Fundamentele Informatica 3

voorjaar 2015

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

Rudy van Vliet
kamer 124 Snellius, tel. 071-527 5777
rvvliet(at)liacs(dot)nl

college 5, 2 maart 2015

7. Turing Machines
7.8. Universal Turing Machines
8. Recursively Enumerable Languages
8.1. Recursively Enumerable and Recursive
8.2. Enumerating a Language

Huiswerkopgave 1

Voor 0.4pt

Inleveren: dinsdag 3 maart 2015, 13:45 uur

A slide from lecture 4

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 $e(y)$.

A slide from lecture 4

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:
$n(R) = 1$, $n(L) = 2$, $n(S) = 3$.

A slide from lecture 4

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 $z_i \in \mathcal{S}$,

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

Theorem 7.36. Let $E = \{e(T) \mid T \text{ is a Turing machine}\}$. Then for every $x \in \{0,1\}^*$, $x \in E$ if and only if all these conditions are satisfied:

1. x matches the regular expression

$$(11^*0)^5 0((11^*0)^5 0)^*$$

so that it can be viewed as a sequence of one or more 5-tuples.

. . .

Theorem 7.36. Let $E = \{e(T) \mid T \text{ is a Turing machine}\}$. Then for every $x \in \{0,1\}^*$, $x \in E$ if and only if all these conditions are satisfied:

1. x matches the regular expression $(11^*0)^5 0((11^*0)^5 0)^*$ so that it can be viewed as a sequence of one or more 5-tuples.

2. No two substrings of x representing 5-tuples can have the same first two parts (no move can appear twice, and there can't be two different moves for a given combination of state and tape symbol).

3. None of the 5-tuples can have first part 1 or 11 (there can be no moves from a halting state).

4. The last part of each 5-tuple must be 1, 11, or 111 (it must represent a direction).

Is this correct?

Simulation of TM T on input z by universal TM T_u

- Three tapes
  1. $e(T)$
  2. e(tape contents)
  3. $e(q)$

- Initialize tapes

- Simulate

- Termination of T
  - if no termination, . . .
  - if reject (three types), . . .
  - if accept, . . .

Example 7.34. A Sample Encoding of a TM

Exercise.

Suppose the three tapes of the universal Turing machine look like this:

> Δ $\underline{1}$11010111101010 0 11110111011110111010 0
> 1111011011111101110110 0 1111010111111010110 0
> 111110111011111101110110 0 1111101010101110 0 Δ
> Δ1011101110$\underline{1}$101110110Δ
> $\Delta\underline{1}$111Δ

What will the three tapes look like after the next simulated move?
What will the three tapes look like after the next next simulated move?

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

8. Recursively Enumerable Languages

8.1. Recursively Enumerable and Recursive

A slide from lecture 4

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. . .

Corollary.

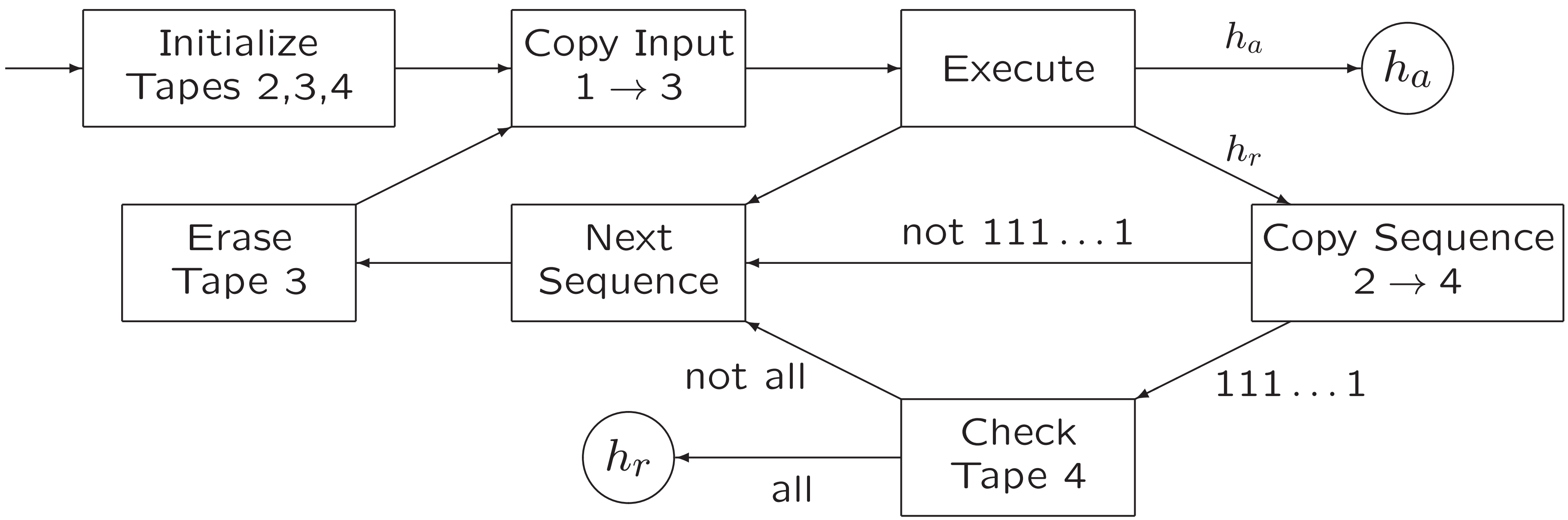
If L is accepted by a nondeterministic TM T, and if there is no input string on which T can possibly loop forever, then L is recursive.

Proof. . .

A slide from lecture 4

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 $L(T_1) = L(T)$.

Proof. . .



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. . .

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?

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 Σ, 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 Σ, 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.

8.2. Enumerating a Language

Definition 8.8. A TM Enumerating a Language

Let T be a k-tape Turing machine for some $k \geq 1$, and let $L \subseteq \Sigma^*$. We say T enumerates L if it operates such that the following conditions are satisfied.

1. The tape head on the first tape never moves to the left, and no nonblank symbol printed on tape 1 is subsequently modified or erased.

2. For every $x \in L$, there is some point during the operation of T when tape 1 (which is initially empty) has contents

$$x_1\#x_2\#\ldots\#x_n\#x\#$$

for some $n \geq 0$, where the strings $x_1, x_2, \ldots, x_n$ are also elements of L and $x_1, x_2, \ldots, x_n, x$ are all distinct. If L is finite, then nothing is printed after the # following the last element of L.

Theorem 8.9. For every language $L \subseteq \Sigma^*$,
• L is recursively enumerable
if and only if there is a TM enumerating L,
• and L is recursive if and only if there is a TM that enumerates the strings in L in canonical order (see Section 1.4).

In other words:
1. If there is a TM that accepts L, then there is a TM that enumerates L.
2. If there is a TM that enumerates L, then there is a TM that accepts L.
3. If there is a TM that decides L, then there is a TM that enumerates L in canonical order.
4. If there is a TM that enumerates L in canonical order, then there is a TM that decides L.

Proof. . .