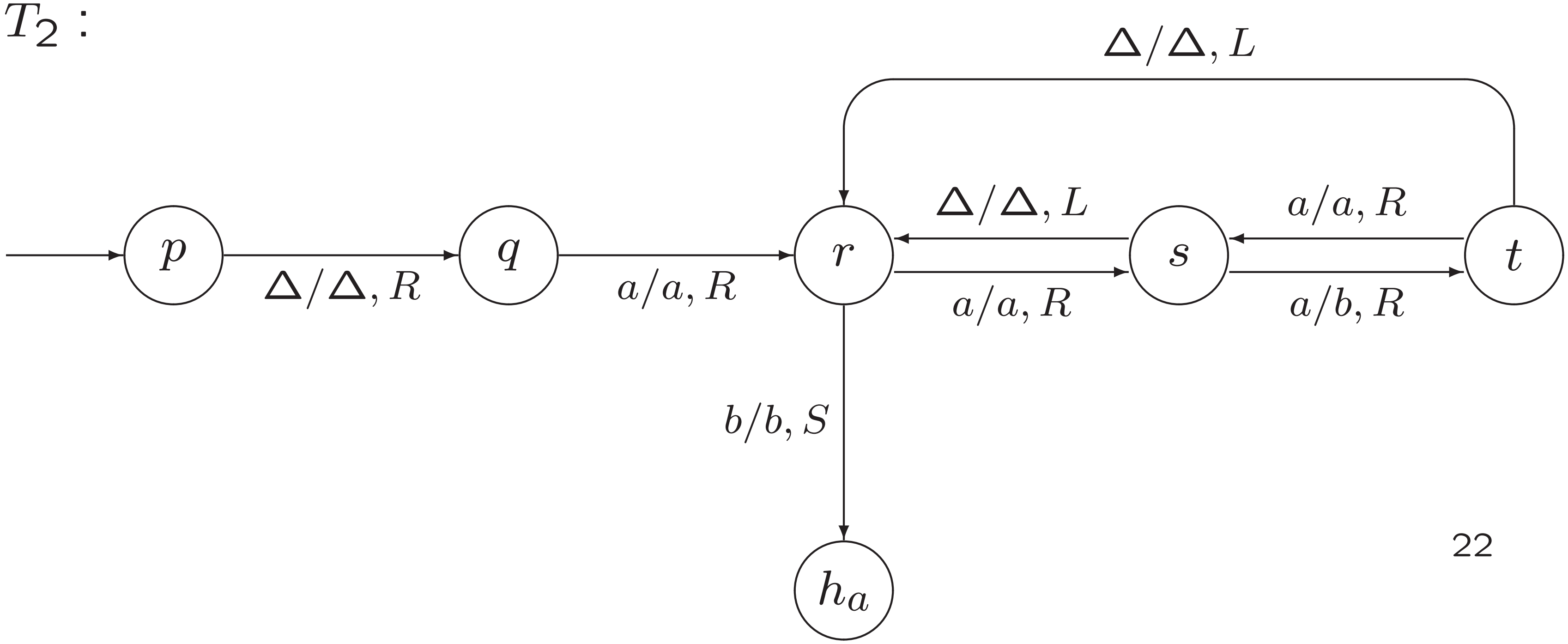

$\Delta/\Delta, L$
q_0
$\Delta/\Delta, R$
q_1
$a/a, R$
q_2
$\Delta/\Delta, L$
$a/a, R$
q_3
$a/a, R$
$a/b, R$
q_4
$b/b, S$
h_a

$L(T_1) = \ldots$

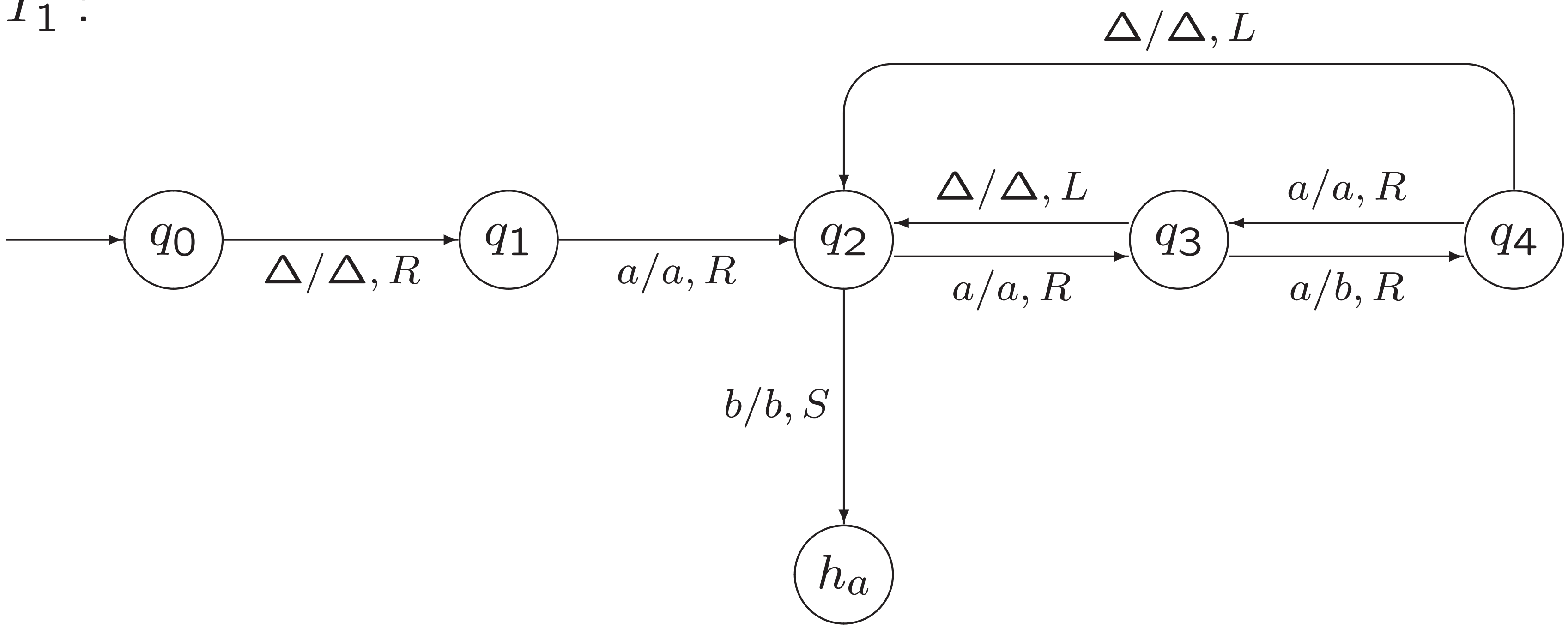
T_1 :


$\Delta/\Delta, L$
q_0
$\Delta/\Delta, R$
q_1
$a/a, R$
q_2
$\Delta/\Delta, L$
$a/a, R$
q_3
$a/a, R$
$a/b, R$
q_4
$b/b, S$
h_a

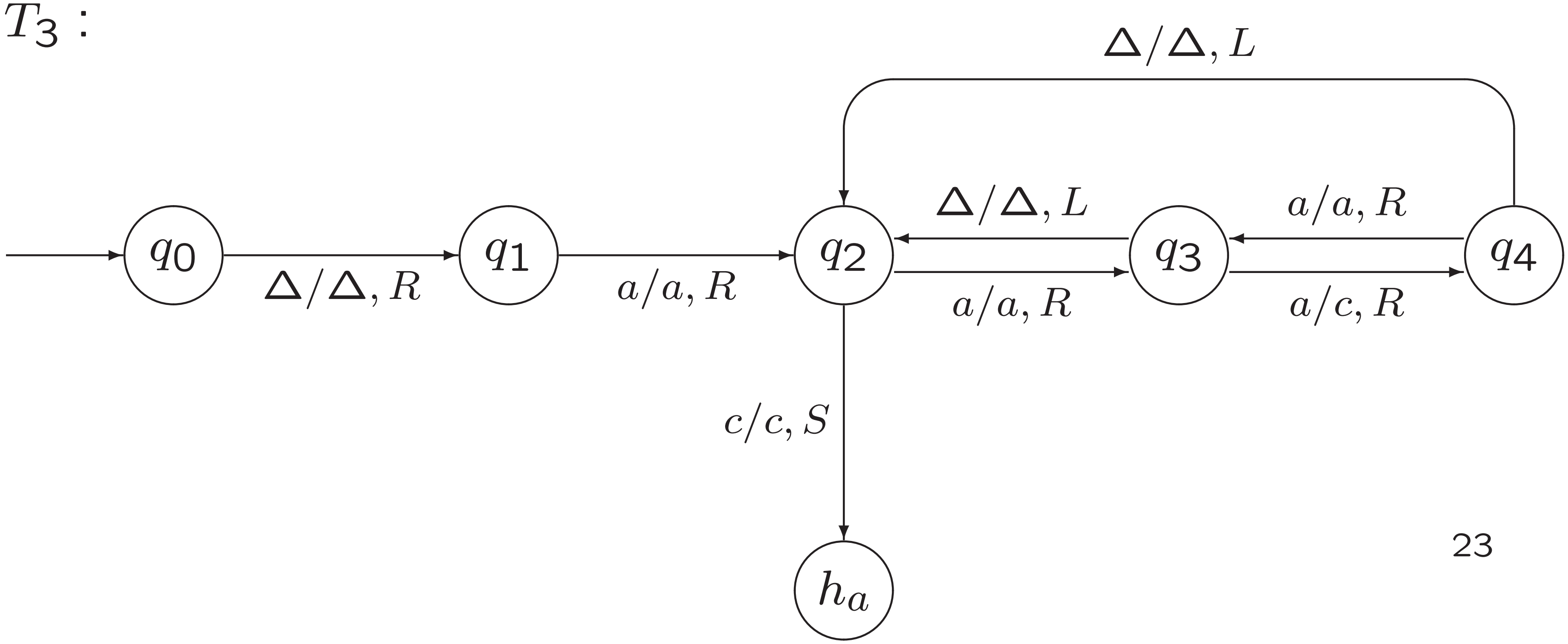
T_2 :


$\Delta/\Delta, L$
p
$\Delta/\Delta, R$
q
$a/a, R$
r
$\Delta/\Delta, L$
$a/a, R$
s
$a/a, R$
$a/b, R$
t
$b/b, S$
h_a

T_1 :


q_0
q_1
q_2
q_3
q_4
h_a
$\Delta/\Delta, R$
$a/a, R$
$a/a, R$
$\Delta/\Delta, L$
$a/b, R$
$a/a, R$
$\Delta/\Delta, L$
$b/b, S$

T_3 :


q_0
q_1
q_2
q_3
q_4
h_a
$\Delta/\Delta, R$
$a/a, R$
$a/a, R$
$\Delta/\Delta, L$
$a/c, R$
$a/a, R$
$\Delta/\Delta, L$
$c/c, S$

Assumptions:

1. Names of the states are irrelevant.

2. Tape alphabet Γ of every Turing machine T is subset of infinite set $\mathcal{S} = \{a_1, a_2, a_3, \ldots\}$, where $a_1 = \Delta$.

Definition 7.33. An Encoding Function

Assign numbers to each state:
$n(h_a) = 1$, $n(h_r) = 2$, $n(q_0) = 3$, $n(q) \geq 4$ for other $q \in Q$.

Assign numbers to each tape symbol:
$n(a_i) = i$.

Assign numbers to each tape head direction:
$n(R) = 1$, $n(L) = 2$, $n(S) = 3$.

Definition 7.33. An Encoding Function (continued)

For each move m of T of the form $\delta(p, \sigma) = (q, \tau, D)$

$$e(m) = 1^{n(p)}01^{n(\sigma)}01^{n(q)}01^{n(\tau)}01^{n(D)}0$$

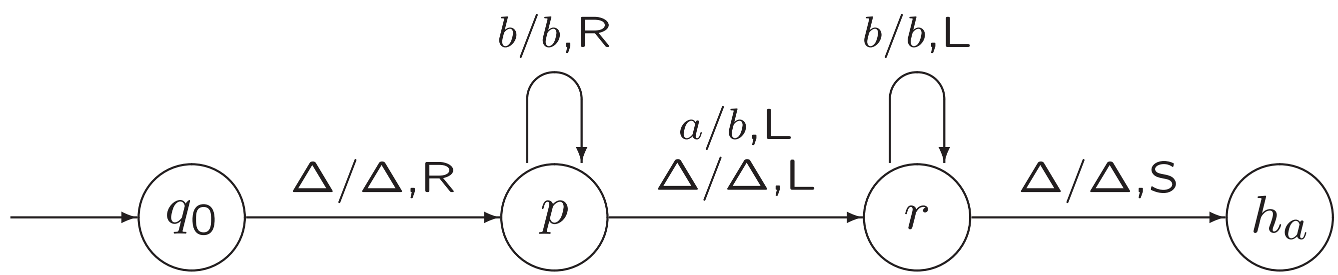
We list the moves of T in some order as $m_1, m_2, \ldots, m_k$, and we define

$$e(T) = e(m_1)0e(m_2)0\ldots0e(m_k)0$$

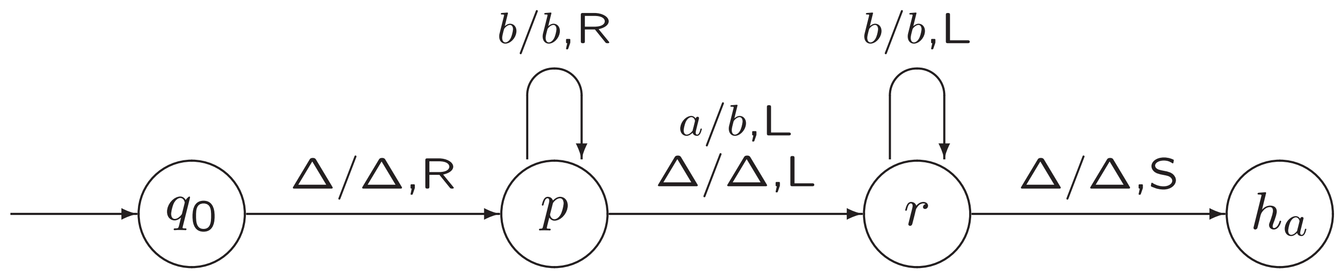
If $z = z_1 z_2 \ldots z_j$ is a string, where each $z_i \in \mathcal{S}$,

$$e(z) = 01^{n(z_1)}01^{n(z_2)}0\ldots01^{n(z_j)}0$$

Example 7.34. A Sample Encoding of a TM

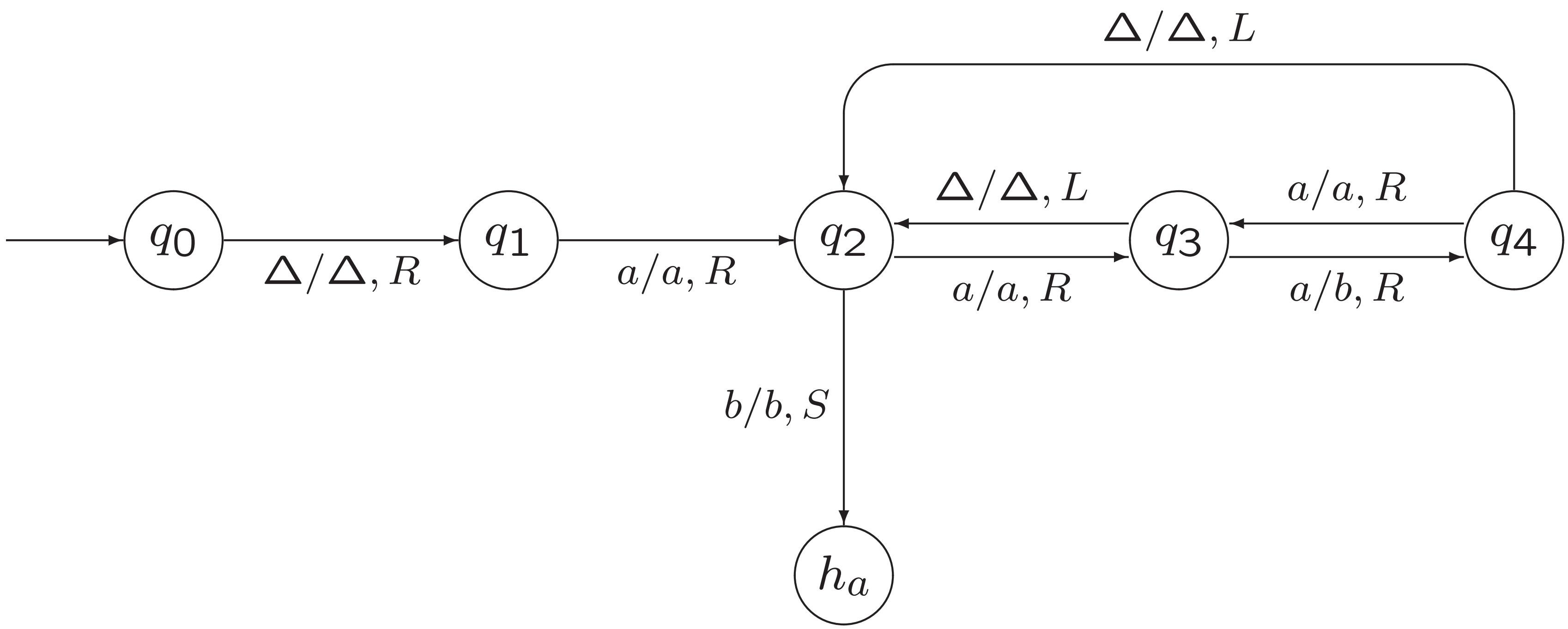


Example 7.34. A Sample Encoding of a TM



$\underline{1}11010111101010$ 0 111101110111101111010 0
1111011011111011110110 0 1111010111111010110 0
111110111011111101110110 0 11111010101011110 0

Does $e(T)$ completely specify $T = (Q, \Sigma, \Gamma, q_0, \delta)$?


Δ/Δ, L
Δ/Δ, L
a/a, R
q0
q1
q2
q3
q4
Δ/Δ, R
a/a, R
a/a, R
a/b, R
b/b, S
ha

Definition 7.32. Universal Turing Machines

A *universal* Turing machine is a Turing machine T_u that works as follows. It is assumed to receive an input string of the form $e(T)e(z)$, where
• T is an arbitrary TM,
• z is a string over the input alphabet of T,
• and e is an encoding function whose values are strings in $\{0, 1\}^*$.

The computation performed by T_u on this input string satisfies these two properties:
1. T_u accepts the string $e(T)e(z)$ if and only if T accepts z.
2. If T accepts z and produces output y, then T_u produces output $e(y)$.

Some Crucial features of any encoding function e:

1. It should be possible to decide algorithmically, for any string $w \in \{0,1\}^*$, whether w is a legitimate value of e.
2. A string w should represent at most one Turing machine with a given input alphabet Σ, or at most one string z.
3. If $w = e(T)$ or $w = e(z)$, there should be an algorithm for *decoding* w.

Computability e itself...

reg. languages	FA	reg. grammar	reg. expression
determ. cf. languages	DPDA		
cf. languages	PDA	cf. grammar	
cs. languages	LBA	cs. grammar	
re. languages	TM	unrestr. grammar	

8. Recursively Enumerable Languages

8.1. Recursively Enumerable and Recursive

7.6. The Church-Turing Thesis

Turing machine is general model of computation.

Any algorithmic procedure that can be carried out at all
(by human computer, team of humans, electronic computer)
can be carried out by a TM.
(Alonzo Church, 1930s)

A slide from lecture 2

Example 7.14. The Characteristic Function of a Set

$$\chi_L(x) = \begin{cases} 1 & \text{if } x \in L \\ 0 & \text{if } x \notin L \end{cases}$$

From computing χ_L to accepting L

From accepting L to computing χ_L

Definition 8.1. Accepting a Language and Deciding a Language

A Turing machine T with input alphabet Σ accepts a language $L \subseteq \Sigma^*$,
if $L(T) = L$.

T *decides* L,
if T computes the characteristic function $\chi_L : \Sigma^* \to \{0, 1\}$

A language L is *recursively enumerable*,
if there is a TM that accepts L,

and L is *recursive*,
if there is a TM that decides L.

Theorem 8.2.
Every recursive language is recursively enumerable.

Proof. . .

Theorem 8.3.
If $L \subseteq \Sigma^*$ is accepted by a TM T that halts on every input string, then L is recursive.

Proof. . .

How to modify TM such that it never falls off the tape?

Corollary.

If L is accepted by a nondeterministic TM T, and if there is no input string on which T can possibly loop forever,
then L is recursive.

Proof. . .

Theorem 7.31.
For every nondeterministic TM $T = (Q, \Sigma, \Gamma, q_0, \delta)$, there is an ordinary (deterministic) TM $T_1 = (Q_1, \Sigma, \Gamma_1, q_1, \delta_1)$ with $L(T_1) = L(T)$.

Moreover, if there is no input on which T can loop forever, then T_1 also halts on every input.

The proof of this result does not have to be known for the exam.

Theorem 8.4. If L_1 and L_2 are both recursively enumerable languages over Σ, then $L_1 \cup L_2$ and $L_1 \cap L_2$ are also recursively enumerable.

Proof. . .

For $L_1 \cup L_2$:

$T_1 \backslash T_2$	h_a	h_r	∞
h_a	h_a	h_a	h_a
h_r	h_a	h_r	∞
∞	h_a	∞	∞

For $L_1 \cap L_2$:

$T_1 \backslash T_2$	h_a	h_r	∞
h_a	h_a	h_r	∞
h_r	h_r	h_r	h_r
∞	∞	h_r	∞

Exercise 8.2. Consider modifying the proof of Theorem 8.4 by executing the two TMs sequentially instead of simultaneously. Given TMs T_1 and T_2 accepting L_1 and L_2, respectively, and an input string x, we start by making a second copy of x.
We execute T_1 on the second copy; if and when this computation stops, the tape is erased except for the original input, and T_2 is executed on it.

a. Is this approach feasible for accepting $L_1 \cup L_2$, thereby showing that the union of recursively enumerable languages is recursively enumerable? Why or why not?

b. Is this approach feasible for accepting $L_1 \cap L_2$, thereby showing that the intersection of recursively enumerable languages is recursively enumerable? Why or why not?

For intersection: not just $T_1 \to T_2$