FORMAL LANGUAGES, AUTOMATA AND THEORY OF COMPUTATION

EXERCISES IN RECURSIVELY ENUMERABLE LANGUAGES

2010

RECURSIVELY ENUMERABLE LANGUAGES & TURING MACHINES

TURING MACHINES

Problem 1. **CH9 Problem 24**

Prove that a language *L* is recursive if and only if *L* and $\overline{L}$ are recursively enumerable.

Problem 2. **CH9 Problem 25 (See even Sipser 3.14-3.15)**

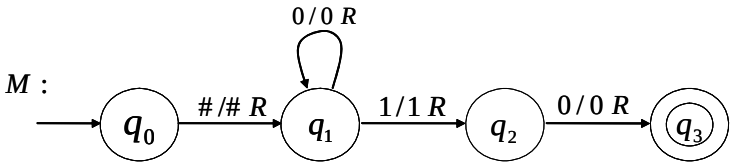
Prove that the recursive languages are closed under union, intersection, and complement.

Problem 3. **CH10 Problem 4a-d**

Prove that the recursively enumerable languages are closed under union, intersection, and concatenation, and Kleene star operations.

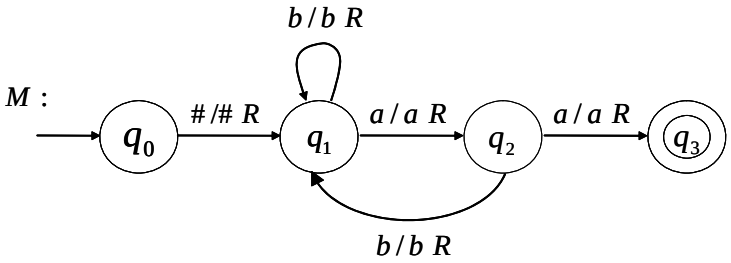
Problem 4. **CH11 Problem 6**

Given a Turing Machine M, what's *L (M)*?



Problem 5. **CH9 (9.2.1)**

Given a Turing Machine M, what's *L (M)*?



Problem 6. **CH9 Problem 3c (Sudkamp)**

Construct a Turing Machine with input alphabet {*a, b*} to perform:

c) Insert a blank between each of the input symbols.

Problem 7. **CH12 Problem 2e**

Construct a TM that computes the number-theoretic function

$$eq(n,m) = \begin{cases} 1 & if \quad n = m \\ 0 & otherwise \end{cases}$$

Problem 8. **"Shifting over"**

A common operation in Turing-machine programs involves "shifting over." We need to move the contents of each of the cells to the right of the current head position one cell right, and then find our way back to the current head position.

a. Give the high level description of how to perform this operation.
Hint: Leave a special symbol to mark the position to which the head must return.

b. Design a Turing machine that shifts the entire input string one cell to the right. In this part, you are to give the formal description of the machine (you can draw the state diagram, or provide the delta function for the entire domain). Precisely, you will design a Turing machine M such that, given an input string w ∈ {0,1}*, M's accepting configuration will be $q_{accept}\#w$.

Problem 9. Design Turing Machine that…

a) computes the function Ratio(x, 2) for arbitrary natural number x in binary representation. Input alphabet is {1} and tape alphabet {1, X, #}.

b) in unary representation decides if one natural number is bigger then the other. ("Decide" means that given $\#\mathbf{1}^{\mathbf{m}}\#\mathbf{1}^{\mathbf{n}}$, M stops and returns **YES** if **m** > **n**, and stops with output **NO** if m≤n.) Run the TM to illustrate its function.

c) accepts the language of odd integers written in binary.

d) accepts the language a#b#c, where a, b, c are in {0,1}*, and a + b = c, where a, b and c are interpreted as positive binary integers.

e) enumerates the language of odd integers written in binary.

f) Think about how tedious it would be to design a TM that enumerates all primes in binary!

Problems, LINZ

Problem 10. **Problem 8, page 257**

Suppose we make the requirement that a Turing machine can halt only in a final state, that is, we ask that δ (q,a) be defined for pairs (q,a) with a $\in \Gamma$ and q $\notin$ F. Does this restrict the power of the Turing Machine? Prove your answer.

Problem 11. **Problem 6, page 270**

Let $\Sigma = \{a, b, c\}$. We can show that $S = \Sigma^+$ is countable if we can find an enumeration procedure that produces its elements in some order, say in the order in which they would appear in the dictionary. However, the order used in dictionaries is not suitable without modification. In a dictionary, all words beginning with *a* are listed before the string *b*. But when there is infinite number of words, we will never reach *b*.

Instead, we can use the modified order, in which we take the length of the string as a first criterion, followed by an alphabetic ordering of all equal-length strings. This is an enumeration procedure that produces the sequence which is called **the proper order**

a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, …

Find a function *f(w)* that gives for each *w* its index in the proper ordering.

Problem 12. **Problem 12, page 282**

Let L_1 be recursive and L_2 recursively enumerable. Show that $L_2 - L_1$ is necessarily recursively enumerable

Problem 13. **Problem 13, page 282**

Suppose that *L* is such that there exists a Turing machine that enumerates the elements of *L* in proper order. Show that this means that L is recursive.

DECIDABILITY

Problem 1. (**CH11 Problem 12)** Prove that there is no algorithm that determines whether an arbitrary Tuning Machine halts when run with the input string 101.

Problem 2. Prove that the problem of determining whether for a Turing machine M there is some input string for which M halts is undecidable.

Problem 3. Prove that the problem of determining whether for a Turing machine M the language L(M) is regular is undecidable.

Problem 4. Prove each of the following languages decidable or undecidable.

(a) $\{(M_1>,(M_2>) \mid L(M_1) \cap \overline{L}(M_2) = \varnothing\}$
(b) {<M > | L(M) has exactly n elements}
(c) {<M > | L(M) is not recognizable}
(d) {<M > | M has an even number of states}
(e) {<M > | there exists a string x which M accepts in fewer than |x| steps}

Problem 5. Consider the problem of testing whether a Turing machine M on input w ever attempts to move its head left when its head is on the left-most tape cell. Formulate this problem as a language and show that it is undecidable.

Problem 6. Prove that the problem of determining if the languages generated by two CFGs are equal is undecidable.

Problem 7. Decidable or not? Given a (code for a) grammar does the language produced by the grammar contains infinitely many strings? Motivate your answer!

a) Is there any TM that can decide the problem for a restriction free grammar?
b) Is there any TM that can decide the problem for a context free grammar?
c) Is there any TM that can decide the problem for a regular grammar?

A Refutation of the Halting Problem?

Consider the language of all TMs that given no input eventually write a non-blank symbol on their tapes. Explain why this set is decidable. Why does this not conflict with the halting problem?

PRIMITIVE RECURSION

Problem 8. **CH12 Problem 7a, Sudkamp**

Describe the mapping defined by $add \circ (mult \circ (id, id), add \circ (id, id))$

Problem 9. **CH13 Problem 1c**

Using only the basic functions, composition, and primitive recursion, show that $f(x) = 2x + 2$ is primitive recursive. Give the function *g* and *h*.

Problem 10. **CH13 Problem 6a**

Show that $\max(x, y) = \begin{cases} x & if \quad x \geq y \\ y & otherwise \end{cases}$ is primitive recursive.

NAME THE LANGUAGE

Problem 11. For each of the languages below, give the smallest complexity class that contains it [i.e. Regular, Deterministic Context Free, Context Free, Turing Machines (Recursive)]. Assume an alphabet of {0,1} unless other specified. You do not need to prove your answers.

a. $\{0^n1^m0^p1^q \mid n+m = p+q \text{ and } n,m,p,q > 0\}$

b. $\{0^n1^m0^m1^n \mid n,m > 0\}$

c. $\{0^n1^m0^p1^q \mid n,m,p,q > 0\}$

d. The set of strings over alphabet {0,1,2} with an equal number of 0s and 2s or an equal number of 0s and 1s.

e. $\{0^m \mid m = 2k+1 \text{ where } k>0\}$

f. The set of strings with 3n 0s and 4m 1s for m,n > 0.

g. The set of strings with at least ten times as many 0s as 1s.

h. The set of strings that are either odd length or contain 5 consecutive 1s.

i. $\{0^m10^{m!} \mid m>0\}$

j. The set of stings over alphabet {0,1,2} where the number of 1s equals the number of 2s and every 0 is followed immediately by at least one 1.

R.E. OR NOT?

Problem 12. Determine for each of the following languages whether or not it is recursively enumerable and whether the complement is or is not recursively enumerable. Give justification for your answers.

a. The language of all TMs that accept nothing.
b. The language of all TMs that accept everything.
c. The language of all TMs that accept Regular languages.
d. The language of all PDAs that accept everything.
e. The language of all CFGs that are ambiguous.

SOLUTIONS

RECURSIVELY ENUMERABLE LANGUAGES & TURING MACHINES

TURING MACHINES

Problem 1. **CH9 Problem 24**

Prove that a language L is recursive if and only if L and $\overline{L}$ are recursively enumerable.

Proof

We have two directions:

$\Rightarrow$

L is recursive. This means that L is accepted by a Turing machine M which always halts. We have to show that L and $\overline{L}$ are recursively enumerable. Since every recursive language is recursively enumerable, L is recursively enumerable. To accept $\overline{L}$ we just switch the accepting and non-accepting states of M. This makes $\overline{L}$ recursively enumerable as well.

$\Leftarrow$

L and $\overline{L}$ are recursively enumerable. This means that L is accepted by a standard Turing Machine M_1 and that $\overline{L}$ is accepted by another standard Turing machine M_2. We have to show that L is recursive, so that it is accepted by a standard Turing Machine M which always halts. To get M, we "run" M_1 and M_2 in parallel, say we have two tapes and we run M_1 on tape 1 and M_2 on tape 2. M accepts if M_1 accepts, and M rejects if M_2 accepts. Note that since either M_1 or M_2 accepts, thus M always halts.

Furthermore, $L(M) = L(M_1) = L$. So L is recursive.

Problem 2. **CH9 Problem 25 (Se even Sipser 3.14-3.15)**

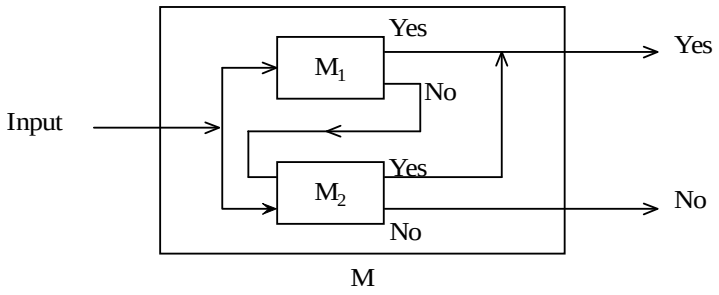
Prove that the recursive languages are closed under union, intersection, and complement.

Proof

Let L_1 and L_2 be recursive languages. Then there are Turning Machines M_1 and M_2, such that $L(M_1) = L_1$, $L(M_2) = L_2$ and both M_1 and M_2 halt on every input string. We construct another Turing machine M based on M_1 and M_2 to simulate union, intersection, and complement. If there is a TM M which accept the language of $L_1 \cup L_2$, $L_1 \cap L_2$, and $\overline{L}$, whenever there is a TM which accepts L, L_1 and L_2, then recursive languages are closed under each operations.

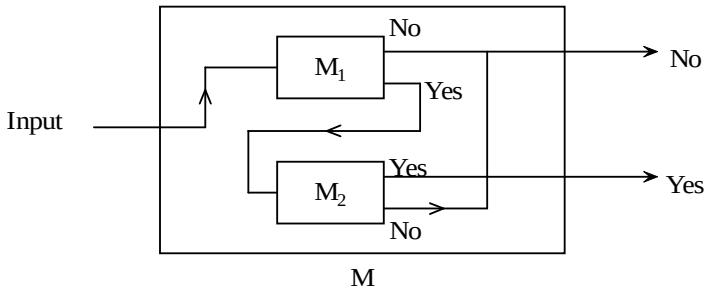
We construct M as following:

a) Union:



$L(M) = L_1 \cup L_2$

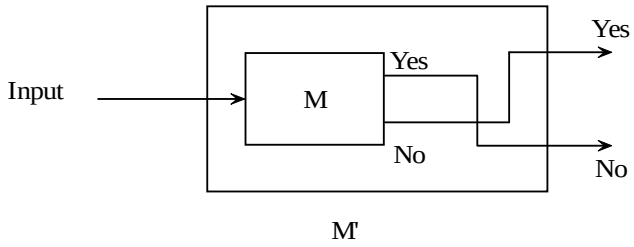
b) Intersection:



$L(M) = L_1 \cap L_2$

c) Complement:

We just switch the accepting and non-accepting states of *M* to construct the new TM *M'*. Then we have $L(M') = \overline{L}$.



Therefore, recursive languages are closed under union, intersection, and complement.

Problem 3. **CH10 Problem 4a-d**

Prove that the recursively enumerable languages are closed under union, intersection, and concatenation, and Kleene star operations.

Proof

a) Union operation

Assume L_1 and L_2 are recursively enumerable. We have two unrestricted grammars G_1 and G_2, such that

$L(G_1) = L_1$ and $L(G_2) = L_2$.

$G_1 = (V_1, \Sigma_1, P_1, S_1)$

$G_2 = (V_2, \Sigma_2, P_2, S_2)$

We can assume without less of generality that $V_1 \cap V_2 = \Phi$.

A String $w \in L(G)$ if there is a derivation

$$S \Rightarrow S_i \overset{*}{\underset{Gi}{\Rightarrow}} w \text{ for } i = 1 \text{ or } i = 2$$

Where *G* is defined as $G = (V_1 \cup V_2 \cup \{S\}, \Sigma_1 \cup \Sigma_2, P_1 \cup P_2 \cup \{S - S_1 | S_2\}, S)$.

L (G) being generated by an unrestricted grammar is obviously recursively enumerable.

b) Intersection operation

Assume L_1 and L_2 are recursively enumerable. We can consider two single tape machines M_1 and M_2, which accept L_1 and L_2 respectively. We define M as a single tape TM with three tracks. Track 1 will hold the input. M will simulates M_1 using track 2. If M_1 halts in an accepting configuration M moves the tape head back to the left end and starts simulating M_2 or track 3. If M_2 also accepts the input string then the string is accepted by M.

c) Concatenation operation

We use the same assumptions for L_1 and L_2 as in problem 4a). We define G as being

$G = (V_1 \cup V_2 \cup \{S\}, \sum_1 \cup \sum_2, P_1 \cup P_2 \cup \{S \to S_1 S_2\}, S)$

We have: $S \Rightarrow S_1 S_2 \overset{*}{\Rightarrow} uS_2 \Rightarrow uv$ where $u \in L_1, v \in L_2$ (leftmost derivation).

The derivation of u uses only rules from G and the derivation of v uses only rules from G (since $V_1 \cap V_2 = \Phi$) ==> $L(G) \subseteq L_1 L_2$.

If $w \in L_1 L_2$ it is possible to write it as uv with $u \in L_1, v \in L_2$. The derivations $S_1 \overset{+}{\Rightarrow}_{G_1} u$ and $S_2 \overset{+}{\Rightarrow}_{G_2} v$ together with $S \to S_1 S_2$ generate w in G ==> $L_1 L_2 \subseteq L(G)$.

$L(G)$ being unrestricted it follows that $L(G)$ is recursively enumerable.

d) Kleene Star operation

Let us define $G = (V_1, \Sigma_1, P_1, S_1)$ such that

$V_1 = V \cup \{S_1\}$

$\Sigma_1 = \Sigma$

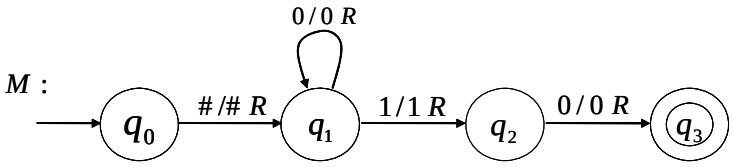
$P_1 = P \cup \{S_1 \to SS_1 | \lambda\}$

Where $G = \{V, \Sigma, P, S\}$ is an unrestricted grammar corresponding to L. the rule $S_1 \to SS_1 | \lambda$ generates any number of copies of S. Each S generates a string in L. The concatenation of any number of strings in L represents L^*.

==> $L_1 = L^*$ is recursively enumerable.

Problem 4. **CH11 Problem 6**

Given a Turing Machine M what's *L (M)*?

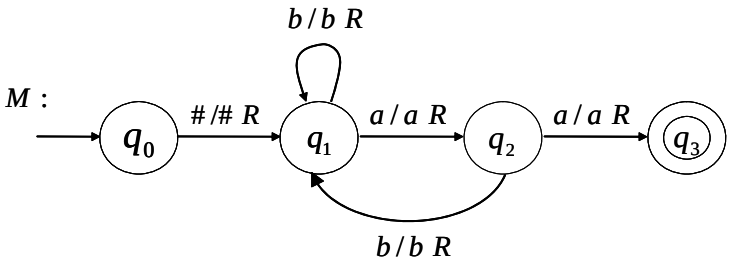


Solution

$L(M) = 0^{*}10(1 \cup 0)^{*}$

Problem 5. **CH9 (9.2.1)**

Given a Turing Machine M, what's *L (M)*?



Solution

$L(M) = (a \cup b)^{*}\, aa\, (a \cup b)^{*}$

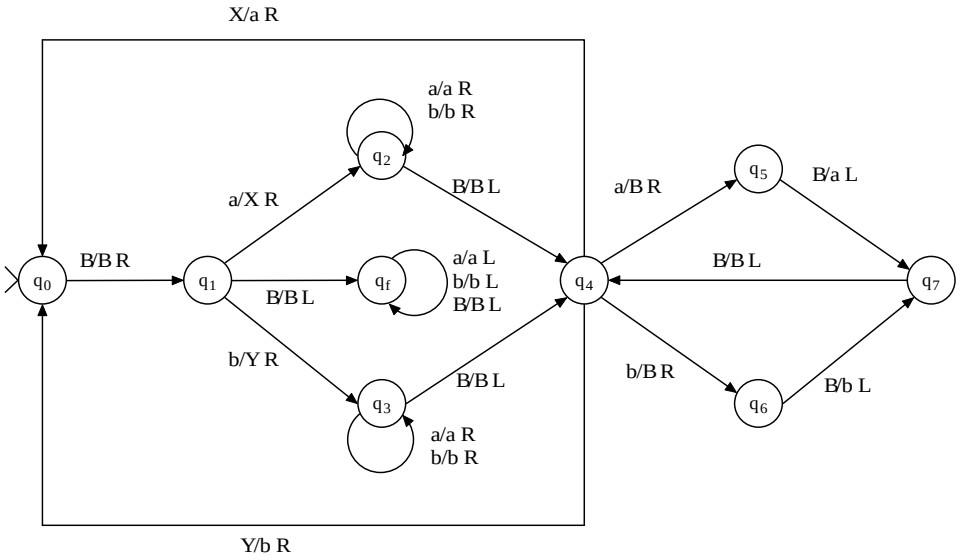
Problem 6. **CH9 Problem 3c (Sudkamp)**

Construct a Turing Machine with input alphabet $\{a, b\}$ to perform:

c) Insert a blank between each of the input symbols.

Solution

TM:



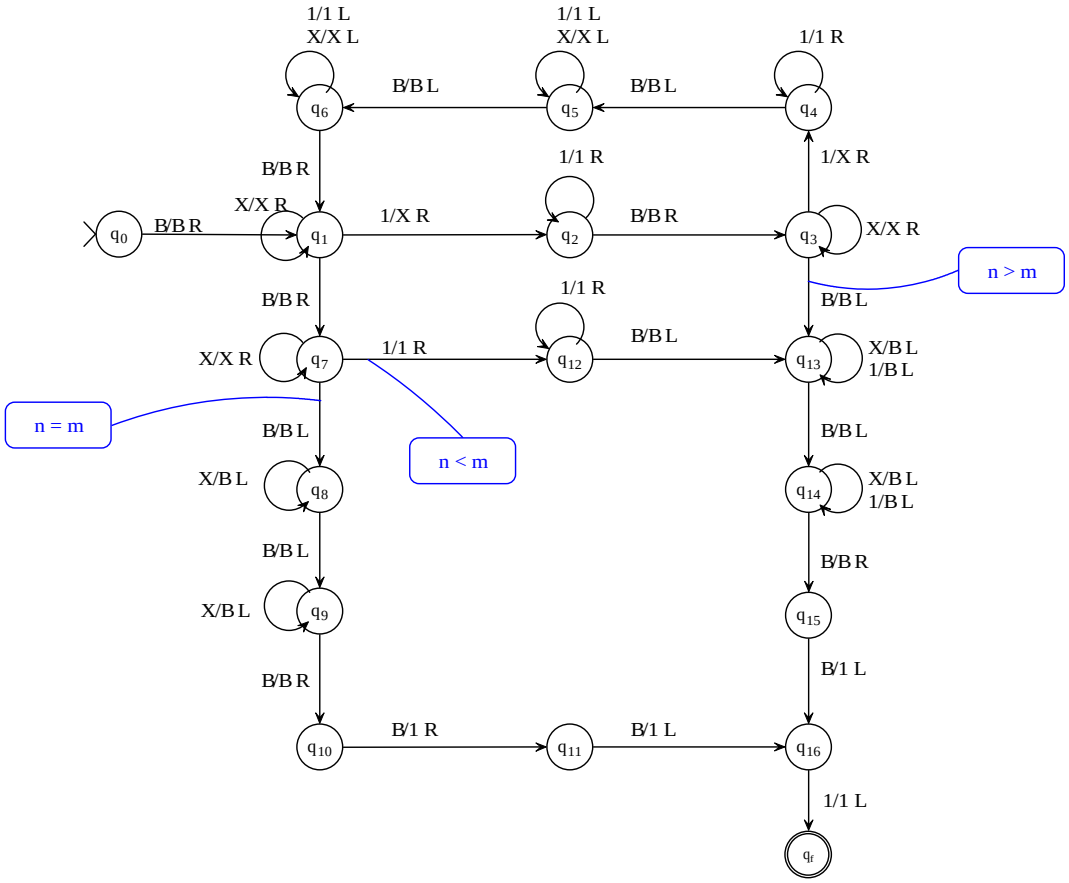
The computation of TM is skipped.

Problem 7. **CH12 Problem 2e**

Construct a TM that computes the number-theoretic function

$$eq(n,m) = \begin{cases} 1 & if \quad n = m \\ 0 & otherwise \end{cases}$$

Solution



Problem 8. **"Shifting over"**

A common operation in Turing-machine programs involves "shifting over." We need to move the contents of each of the cells to the right of the current head position one cell right, and then find our way back to the current head position.

Give the high level description of how to perform this operation.

Hint: Leave a special symbol to mark the position to which the head must return.

Solution

(a) "On input string w:

1) We maintain a set of states corresponding to the symbol we just read.
2) Mark the first symbol with a $ (assume $ $\notin \sum$) and go on to the state corresponding to that symbol read.
3) Move the head toward right till the end of the string (till you hit #) while replacing the current symbol with the previous symbol (this information is contained in the state the you are currently in), shifting to the state corresponding to the symbol just read.
4) After hitting the # at the end of the input, replace the #with the last symbol and start going left back to the start position.
5) Clear the # symbol and move right one position to place the head right on top of the first symbol of the shifted string.
6) Move one position backwards without doing anything to the tape to place the head right on top of the original start position, *accept*."
7) For any situation not described above, *reject*.

Problem 9. Design Turing Machine that…

Solution

a) computes the function Ratio(x, 2) for arbitrary natural number x in binary representation. Input alphabet is {1} and tape alphabet {1, X, #}.

b) in unary representation decides if one natural number is bigger then the other. ("Decide" means that given $\#1^m\#1^n$, M stops and returns **YES** if **m > n**, and stops with output **NO** if m≤n.) Run the TM to illustrate its function.

c) accepts the language of odd integers written in binary.

Hint: To accept odd-valued binary strings, we only have to look at the last bit. The TM moves right until it reads a blank, moves left one space and accepts if and only if there is a 1 on the tape.

d) accepts the language a#b#c, where a, b, c are in {0,1}*, and a + b = c, where a, b and c are interpreted as positive binary integers.

Simulate adding a and b, marking the digits right to left as we go and verifying the digits of c.

- Add a $ to the beginning of the tape and shift the rest of the input.
- Move right past the input string, write another # and a 0. This is the carry bit. Move left until you hit the $.
- Move right to the first #, then move left until you find an unmarked digit.
- Remembering this digit, move right past a #.
- Move right to the second #, move left until you find an unmarked digit.
- Remembering this second digit, move right past the second #.
- Move right past the last #.
- If the two bits read, plus the carry bit, are equal 2 or 3, write a one to the carry bit.
- Move left past the # and stop at the first unmarked bit. If the the remembered bits plus the old carry bit are equal to 1 or 3, check for a 1 under the head, and mark it. Otherwise, check for a zero, and mark it. Reject if the check fails.
- Move left to the $, and repeat from 3.
- When there are no unmarked digits in a and b, move to the carry bit.
- If the carry bit is 1, check for an unmarked 1 in c. Accept as long as there are no unmarked 1s in c.

e) enumerates the language of odd integers written in binary.

This solution is a bit non-intuitive, because it does not construct the odds by adding one each time, but by constructing all possible odd-valued n-bit strings by prepending 0 and 1 to all odd-valued n-1-bit strings. The resulting machine, though, is considerably simpler.

- Write $*1* to the tape
- Move to the left pass a * and until you hit another *.
- Moving right, copy symbols to the end of the tape, replacing the first * with *0, and #s with #0. Stop when you encounter a second *. Don't copy it.
- Move left back past two stars and stop on the third.
- Copy the string between the stars again to the end of the tape, this time prefixing each bit with a 1. Also, this time write a # instead of a leading *. Stop when you get to the second *, but copy it to the end of the tape.
- From the end of the tape, repeat back to step 2.

Problems LINZ. Prove your answer.

Problem 10. **Problem 8, page 257**

Suppose we make the requirement that a Turing machine can halt only in a final state, that is, we ask that δ (q,a) be defined for pairs (q,a) with a $\in \Gamma$ and q $\notin$ F. Does this restrict the power of the Turing Machine? Prove your answer.

Solution

No, the requirement that a Turing machine must halt only in a final state does not restrict the power of the machine. Let's call the modified kind of machines, which halt only on final states, as the "final-halt" machines. We need to prove that any language accepted by a standard Turing Machine is also accepted by a no-halt machine, and vice-versa.

(a) Any language accepted by a standard Turing machine is also accepted by a final-halt machine.

(Proof by construction)
We can convert any standard Turing machine into a final-halt machine using the following method:

1. Create a new "trap state" with transitions to itself for all inputs
2. For each undefined transition from a non-final state, define a transition to the trap state

For all instances where the standard machine previously halted in a non-final state, the final-halt machine now enters an infinite loop in a trap state. Therefore, all previous inputs which where not accepted are still not accepted because the machine loops forever. All input strings previously accepted are still accepted since transitions towards final states are not modified.

(b) Any language accepted by a final-halt machine is also accepted by a standard Turing Machine.

Trivially true, since any final-halt machine is also a standard Turing machine.

Problem 11. **Problem 6, page 270**

Let $\Sigma = \{a, b, c\}$. We can show that $S = \Sigma^+$ is countable if we can find an enumeration procedure that produces its elements in some order, say in the order in which they would appear in the dictionary. However, the order used in dictionaries is not suitable without modification. In a dictionary, all words beginning with *a* are listed before the string *b*. But when there is infinite number of words, we will never reach *b*.

Instead, we can use the modified order, in which we take the length of the string as a first criterion, followed by an alphabetic ordering of all equal-length strings. This is an enumeration procedure that produces the sequence which is called **the proper order**

$$a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, \ldots$$

Find a function *f(w)* that gives for each *w* its index in the proper ordering.

Solution

For any string $w \in \{a, b, c\}^+$, let w_i denote the *i*-th symbol of *w*, namely, $w = w_{|w|}\, w_{|w|-1} \ldots w_2\, w_1$. Assign to letters *a*, *b*, and *c* the respective values 1, 2, and 3. The function *f(w)* that returns the index for all $w \in \{a,b,c\}^+$ in the proper ordering is:

$$f(w) = w_{|w|}3^{|w|-1} + w_{|w|-1}3^{|w|-2} + \ldots + w_2\, 3 + w_1$$

Problem 12. **Problem 12, page 282**

Let L_1 be recursive and L_2 recursively enumerable. Show that $L_2 - L_1$ is necessarily recursively enumerable

Solution

Proof by construction:
There exists a Turing machine M_1 that accepts L_1 and halts on all inputs.
There exists a Turing machine M_2 that accepts L_2 and may not halt for all inputs.
We can construct a new Turing machine M_3 from M_1 and M_2 that accepts $L_2 - L_1$.

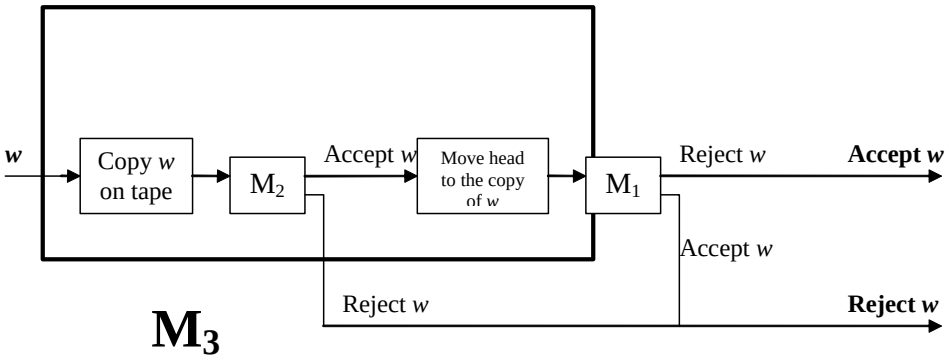
M_3 should accept all strings accepted by M_2 but not accepted by M_1.

Machine M_3 first needs a new Turing machine to duplicate the input string on the tape.
Next M_3 will run M_2 and then M_1 on the copy of the input with the following changes:

- Add transitions from all final states of M_2 (and make them non-final states) to the starting state of M_1 (moving the read-write head to the second copy of the input string in the process). Therefore, if a string is accepted by M_2 it will be tested on M_1.

- Convert all final states of M_1 into non-final states. Furthermore, convert any other hatling states of M_1 to final states. Therefore, if a string is accepted by M_1 it is rejected in M_3, and furthermore if a string is rejected by M_1 it is accepted by M_3.

M_3 will run M_2 and then M_1.



Since M_3 contains M_2 there are must be instances of w for which M_3 does not halt (M_2 doesn't halt for these instances). Therefore, $L_2 - L_1$ is necessarily recursively enumerable. However, if L_2 was recursive then M_3 would always halt and $L_2 - L_1$ would be recursive as well.

Problem 13. **Problem 13, page 282**
Suppose that L is such that there exists a Turing machine that enumerates the elements of L in proper order. Show that this means that L is recursive.

Solution

If there exists a Turing machine that enumerates the elements of L in proper order then it must be a recursive language.

Proof by construction:
By definition, *L* is recursive if and only if there exists a membership algorithm for it.
We will construct a Turing machine *M* that determines membership in language *L*,
namely, for any input string w, machine M determines if w is in L or not.

Let M_1 be the Turing machine that enumerates the elements of *L* in proper order.
Machine *M* compares the input string *w* with each output from M_1.
If string *w* equals an output from machine M_1 then machine *M* returns "yes".
If the length of an output string from M_1 exceeds the length of *w* then *M* returns "no".

This works because strings in proper order are in order of increasing length.
Thus, if M_1 reaches a point where it is producing strings with a length greater than that of *w*, we know that it will never produce *w*. This membership algorithm always produces an answer in finite time (always halts) so *L* must be recursive.

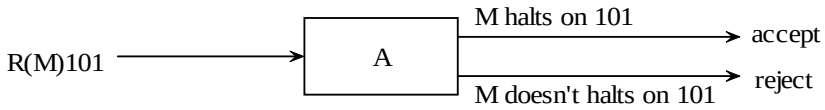
DECIDABILITY

Problem 1. **CH11 Problem 12 Sudkamp**

Prove that there is no algorithm that determines whether an arbitrary Tuning Machine halts when run with the input string 101.

Solution 1 (By Blank Tape Problem):

Assume that there is such a TM A exist:

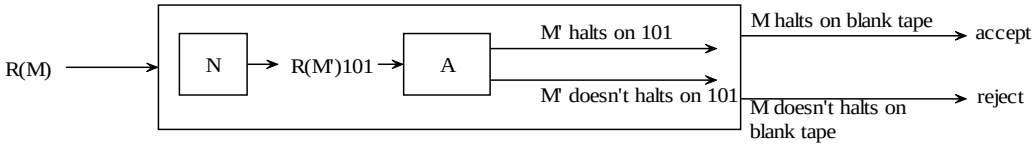


Consider a TM N, which produces a representation of a machine M' where M':

a) Erase any input.

b) Runs M on blank tape

Then M' halts on input string 101 iFF M halts on a blank tape.

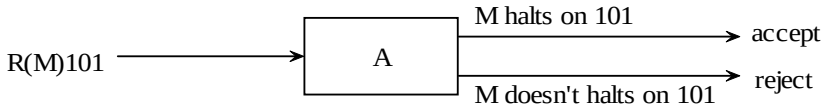


So this solves the blank tape problem but blank tape problem is undecidable.

Therefore, there is no algorithm that determines whether an arbitrary Tuning Machine halts when run with the input string 101.

Solution 2 (By Halting Problem):

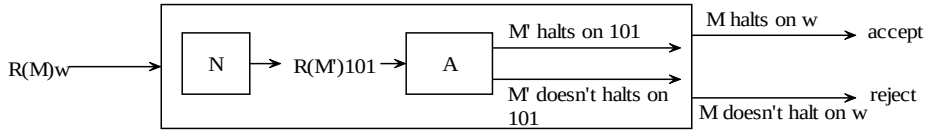
Assume that there is such a TM A exist:



We can reduce the halting problem to this problem by a TM N. The input is the representation of a TM M followed by an input string *w*. The result of a computation of N is the representation of a machine M' that:

1. Replace 101 with *w*.
2. Return the tape head to the initial position with the machine in the initial state of M.

3. Runs M.

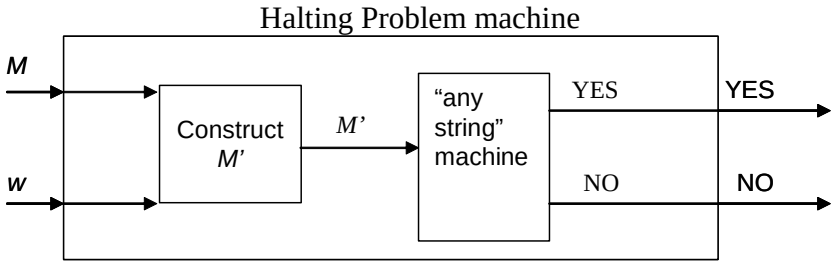


This reduces the halting problem to the "101" problem and the halting problem is undecidable.

Therefore, there is no algorithm that determines whether an arbitrary Tuning Machine halts when run with the input string 101.

Problem 2. Prove that the problem of determining whether for a Turing machine M there is some input string for which M halts is undecidable.

Solution



Let's call the problem for which we want to prove it is undecidable as the "any string" problem. We will reduce the halting problem to the "any string" problem, as shown in the above figure.

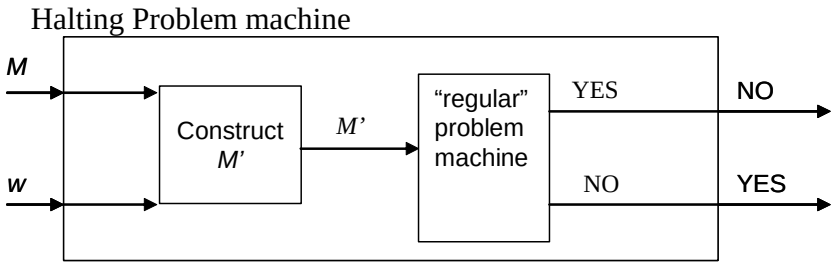
First, we take *M* and *w* and we construct a new machine *M'*, such that *M'* first simulates *M* on input *w*. If during the simulation *M* halts on input w, then *M'* enters a new set of states in which *M'* halts for some input string on the tape of *M'*. However, if during the simulation *M* doesn't halt on input *w*, then *M'* is stuck in the simulation of *M*, and thus *M'* will not halt either, for any input string on the tape of *M'*.

We now have constructed *M'* in such a way that if *M* halts on *w*, *M'* will halt on some string. Otherwise, if *M* doesn't halt on *w*, then *M'* will not halt as well. Therefore, if we now the answer to the question of whether *M'* halts on any input string, then we know the answer to the question of whether *M* halts on *w* or not.

Because we have reduced the halting problem to the "any string" problem, we can conclude that because the halting problem is undecidable, neither is the "any string" problem.

Problem 3. Prove that the problem of determining whether for a Turing machine M the language L(M) is regular is undecidable.

Solution



Let's call the problem which we want to prove is undecidable as the "regular" problem. We will reduce the halting problem to the "regular" problem, as shown in the above figure.

First, we take M and w and we construct a new machine *M'*, such that *M'* first simulates *M* on input *w*. If during the simulation *M* halts on *w*, then *M'* enters a new set of states in which M' accepts a non-regular language, like $a^n b^n$, on the tape of *M'*. Namely, in this case $L(M') = \{a^n b^n\}$ which is non-regular . Otherwise, if during the simulation *M* does not halt on *w*, then *M'* is stuck in the simulation of *M*, and thus *M'* will not halt either. In this case *M'* will not accept any string and thus $L(M') = \varnothing$, which is a regular language since there is a finite automaton with one state which doesn't accept any string.

We now have constructed *M'* in such a way that if *M* halts on *w*, then *L(M')* is a non-regular language. Otherwise, if *M* doesn't halt on *w*, then *L(M') is* a regular language. Therefore, if we know the answer to the question of whether *L(M')* is regular or not, then we know the answer to the question of whether *M* halts on *w* or not.

Therefore, the halting problem can be reduced to the "regular" problem, and because the halting problem is undecidable, so must be the "regular" problem.

Problem 4. Prove each of the following languages decidable or undecidable.

1. $\{(M_1 >, (M_2 >) \mid L(M_1) \cap \overline{L}(M_2) = \varnothing\}$
2. {<M > | L(M) has exactly n elements}
3. {<M > | L(M) is not recognizable}
4. {<M > | M has an even number of states}
5. {<M > | there exists a string x which M accepts in fewer than |x| steps}

Solution

1. Undecidable. We give a reduction from E_{TM}. Suppose that this language is decidable. Let T be a Turing machine that decides it. Then we can construct a Turing machine T´ deciding E_{TM}, that behaves as follows on input <M>.

(a) Simulate T on input <M, M´>, where M´ is a Turing machine recognizing the empty language.
(b) Accept if and only if T accepts.

Suppose that M accepts the empty language. Then T accepts <M, M´>, and so T´ accepts. On the other hand, suppose that M accepts some string. Then T rejects <M, M´>, and so T´ rejects. Thus, T´ *decides* E_{TM}.

2. Undecidable. This is a non-trivial property of languages, so Rice's Theorem applies.

3. Decidable. This is a trick question. L(M) is always recognizable; in particular, it is the language recognized by Turing machine M. Therefore, this is the empty language, which is recognizable.

4. Decidable. Deciding the language requires only inspection of the encoding <M>.

5. Undecidable. We give a reduction from A_{TM} (accept(halting) problem). Suppose that this language is decidable. Let T be a Turing machine that decides it. Then we can construct a Turing machine T´ deciding A_{TM}, that behaves as follows on input <M;w>.

(a) Construct a Turing machine M´ that behaves as follows on input v.
    i. Simulate M on input w.
    ii. Accept if and only if M accepts.
(b) Simulate T on input <M>.
(c) Accept if and only if T accepts.

Suppose that M does not accept w. Then M´ never accepts, the simulation of T rejects, and so T´ rejects. On the other hand, suppose that M does accept w. Then M´ accepts every string v in a number of steps independent of |v| thus, in particular, M´ accepts some string v in fewer thatn |v| steps. Therefore, the simulation of T accepts, and so T´ accepts. Thus T´ decides A_{TM}.

Problem 5. Consider the problem of testing whether a Turing machine M on input w ever attempts to move its head left when its head is on the left-most tape cell. Formulate this problem as a language and show that it is undecidable.

Solution

The proof is by contradiction. Assume that this problem is decidable, then we will show that A_{TM} is also decidable:
A_{TM} = {<M, w> | M is a TM and M accepts w}.
Let R be a Turing machine that decides the problem. That is, R decides the language
LEFTMOST = {<M, w> | M on input w ever attempts to move its head left when it's head is on the left-most tape cell }.

We can construct a Turing machine S that decides A_{TM} as follows. On input <M; w>, S first modifies machine M to M´, so that M´ moves its head to the left from the left-most cell only when M accepts its input. To ensure that during its computation M´ doesn't move the head left from the left-most position, machine M´ first shifts the input w one position to the right, and places a special symbol on the left-most tape cell. The computation of M´ starts with the head on the second tape cell. If during its computation M´ ever attempts to move its head to the left-most tape cell, M´ finds out that it did so by reading the special symbol and then puts the head back to the second cell, and continues its execution. If M would enter an accept state, then M´ enters a loop that forces the head to move always to the left.
After S has constructed M´ it runs the decider R on input < M´; w>. If R accepts then S accepts, otherwise if R rejects then S rejects.

Problem 6. Prove that the problem of determining if the languages generated by two CFGs are equal is undecidable.

Solution

We show this by reducing the problem of determining whether a CFG accepts everything to the problem of determining if the languages generated by two CFGs are equal by taking our input CFG and comparing it to the CFG that generates everything. Since we know from class that the "everything" problem is undecidable, the "equal" probably must also be.

Problem 7. Decidable or not? Given a (code for a) grammar does the language produced by the grammar contains infinitely many strings? Motivate your answer!

d) Is there any TM that can decide the problem for a restriction free grammar?
e) Is there any TM that can decide the problem for a context free grammar?
f) Is there any TM that can decide the problem for a regular grammar?

Solution

a) Undecidable. A restriction free grammar produces exactly the Turing acceptable languages. We use Rices Theorem with $\Omega = \{L \mid L \text{ Turing acceptable and L infinite}\}$. Ω is non-trivial. (E.g. $\Sigma^* \in \Omega$ and $\varnothing \notin \Omega$).
b) Decidable. For a context free grammar there is an algorithm to decide whether it produces an infinite language.
c) Decidable. A regular grammar is a special case of a CFG.

A Refutation of the Halting Problem?

Consider the language of all TMs that given no input eventually write a non-blank symbol on their tapes. Explain why this set is decidable. Why does this not conflict with the halting problem?

The state of a Turing machine is determined by the state of its controller (a DFA) and the state of the tape. Since the controller is finite, there are only a finite number of states possible before the TM is forced to either loop or write a symbol, so we simply run the Turing machine for that number of steps and then accept if it has written a symbol and reject otherwise.

This does not conflict with the halting problem because we are considering only a subset of Turing machines (those roughly equivalent to DFAs) and not really addressing the halting problem for Turing machines.

PRIMITIVE RECURSIVE

Problem 8. **CH12 Problem 7a, Sudkamp**

Describe the mapping defined by $add \circ (mult \circ (id, id), add \circ (id, id))$

Solution:

$$\begin{aligned} f(n) &= add \circ (mult \circ (id, id), add \circ (id, id))(n) \\ &= \text{add} \circ (mult \circ (id(n), id(n)), add \circ (id(n), id(n))) \\ &= add \circ (mult(n, n), add(n, n)) \\ &= add(n^2, 2n) \\ &= n^2 + 2n \end{aligned}$$

Problem 9. **CH13 Problem 1c**

Using only the basic functions, composition, and primitive recursion, show that $f(x) = 2x + 2$ is primitive recursive. Give the function *g* and *h*.

Solutions:

First let's do $f(x) = 2x$, nothing that

$$f(y+1) = f(y) + 2 \text{ and } f(0) = 0$$

Have

$$f(0) = 0 \leftarrow k$$
$$f(y+1) = s \circ s \circ p_2^{(2)}(y, f(y))$$

Now, $f(y) = 2y + 2$ just adds "2", so just need to start with $f(0) = 2$.

Have

$$f(0) = 2 \leftarrow k$$
$$f(y+1) = s \circ s \circ p_2^{(2)}(y, f(y))$$

So

$$g = 2$$
$$h(y, f(y)) = s \circ s \circ p_2^{(2)}(y, f(y))$$

Check:

$$\begin{aligned} f(3) &= h(2, f(2)) \\ &= s \circ s \circ p_2^{(2)}(2, f(2)) \\ &= s \circ s \circ f(2) \\ &= s \circ s \circ s \circ s \circ p_2^{(2)}(1, f(1)) \\ &= s \circ s \circ s \circ s \circ f(1) \\ &= s \circ s \circ s \circ s \circ s \circ s \circ p_2^{(2)}(0, f(0)) \\ &= s \circ s \circ s \circ s \circ s \circ s \circ f(0) \\ &= s \circ s \circ s \circ s \circ s \circ s \circ 2 \\ &= 8 \end{aligned}$$

Therefore $f(x) = 2x + 2$ is primitive recursive.

Problem 10. **CH13 Problem 6a**

Show that $\max(x, y) = \begin{cases} x & \text{if} \quad x \geq y \\ y & \text{otherwise} \end{cases}$ is primitive recursive.

Solution

$$\max(x, y) = x \cdot (gt(x, y) + eq(x, y)) + y \cdot gt(y, x)$$

Check:

$$\begin{aligned} \max(2,3) &= 0 \cdot 2 + 1 \cdot 3 = 3 \\ \max(3,2) &= 1 \cdot 3 + 0 \cdot 2 = 3 \\ \max(2,2) &= 1 \cdot 2 + 0 \cdot 2 = 2 \end{aligned}$$

Therefore $\max(x, y)$ is primitive recursive.

NAME THE LANGUAGE

Problem 11. For each of the languages below, give the smallest complexity class that contains it [i.e. Regular, Deterministic Context Free, Context Free, Turing Machines (Recursive)]. Assume an alphabet of {0,1} unless other specified. You do not need to prove your answers.

a. $\{0^n1^m0^p1^q \mid n+m = p+q \text{ and } n,m,p,q > 0\}$

b. $\{0^n1^m0^m1^n \mid n,m > 0\}$

c. $\{0^n1^m0^p1^q \mid n,m,p,q > 0\}$

d. The set of strings over alphabet {0,1,2} with an equal number of 0s and 2s or an equal number of 0s and 1s.

e. $\{0^m \mid m = 2k+1 \text{ where } k>0\}$

f. The set of strings with 3n 0s and 4m 1s for m,n > 0.

g. The set of strings with at least ten times as many 0s as 1s.

h. The set of strings that are either odd length or contain 5 consecutive 1s.

i. $\{0^m10^{m!} \mid m>0\}$

j. The set of stings over alphabet {0,1,2} where the number of 1s equals the number of 2s and every 0 is followed immediately by at least one 1.

Solution

a. **DCFL**. $\{0^n1^m0^p1^q \mid n+m = p+q \text{ and } n,m,p,q > 0\}$

b. **DCFL**. $\{0^n1^m0^m1^n \mid n,m > 0\}$

c. **REGULAR** . $\{0^n1^m0^p1^q \mid n,m,p,q > 0\}$

d. **CFL**. The set of strings over alphabet {0,1,2} with an equal number of 0s and 2s or an equal number of 0s and 1s.

e. **REGULAR**. $\{0^m \mid m = 2k+1 \text{ where } k>0\}$

f. **REGULAR**. The set of strings with 3n 0s and 4m 1s for m,n > 0.

g. **DCFL**. The set of strings with at least ten times as many 0s as 1s.

h. **REGULAR**. The set of strings that are either odd length or contain 5 consecutive 1s.

i. **TM**. $\{0^m10^{m!} \mid m>0\}$

j. **DCFL**. The set of stings over alphabet {0,1,2} where the number of 1s equals the number of 2s and every 0 is followed immediately by at least one 1.

R.E. OR NOT?

Problem 12. Determine for each of the following languages whether whether or not it is recursively enumerable and whether the complement is or is not.

a. **no yes** The language of all TMs that accept nothing.

   We need to try an infinite number of strings in a TM to determine that it accepts nothing, but we only need to find a single string that it accepts to show that it accepts something.

b. **no no** The language of all TMs that accept everything.

   We need to try an infinite number of string in a TM to determine whether it accepts everything, and we may or may not need to if it does not.

c. **no no** The language of all TMs that accept Regular languages.

   We can enumerate all possible regular languages, but testing every regular language against every TM would take forever. If fact, even testing a TM against a single infinite regular language would take forever.

d. **no yes** The language of all PDAs that accept everything.

   We would have to try every string before declaring that a PDA accepts everything, but since membership in a CFG (equivalent to a PDA) is decidable, we determine that a PDA does not accept anything once it rejects anything.

e. **yes no** The language of all CFGs that are ambiguous.

   We can determine that a CFG is ambiguous by finding a single string which has an ambiguous derivation, but we cannot determine if a CFG is unambiguous unless we try everything string in it.

References

1. Salling: Formella språk, automater och beräkningar 2001
2. Linz, An Introduction to Formal Languages and Automata, Jones & Bartlett 2000
3. Sipser, Introduction to the Theory of Computation, PWS 1997
4. Sudkamp, Languages and Machines, Addison Wesley 1998
5. Lewis-Papadimitriou, Elements of the Theory of Computation, Prentice Hall 1998
6. Kinber-Smith, Theory of Computing, A Gentle Introduction, Prentice Hall 2001