Final Exam Practice Sheet

Introduction to Computer Science, Fall 2018
Final Exam: December 20, 2018 at 16:00 – 18:00

Instructor	Teaching Assistants
Prof. Jürgen Schönwälder	Jonas Bayer
	Marco David
	Dung Huynh (Oscar)
	Irsida Mana
	Abhik Pal

December 17, 2018

These problems have been created by the TAs for your final exam preparation. Accordingly, they neither represent the extent nor necessarily the difficulty of the final exam. We tried to cover all topics that are relevant, however, you should also study the course notes and revise the homework assignments in order to prepare for the exam. We will upload the solutions to these problems on Moodle two days before the exam. In case you have any questions on this problem sheet, please contact your TAs.

In this document you can find sample solutions for all the practice problems. Sometimes, multiple possible solutions are given to illustrate different ways to approach a problem. However, in the exam only one solution is expected.

1 Introductory examples and Discrete Mathematics

Problem 1 Boyer-Moore

Problem 2 Induction Prove that

$$\sum_{k=1}^{n} k^3 = \frac{1}{4}n^2(n+1)^2$$

and use your result to compute

$$\sum_{k=1}^{n}(k^3 - k).$$

Solution We employ induction.

Base Case

$$\sum_{k=1}^{1} k^3 = 1 = \frac{1}{4}1^2(1+1)^2 = 1 \quad \checkmark$$

Inductive Step

$$\begin{aligned}\sum_{k=1}^{n+1} k^3 &= (n+1)^3 + \sum_{k=1}^{n} k^3 \\ &= (n+1)^3 + \frac{1}{4}n^2(n+1)^2 \\ &= \frac{1}{4}(n+1)^2\left(4(n+1) + n^2\right) \\ &= \frac{1}{4}(n+1)^2(n+2)^2\end{aligned}$$

□

Next, we know that

$$\begin{aligned}\sum_{k=1}^{n}(k^3 - k) &= \frac{1}{4}n^2(n+1)^2 - \frac{1}{2}n(n+1) \\ &= \frac{1}{4}n(n+1)\left(n(n+1) - 2\right) \\ &= \frac{1}{4}(n-1)n(n+1)(n+2)\end{aligned}$$

Problem 3 Injectivity and Surjectivity State for each function if it is injective and/or surjective. Compute the inverse if the function is bijective. If the function is not surjective, restrict its codomain to the functions image and make it surjective.

(a) $f_1 : \mathbb{R} \mapsto [-2, 2] \quad f_1(x) = \sin x$

(b) $f_2 : \mathbb{Z} \mapsto \{0, 1, 2, 3\}$ $f_2(x) = x^2 - 5 \mod 4$

(c) $f_3 : \mathbb{R} \mapsto \mathbb{R}$ $f_3(x) = x^3 - 42$

(d) $f_4 : \mathbb{R} \mapsto \mathbb{R}$ $f_4(x) = e^{-x^2}$

Solution

(a) f_1 is neither injective nor surjective. Its image is $[-1, 1]$.

(b) f_2 is not injective and not surjective because all even squares $(2n)^2 - 5 = 4n^2 - 5$ have remainder three, and all odd squares $(2n+1)^2 - 5 = 4(n^2 + n) + 1 - 5$ have remainder zero. Its image is the set $\{0, 3\}$.

(c) f_3 is a bijection with inverse $f_3^{-1}(y) = \sqrt[3]{y + 42}$.

(d) f_4 is, like x^2, not injective and because of the exponential not surjective. It only maps to positive real numbers less than or equal to one (the function's maximum at $x = 0$).

2 Number systems

Problem 4 Wrong conversion to IEEE 754 You are given a program that is supposed to convert numbers into their IEEE 754 representations. You enter the values `23`, `-45` and `0.4` and the program outputs returns the following IEEE 754 representation in hexadecimal:

```
0x41dc0000
0xc25a0000
0x3ee66666
```

Are these values correct? If not, what is wrong about the program? To investigate the issue you can first convert the numbers back to binary. Then write them in a formatted way so it becomes clear what sign bit, biased exponent and mantissa are:

```
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|S|    exponent   |             mantissa (23 bits)              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Look at the representations carefully and see if there is anything suspicious. If you do not find anything, convert one of the numbers to their IEEE 754 representation and compare with what the program gave you.

Solution First we convert the numbers back to their binary representation:

```
             S exponent ------- mantissa ------
0x41dc0000 = 0 10000011 10111000000000000000000
0xc25a0000 = 1 10000100 10110100000000000000000
0x3ee66666 = 0 01111101 11001100110011001100110
```

We see that the first bit of the mantissa is always `1`, so maybe the program did not remove the leading 1 after normalization. One can calculate that the correct representations of `23`, `-45` and `0.4` are:

```
S exponent ------- mantissa ------
0 10000011 01110000000000000000000
1 10000100 01101000000000000000000
0 01111101 10011001100110011001101
```

Which are identical to the numbers that the program returned if one removes the leading 1's.

3 Boolean Algebra

4 Computer Architecture

Let's define the nth register Rn (equivalent to the nth instruction In) the accumulator ACC. For example, register 2 shall be R2. Also, for the brevity of descriptions, one can assume that when stating ACC or Rn, it means the value in the accumulator or that particular register.

Problem 5 Very Simple Assembly Program Write an assembly program that "exhausts" your machine by making it run the program forever. In other words, how can this simple machine be stuck in an infinite loop?

Solution The star * stands for anything. This program can be interpreted such that the instructions 1-15 can be anything because it doesn't matter as the program is stuck in Instruction 0 jumping to itself endlessly.

#	Machine Code	Assembly Code	Description
0	110 1 0000	JUMP # 0	Jump to I0
1-15	*** * ****	***** **	**********

Problem 6 Simple Assembly Program Given the following program as a table below, fill in the corresponding assembly code and description for each instruction. Then, describe, in plain English, what the purpose behind this program is.

#	Machine Code	Assembly Code	Description
0	001 0 1111		
1	101 1 1101		
2	111 1 0000		
3	001 0 0001		
4	100 1 0001		
5	010 0 0001		
6	001 0 1111		
7	011 1 0001		
8	010 0 1111		
9	001 0 0111		
10	011 1 0001		
12	010 0 0111		
13	110 1 0000		
14	000 0 0000		no instruction, initialized to 0
15	000 0 0000		no instruction, initialized to 0

Solution At every iteration, R1 decreases by 1 (I3-I5) and R15 increases by n (I6-I8) since the instruction is modified. The program checks to see whether the condition in I1 holds, if yes, halt, else skip to the next instruction.

#	Machine Code	Assembly Code		Description
0	001 0 1111	LOAD	15	Load R15 into ACC
1	101 1 1101	EQUAL	# 13	If ACC is 13,
2	111 1 0000	HALT		then: stop the program
3	001 0 0001	LOAD	1	else: load R1 into the ACC
4	100 1 0001	SUB	# 1	Subtract 1 from ACC
5	010 0 0001	STORE	1	Store ACC into R1
6	001 0 1111	LOAD	15	Load R15 into ACC
7	011 1 0001	ADD	1	Add 1 to ACC
8	010 0 1111	STORE	15	Store ACC to R15
9	001 0 0111	LOAD	7	Load R7 into ACC
10	011 1 0001	ADD	1	Add 1 to ACC
12	010 0 0111	STORE	7	Store ACC to R7
13	110 1 0000	JUMP	# 0	Jump to I0
14	000 0 0000			no instruction, initialized to 0
15	000 0 0000			no instruction, initialized to 0

R15 value gradually becomes 1=0+1, 3=1+2, 6=3+3, 10=6+4. The first term is R15 itself while second term of the addition comes from R7 which increases by 1 at every iteration (I9-I12). Hence, after 4 iterations, I1 is EQUAL #10 and R15 is 10, and the program terminates.

Problem 7 Another Simple Assembly Program Write an assembly program (similar to that in the lecture slides) that performs the task defined by the following pseudo-code:

```
x = 30, i = 0
while i < x:
    x = x - 5
    i = i + 5
```

```
x = x - i
```

What will the value of x (at R15) after the program terminates?

Solution Let `x` be stored in R15 and `i` be stored in R14.

#	Machine Code	Assembly Code		Description
0	001 0 1110	LOAD	14	Load R14 into ACC
1	101 0 1111	EQUAL	15	Compare ACC with R15
2	110 1 1001	JUMP	#9	if equal, jump to I9
3	011 1 0101	ADD	#5	else, add 5 to ACC
4	010 0 1110	STORE	14	Store ACC to R14
5	001 0 1111	LOAD	15	Load R15 into ACC
6	100 1 0101	SUB	#5	else, subtract 5 from ACC
7	010 0 1111	STORE	15	Store ACC to R15
8	110 1 0000	JUMP	#0	Jump to I0
9	100 0 1110	SUB	14	Subtract R14 from ACC
10	111 1 0000	HALT		Terminates
12	000 0 0000			no instruction, initialized to 0
13	000 0 0000			no instruction, initialized to 0
14	000 0 0000			initialize i to 0
15	000 1 1110			initialize x to 30

After the program terminates, x becomes 0 because the loop exits when i = x = 15. Then, x = x - i = 15 - 15 = 0.

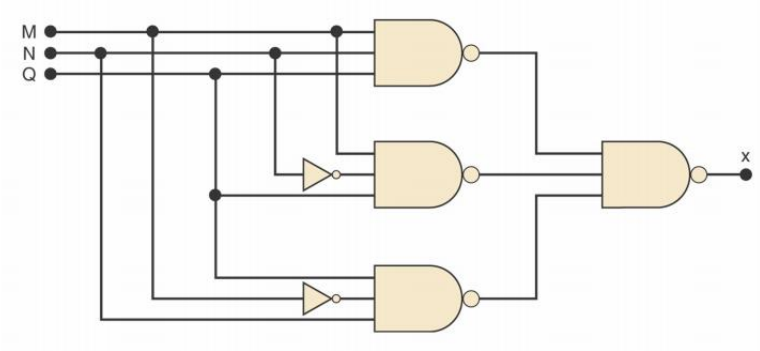
Problem 8 Slightly More Complicated Assembly Program Due to some technical problems, your simple machine has failed to recognize the instruction ADD with a register; however, ADD with a constant still works perfectly fine. Write an assembly program (similar to that in the lecture slides) which **does not use ADD with a register** and performs the task defined by the following pseudo-code:

```
x = 128
y = 40
x = x + y
```

Solution Let `x` be stored in R13 and `y` be stored in R14.

#	Machine Code	Assembly Code		Description
0	001 0 1110	LOAD	14	Load R14 into ACC
1	010 0 1111	STORE	15	Store ACC in R15
2	001 0 1111	LOAD	15	Load R15 into ACC
3	101 1 0000	EQUAL	#0	Compare ACC with 0
4	111 1 0000	HALT		if equal, terminates
5	100 1 0001	SUB	#1	Subtract 1 from ACC
6	010 0 1111	STORE	15	Store ACC in R15
7	001 0 1101	LOAD	13	Load R13 into ACC
8	011 1 0001	ADD	#1	Add 1 to ACC
9	010 0 1101	STORE	13	Store ACC into R13
10	111 1 0010	JUMP	#2	Jump to I2
11	000 0 0000			no instruction, initialized to 0
12	000 0 0000			no instruction, initialized to 0
13	100 0 0000			initialize x to 128
14	001 0 1000			initialize y to 40
15	000 0 0000			no instruction, initialized to 0

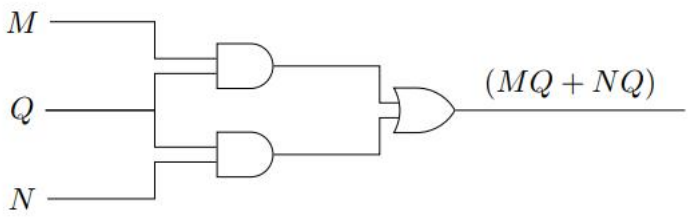
Problem 9 Logic Circuits Write down the truth table of the logic circuit below, then the corresponding sum-of-products expression. Simplify the expression using Boolean Algebra Equivalence Laws and finally draw the logic circuit corresponding to the simplified expression.



M	N	Q	$A = \overline{(MNQ)}$	$B = \overline{(M\overline{N}Q)}$	$C = \overline{(\overline{M}NQ)}$	$\overline{(ABC)}$
0	0	0	1	1	1	0
0	0	1	1	1	1	0
0	1	0	1	1	1	0
0	1	1	1	1	0	1
1	0	0	1	1	1	0
1	0	1	1	0	1	1
1	1	0	1	1	1	0
1	1	1	0	1	1	1

Solution

$$
\begin{aligned}
x &= \bar{M}NQ + M\bar{N}Q + MNQ \\
&= \bar{M}NQ + MQ(\bar{N} + N) \\
&= \bar{M}NQ + MQ(1) \\
&= \bar{M}NQ + MQ \\
&= Q(\bar{M}N + M) \\
&= Q(M + \bar{M})(M + N) \\
&= Q(1)(M + N) \\
&= Q(M + N) \\
&= QM + QN
\end{aligned}
$$



5 Operating System

Problem 10 Forking Consider the following C program `foo`. Find a closed-form expression that gives the number of processes (children and initial parent process) as a function of the arguments passed to the call `./foo`. Also draw the fork-tree for one iteration of the loop.

```
#include <unistd.h>

int main(int argc, char *argv[])
{
    int x = argc * argc;
    for (; x > 1; x--) {
        int pid1 = fork();
        int pid2 = fork();
        if (0 == pid1 * pid2) {
            if (pid1 + fork() == 0) {
                (void) fork();
            }
        }
    }
    return 0;
}
```

Solution With every iteration, the lines initializing `pid1, pid2` create three child threads (because they are sequentially executed). Four threads including the original main thread exist now. In all three child threads the product `pid1 * pid2` will be zero, so they will each create another child thread. Now seven threads exist in total. Finally, the sum `pid1 + fork()` will be zero in two threads, the child of the 1st-order child thread created when initializing `pid1` as well as the child of the previously mentioned thread's child created when initializing `pid2`. Two further forks occur 1–within one iteration the total number of threads has increased ninefold. Since `x = argc * argc`, the final result of the number of threads N in terms of the number of arguments a is

$$N(a) = 9^{a^2}.$$

Problem 11 Forking and Sleeping What does the following program do? How many children processes are there? What is the purpose of `sleep(x)`? What if we remove the `else` condition?

```
int main(int argc, char *argv[]) {
    unsigned int x = 3;
    while(--x) {
        if (!fork()) continue;
        else sleep(x);
    }
    return 0;
}
```

Solution At every iteration of the `while`, `x` decreases by 1. If `x` is 0, exits the loop. The current process creates a new process through `fork()`. If the process is the parent, sleeps for `x` seconds, otherwise, it continues to next iteration.

There are 3 children processes in total. If the `else` was not included, those processes might become zombies if their executions take longer times than their parents.

6 Automata and Formal Languages

Problem 12 Detecting integers In the lecture notes (p. 204) you can find a Finite State machine that decides if a given input string is an integer. The input strings that we consider here are generated from the alphabet $\Sigma = \{-, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}$. Remember, that in Homework 4 we introduced the Kleene closure Σ^* that contains all words that can be built out of elements of Σ as well as the empty word `""` which we denote by ε. Which of the following statements are true?

(a) $\Sigma \subset \Sigma^*$

(b) The word `-230-4` is in Σ^* but not in Σ

(c) The word `-0` is accepted by the finite state machine in the lecture notes

(d) The finite state machine accepts the empty word ε.

While the given FSM does not accept numbers like `-2-3`, it still accepts an input such as `007`, which is not how we would normally write an integer. Modify the FSM such that it does not accept numbers with leading 0's. To do so, you need to use additional states. Draw the new FSM as a graph and also give it in mathematical notation.

Solution

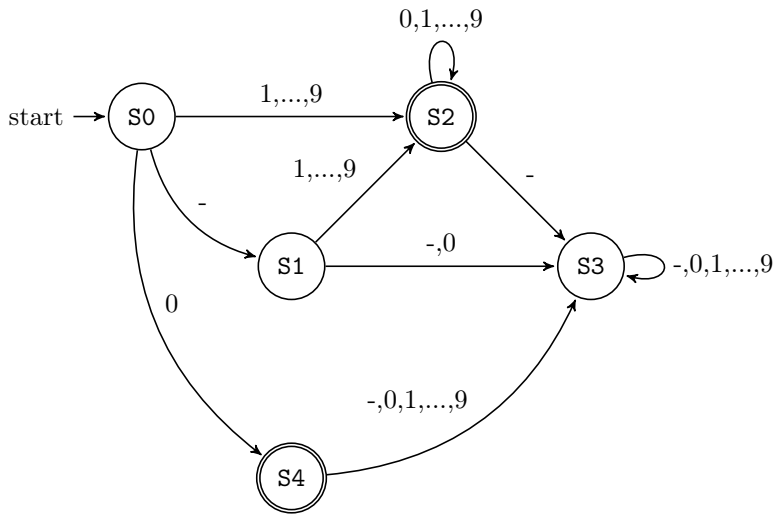
(a) It is true that $\Sigma \subset \Sigma^*$. With $\Sigma*$ we refer to any words that can be generated from Σ, which are in particular the one-letter words.

(b) It is true that the word `-230-4` is in Σ^* but not in Σ

(c) The FSM from the lecture notes indeed accepts the word `-0`. Initially the FSM is in state `S0`. Then it recognizes the first symbol `-` and goes to state `S1`. In the next step, the FSM goes to state `S2` and stops. Therefore, as `S2` is an accepting state the word `-0` is accepted by the FSM.

(d) It is not true that the finite state machine accepts the empty word ε. In this case, the FSM just stays in `S0` which is not an accepting state.

One can modify the FSM by just adding one more state `S4`:



This FSM is formally described as $(\Sigma, S, s_0, \delta, F)$ where

$$\Sigma = \{-, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$
$$S = \{\texttt{S0, S1, S2, S3, S4}\}$$
$$s_0 = \texttt{S0}$$
$$F = \{\texttt{S2, S4}\}$$

and $\delta\colon \Sigma \times S \to S$ is given by

$$\begin{aligned}
\delta(\texttt{S0}, \texttt{-}) &= \texttt{S1} \\
\delta(\texttt{S0}, \texttt{0}) &= \texttt{S4} \\
\delta(\texttt{S0}, \texttt{1}) &= \ldots = \delta(\texttt{S0}, \texttt{9}) = \texttt{S1} \\
\delta(\texttt{S1}, \texttt{-}) &= \texttt{S3} \\
\delta(\texttt{S1}, \texttt{0}) &= \texttt{S3} \\
\delta(\texttt{S1}, \texttt{1}) &= \ldots = \delta(\texttt{S1}, \texttt{9}) = \texttt{S2} \\
\delta(\texttt{S2}, \texttt{-}) &= \texttt{S3} \\
\delta(\texttt{S2}, \texttt{0}) &= \ldots = \delta(\texttt{S2}, \texttt{9}) = \texttt{S2} \\
\delta(\texttt{S3}, \texttt{-}) &= \texttt{S3} \\
\delta(\texttt{S3}, \texttt{0}) &= \texttt{S3} \\
\delta(\texttt{S3}, \texttt{1}) &= \ldots = \delta(\texttt{S3}, \texttt{9}) = \texttt{S3} \\
\delta(\texttt{S4}, \texttt{-}) &= \texttt{S3} \\
\delta(\texttt{S4}, \texttt{0}) &= \texttt{S3} \\
\delta(\texttt{S4}, \texttt{1}) &= \ldots = \delta(\texttt{S4}, \texttt{9}) = \texttt{S3}
\end{aligned}$$

Problem 13 Unary numbers Consider the alphabet $\Sigma = \{1\}$. Then its Kleene closure

$$\Sigma^* = \{\varepsilon, 1, 11, 111, \ldots\}$$

can be considered as natural numbers in the unary number system, where

$$\begin{aligned}
\varepsilon &= 0_{10} \\
1 &= 1_{10} \\
11 &= 2_{10} \\
111 &= 3_{10} \ \ldots
\end{aligned}$$

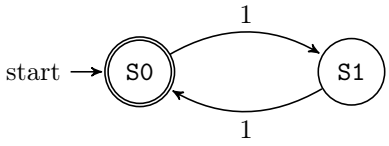
We are interested in finding out whether this number is divisible by two.

Can this property be tested by a finite state machine? If not, can a pushdown automaton or a Turing machine find out if the number is divisible by two? Give a graph representation and the formal representation of the machine.

Solution A finite state machine suffices to find out if a given number in unary is divisible by two. It can be defined as $(\Sigma, S, s_0, \delta, F)$ where

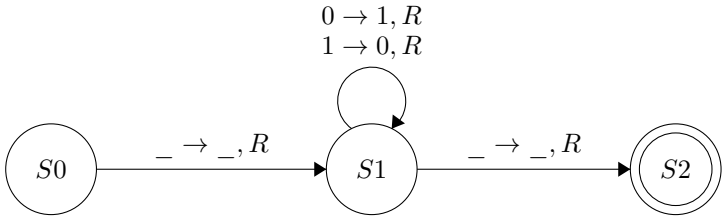
$$\begin{aligned}
\Sigma &= \{1\} \\
S &= \{\texttt{S0, S1}\} \\
s_0 &= \texttt{S0} \\
\delta(\texttt{S0}, 1) &= \texttt{S1} \quad \text{and} \\
\delta(\texttt{S1}, 1) &= \texttt{S0} \\
F &= \{\texttt{S0}\}
\end{aligned}$$

The corresponding graph is given below:



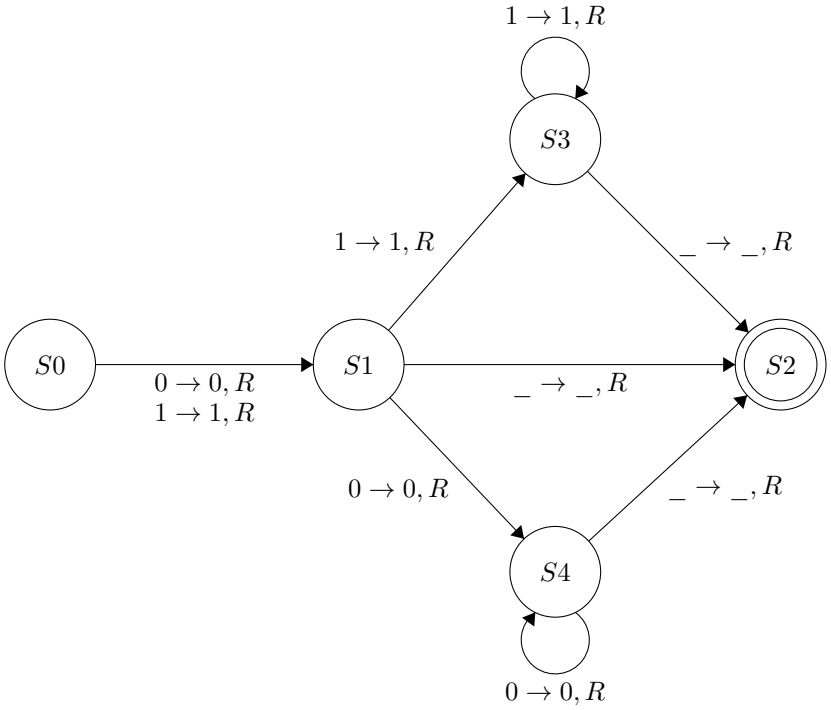
Problem 14 Inverting Turing Machine Consider the alphabet $\Sigma = \{0, 1\}$, construct a Turing Machine that inverts an inputted binary number which is formatted such as _0011_; the size of the string varies. Please note that the machine will not accept the input if the binary string is enclosed by any other symbols beside the empty spaces.

Solution



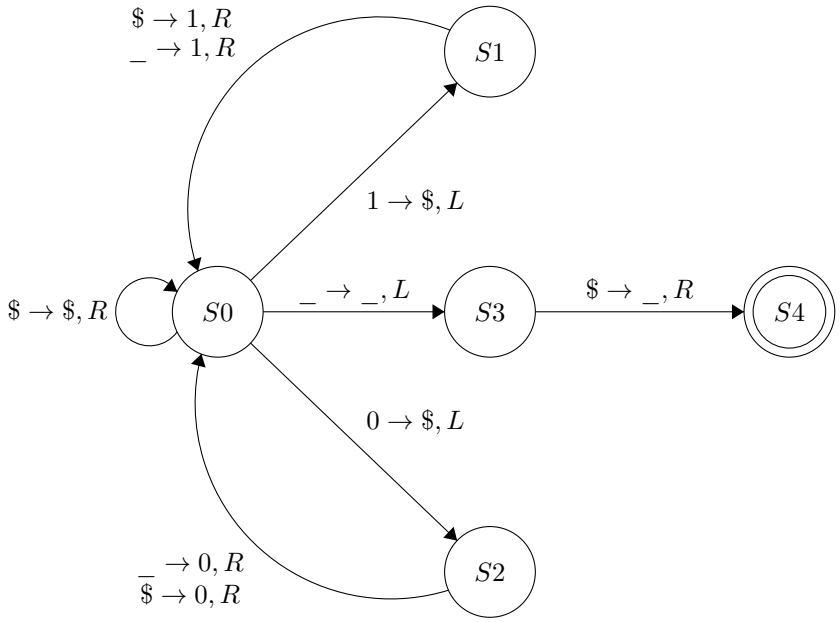
Problem 15 Trailing Zeroes Or Trailing Ones Turing Machine Consider the same alphabet $\Sigma = \{0, 1\}$, construct a Turing Machine that accepts either $0, 1$ for the first symbol, but only zeroes, ones, or nothing trailing. For example, 111111, 1000, 11, 1, 000000, 011, 00, 01, 0 are some words in this language; however, 0010, 00001, 101, 1110 are not. You can safely assume that the surroundings of an input is always empty space, denoting by "_".

Solution



Problem 16 Left-Shift Turing Machine Consider the same alphabet $\Sigma = \{0, 1\}$, construct a Turing Machine that shifts an input binary string to the left. You can safely assume that the surroundings of an input is always empty space, denoting by "_" and the first read symbol is a digit, not an empty space. For example, "__0001001__" will become "_0001001___" and 0 is read first.

Solution



7 Haskell

Problem 17 Folding In Haskell, the operator `++` concatenates two lists. Remember the functions fold left and fold right from homework 9. How is the expression

```
foldr (+) 0 $ foldr (++) [2] [[1, 2], [3, 4], 5]
```

evaluated? Next, remember the duality theorems from the same assignment:

```
(1) foldr op e xs = foldl op e xs
(2) foldr op1 e xs = foldl op2 e xs
(3) foldr op1 a xs = foldl op2 a (reverse xs)
```

Does `(1)` hold if `op` is `+` or `++`?