# Assignment 1

due November 16, 2012

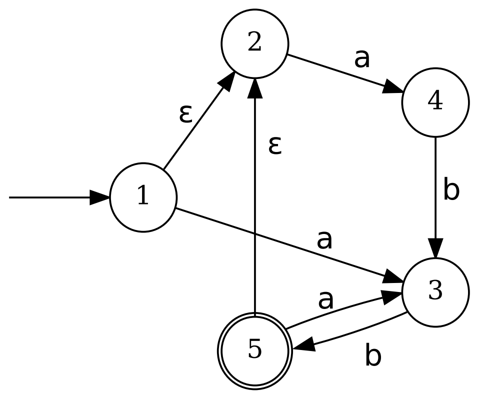
Compiler Design 1 (Kompilatorteknik I) 2012

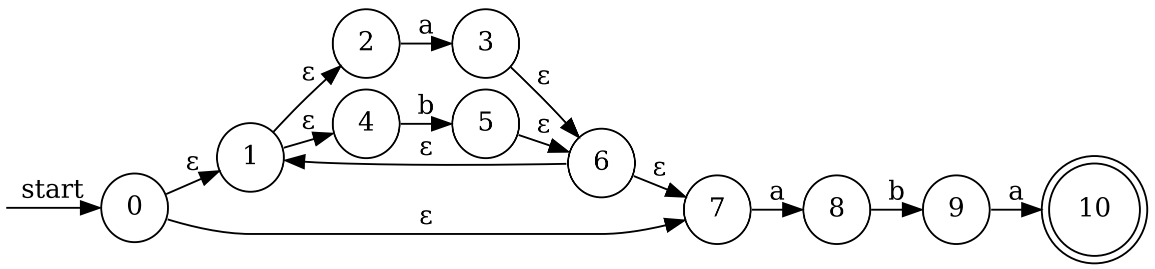
## 1 Automata

Construct a finite state automaton (DFA or NFA) that accepts all the strings that consist of the symbols `0` and `1` where the amount of `0`'s is divisible by 2 (even amount of `0`'s) and the amount of `1`'s is divisible by 3.

## 2 NFA to DFA Conversion

Convert the following NFAs into DFAs:





## 3 Regular Expressions

Describe the languages denoted by the following regular expressions:

1. $a(a|b)^*a$

2. $((\epsilon|a)b^*)^*$
3. $(a|b)^*a(a|b)(a|b)$
4. $a^*ba^*ba^*ba^*$
5. $(aa|bb)^*((ab|ba)(aa|bb)^*(ab|ba)(aa|bb)^*)^*$

## 4 Regular Expressions

Give a regular expression of a simplified C function prototype:

- function prototypes are of the form
  `<type> <name> (<arguments_list>);`
- `<arguments_list>` consist of a list of `<argument>` which are separated by `,`
- `<argument>` consists of a `<type>` and a `<name>`
- `<type>` can be any one of the following types: `int, char, byte, word, void`; it can also be a pointer of a type (denoted by a `*`)
- `<name>` contains only of letters

| **Valid** | **Invalid** |
|---|---|
| `int add(int a, int b);` | |
| `char* cat3(char* a, char * b, char *c);` | `int foo;` |
| `int **allocate2darray(int nx, int ny);` | `int foo(int bar,);` |
| `void markmy(word s);` | `int foo(int bar)` |
| `void dostuff();` | |

## 5 Theory

When you convert a NFA into a DFA, why can the resulting DFA have exponentially as many states as the original NFA? *(Explain in your own words. No mathematical proof is necessary.)*

## 6 Lexical Analysis

In the example we saw in the lesson, we tried to encode information about the tokens that are emitted by the lexical analyzer as outputs for the DFA. In practice, the algorithm is somewhat different: The DFA of the lexical analyzer does not contain extra output information, just accepting states that have an added label about what category of tokens will be emitted if the input ends at these states. Using this DFA we begin to read the input, continuing for as long as we don't fall into the null state. If the next input symbol would lead to a transition in the null state, we instead back up through the sequence of states we entered and, as soon as we find an accepting DFA state, we output a token of the category in its label and start again with the rest of the input (adding of course any symbols we "took back" while backing up). An example is shown in section 3.8.3 of Aho, Lam, Sethi, Ullman *Compilers Principles, Techniques & Tools.*

Let $L$ be a language of strings on $\Sigma = \{x, y\}$ defined by $L = \{x, xxx, (xxxx)*y\}$, that is $\{x, xxx\} \bigcup \{x^{4n}y | n \geq 1\}$.

1. Construct a DFA that accepts L. *(Hint: Begin with an NFA and then convert it)*
2. Describe how the lexical analyzer will tokenized the following inputs:
   - `xx`
   - `xxxxxxxxy`
   - `xxxxxxxxxxxxxy` *(13 x characters)*

## Instructions

There are two ways to submit this assignment:

1. Submit a physical copy of your answers in my mailbox (Andreas Löscher, 99) on the 4th floor of building 1, opposite the 'fika' room.
2. Send an email with an electronic copy of your answers to `andreas.loscher@it.uu.se`.