

# Fundamentele Informatica 3

voorjaar 2018

<http://www.liacs.leidenuniv.nl/~vlietrvan1/fi3/>

**Rudy van Vliet**

kamer 140 Snellius, tel. 071-527 2876  
rvvliet(at)liacs(dot)nl

college 4, 26 februari 2018

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

*A slide from lecture 3*

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

Let  $L = L(T)$ . Then

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

*NB  $\rightarrow$  G  $\rightarrow$  Delete  $\rightarrow$  PB  $\rightarrow$  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.

## 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;
}
```

## 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...

## Assumptions:

1. Names of the states are irrelevant.
2. Tape alphabet  $\Gamma$  of every Turing machine  $T$  is subset of infinite set  $\mathcal{S} = \{a_1, a_2, a_3, \dots\}$ , where  $a_1 = \Delta$ .

**Definition 7.33.** An Encoding Function

Assign numbers to each state:

$$n(h_a) = 1, n(h_r) = 2, n(q_0) = 3, n(q) \geq 4 \text{ for other } q \in Q.$$

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  $\delta(p, \sigma) = (q, \tau, D)$

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

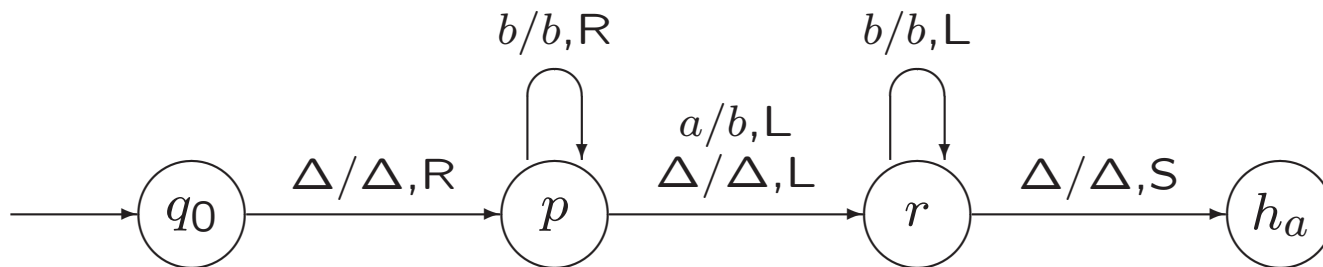
We list the moves of  $T$  in **some** order as  $m_1, m_2, \dots, m_k$ , and we define

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

If  $z = z_1z_2 \dots z_j$  is a string, where each  $z_i \in \mathcal{S}$ ,

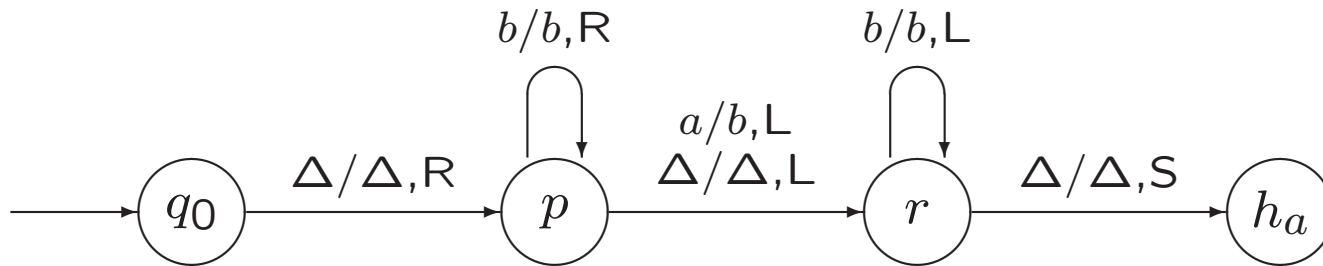
$$e(z) = 01^{n(z_1)}01^{n(z_2)}0 \dots 01^{n(z_j)}0$$

**Example 7.34.** A Sample Encoding of a TM





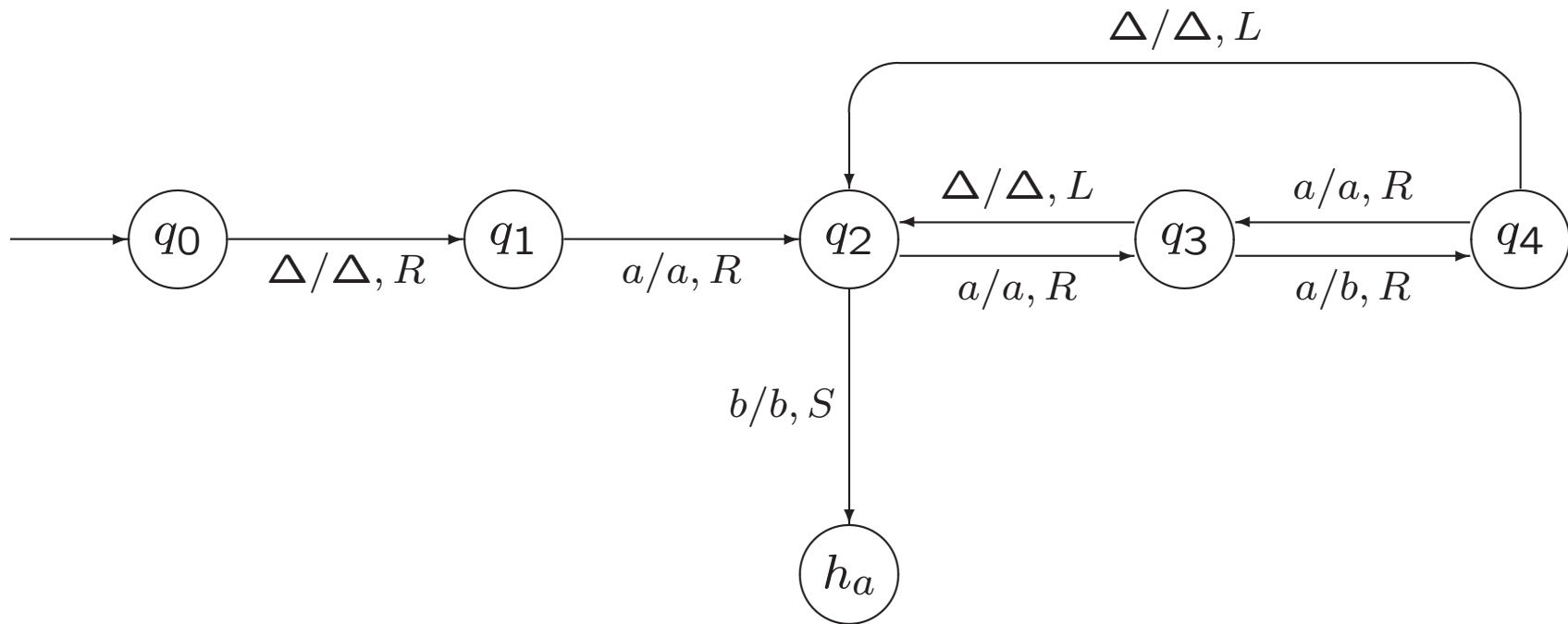
**Example 7.34.** A Sample Encoding of a TM



```

111010111101010 0 11110111011110111010 0
111101101111101110110 0 111101011111010110 0
11111011101111101110110 0 1111101010101110 0
    
```

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



## 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  $\Sigma$** , 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...

# Huiswerkopgave 1

|                       |      |                  |                 |
|-----------------------|------|------------------|-----------------|
| 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  $\chi_L$  to accepting  $L$

From accepting  $L$  to computing  $\chi_L$

**Definition 8.1.** Accepting a Language and Deciding a Language

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

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

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...**

## 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  $\Sigma$ , then  $L_1 \cup L_2$  and  $L_1 \cap L_2$  are also recursively enumerable.

**Proof...**



For  $L_1 \cup L_2$ :

|          |       |          |          |
|----------|-------|----------|----------|
| $T_2$    | $h_a$ | $h_r$    | $\infty$ |
| $T_1$    |       |          |          |
| $h_a$    | $h_a$ | $h_a$    | $h_a$    |
| $h_r$    | $h_a$ | $h_r$    | $\infty$ |
| $\infty$ | $h_a$ | $\infty$ | $\infty$ |

For  $L_1 \cap L_2$ :

|          |          |       |          |
|----------|----------|-------|----------|
| $T_2$    | $h_a$    | $h_r$ | $\infty$ |
| $T_1$    |          |       |          |
| $h_a$    | $h_a$    | $h_r$ | $\infty$ |
| $h_r$    | $h_r$    | $h_r$ | $h_r$    |
| $\infty$ | $\infty$ | $h_r$ | $\infty$ |

**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  $T_1 \rightarrow T_2$

### Exercise 8.1.

Show that if  $L_1$  and  $L_2$  are recursive languages, then  $L_1 \cup L_2$  and  $L_1 \cap L_2$  are also.

**Theorem 8.5.** If  $L_1$  and  $L_2$  are both recursive languages over  $\Sigma$ , then  $L_1 \cup L_2$  and  $L_1 \cap L_2$  are also recursive.

**Proof.** Exercise 8.1.

**Theorem 8.6.** If  $L$  is a recursive language over  $\Sigma$ , then its complement  $L'$  is also recursive.

**Proof...**

**Theorem 8.7.** If  $L$  is a recursively enumerable language, and its complement  $L'$  is also recursively enumerable, then  $L$  is recursive (and therefore, by Theorem 8.6,  $L'$  is recursive).

**Proof...**

**Corollary.**

Let  $L$  be a recursively enumerable language.

Then

$L'$  is recursively enumerable,

if and only

if  $L$  is recursive.



**Corollary.**

There exist languages that are not recursively enumerable,  
if and only if  
there exist languages that are not recursive.