Praktisch

- Huiswerkopgave 4. . .
- Werkcollege-dag-tijd. . .
- 12.45: grammatica's

7.5. Multitape Turing Machines

7.7. Nondeterministic Turing Machines

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

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

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

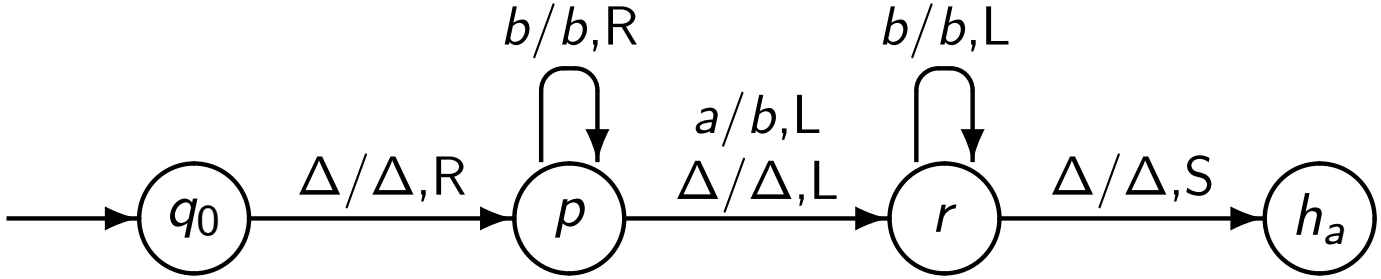
We list the moves of T in some order as $m_1, m_2, \ldots, m_k$, and we define

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

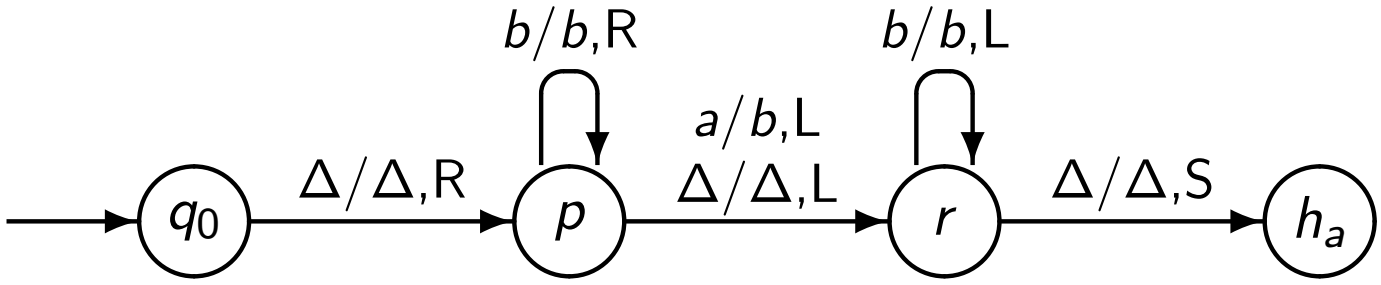
If $z = z_1 z_2 \ldots z_j$ is a string, where each $z_i \in \mathcal{S}$,

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

Example 7.34. A Sample Encoding of a TM



Example 7.34. A Sample Encoding of a TM



$\underline{1}11010111101010\ 0\ 1111011101111011101 0$
$1111011011111011101100\ 0\ 111101011111010110\ 0$
$1111011101111101101100\ 0\ 111110101010110\ 0$

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	

Section 8

Recursively Enumerable Languages

7 Recursively Enumerable Languages
- Recursively Enumerable and Recursive
- Not Every Language is Recursively Enumerable

A slide from lecture 12

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?

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 \ 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 \ 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 $T_1 \rightarrow T_2$

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

Not Every Language is Recursively Enumerable

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	

Section 9

Undecidable Problems

8 Undecidable Problems
- A Language
  That Can't Be Accepted,
  and a Problem That Can't Be Decided

A slide from lecture 13

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.

A slide from lecture 13

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 13

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

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

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

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

	$e(T_0)$	$e(T_1)$	$e(T_2)$	$e(T_3)$	$e(T_4)$	$e(T_5)$	$e(T_6)$	$e(T_7)$	$e(T_8)$	$e(T_9)$	...
$L(T_0)$	1	0	1	0	0	1	0	0	0	1	...
$L(T_1)$	0	1	1	1	0	0	0	0	1	0	...
$L(T_2)$	1	0	0	1	0	0	1	0	0	0	...
$L(T_3)$	0	0	0	0	0	0	0	0	0	0	...
$L(T_4)$	0	0	0	0	1	0	0	0	0	0	...
$L(T_5)$	0	0	1	1	0	1	0	1	0	0	...
$L(T_6)$	0	0	0	0	0	0	0	0	1	0	...
$L(T_7)$	1	1	1	1	1	1	1	1	1	1	...
$L(T_8)$	0	1	0	1	0	1	0	1	0	1	...
$L(T_9)$	0	0	0	0	0	0	0	0	0	0	...
...						...					

	$e(T_0)$	$e(T_1)$	$e(T_2)$	$e(T_3)$	$e(T_4)$	$e(T_5)$	$e(T_6)$	$e(T_7)$	$e(T_8)$	$e(T_9)$	...
$L(T_0)$	1	0	1	0	0	1	0	0	0	1	...
$L(T_1)$	0	1	1	1	0	0	0	0	1	0	...
$L(T_2)$	1	0	0	1	0	0	1	0	0	0	...
$L(T_3)$	0	0	0	0	0	0	0	0	0	0	...
$L(T_4)$	0	0	0	0	1	0	0	0	0	0	...
$L(T_5)$	0	0	1	1	0	1	0	1	0	0	...
$L(T_6)$	0	0	0	0	0	0	0	0	1	0	...
$L(T_7)$	1	1	1	1	1	1	1	1	1	1	...
$L(T_8)$	0	1	0	1	0	1	0	1	0	1	...
$L(T_9)$	0	0	0	0	0	0	0	0	0	0	...
...						...					
NSA	0	0	1	1	0	0	1	0	1	1	...

Hence, NSA is not recursively enumerable.

Set-up of constructing language *NSA* that is not RE:

1. Start with list of RE languages over $\{0, 1\}$
   (which are subsets of $\{0, 1\}^*$): $L(T_0), L(T_1), L(T_2), \ldots$
   each one associated with specific element of $\{0, 1\}^*$
   (namely $e(T_i)$)
2. Define another language *NSA* by:
   $e(T_i) \in NSA \iff e(T_i) \notin L(T_i)$
3. Conclusion: for all i, $NSA \neq L(T_i)$
   Hence, *NSA* is not RE

Set-up of constructing language L that is not RE:

1. Start with list of RE languages over $\{0,1\}$
   (which are subsets of $\{0,1\}^*$): $L(T_0), L(T_1), L(T_2), \ldots$
   each one associated with specific element of $\{0,1\}^*$
   (namely x_i)
2. Define another language L by:
   $x_i \in L \iff x_i \notin L(T_i)$
3. Conclusion: for all i, $L \neq L(T_i)$
   Hence, L is not RE

Every infinite list $x_0, x_1, x_2, \ldots$ of different elements of $\{0,1\}^*$ yields language L that is not RE

	Λ	0	1	00	01	10	11	000	001	010	...
$L(T_0)$	1	0	1	0	0	1	0	0	0	1	...
$L(T_1)$	0	1	1	1	0	0	0	0	1	0	...
$L(T_2)$	1	0	0	1	0	0	1	0	0	0	...
$L(T_3)$	0	0	0	0	0	0	0	0	0	0	...
$L(T_4)$	0	0	0	0	1	0	0	0	0	0	...
$L(T_5)$	0	0	1	1	0	1	0	1	0	0	...
$L(T_6)$	0	0	0	0	0	0	0	0	1	0	...
$L(T_7)$	1	1	1	1	1	1	1	1	1	1	...
$L(T_8)$	0	1	0	1	0	1	0	1	0	1	...
$L(T_9)$	0	0	0	0	0	0	0	0	0	0	...
...							...				
newL	0	0	1	1	0	0	1	0	1	1	...

Hence, newL is not recursively enumerable.

Definition 9.1. The Languages *NSA* and *SA*
Let

$$\begin{aligned} NSA &= \{e(T) \mid \ T \text{ is a TM, and } e(T) \notin L(T)\} \\ SA &= \{e(T) \mid \ T \text{ is a TM, and } e(T) \in L(T)\} \end{aligned}$$

(*NSA* and *SA* are for "non-self-accepting" and "self-accepting.")

A slide from lecture 13

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.

Theorem 9.2. The language *NSA* is not recursively enumerable. The language *SA* is recursively enumerable but not recursive.
Proof. . .

Given a TM T, does T accept the string $e(T)$?

Decision problem: problem for which the answer is 'yes' or 'no':

Given . . . , is it true that . . . ?

Given an undirected graph $G = (V, E)$,
does G contain a Hamiltonian path?

Given a list of integers $x_1, x_2, \ldots, x_n$,
is the list sorted?

Self-Accepting*: Given a TM T, does T accept the string $e(T)$?*

instances. . .