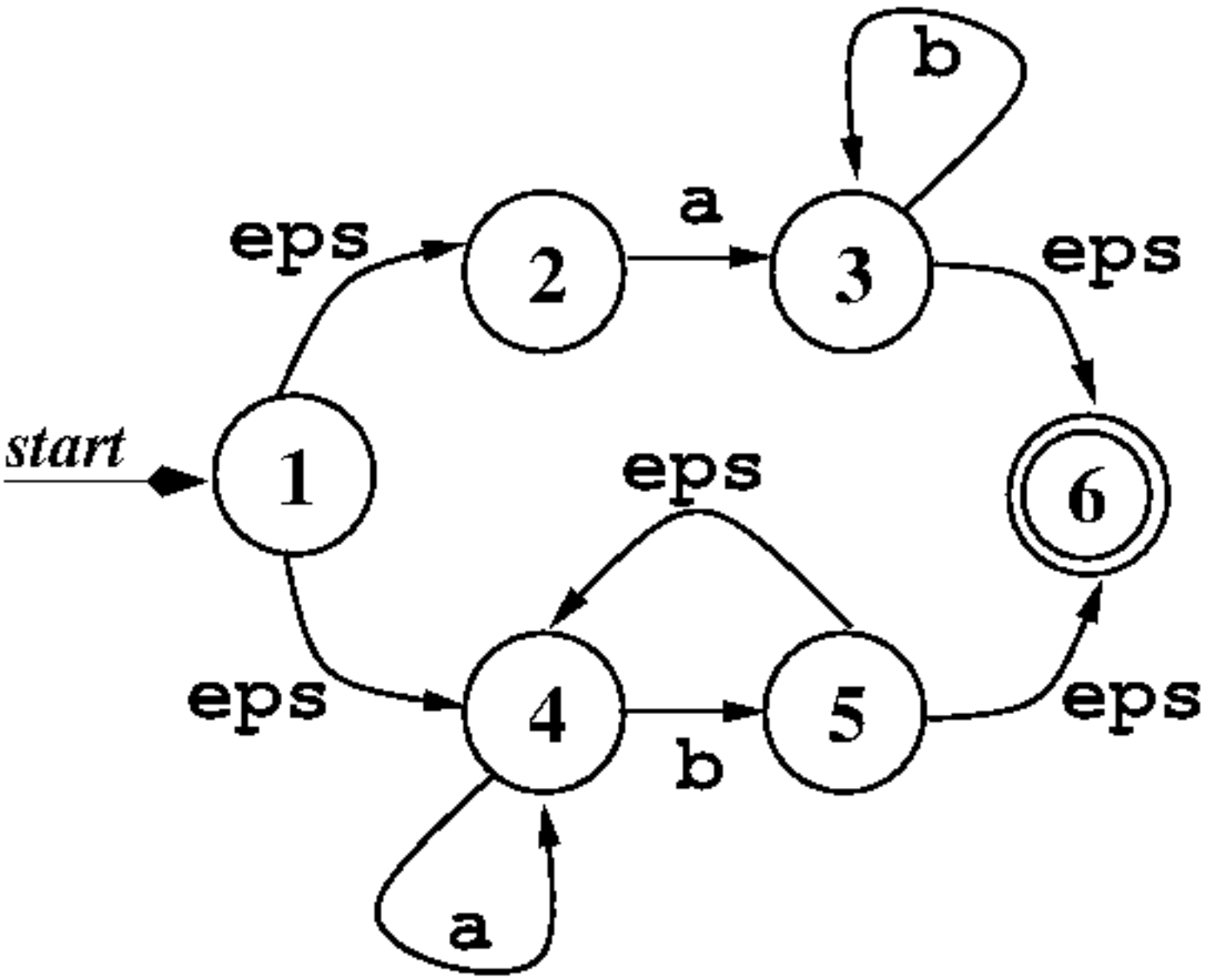
Compilers Assignment I

Hand in your solution in the form of a short report in text or PDF format.

Task 1 (NFA to DFA translation)

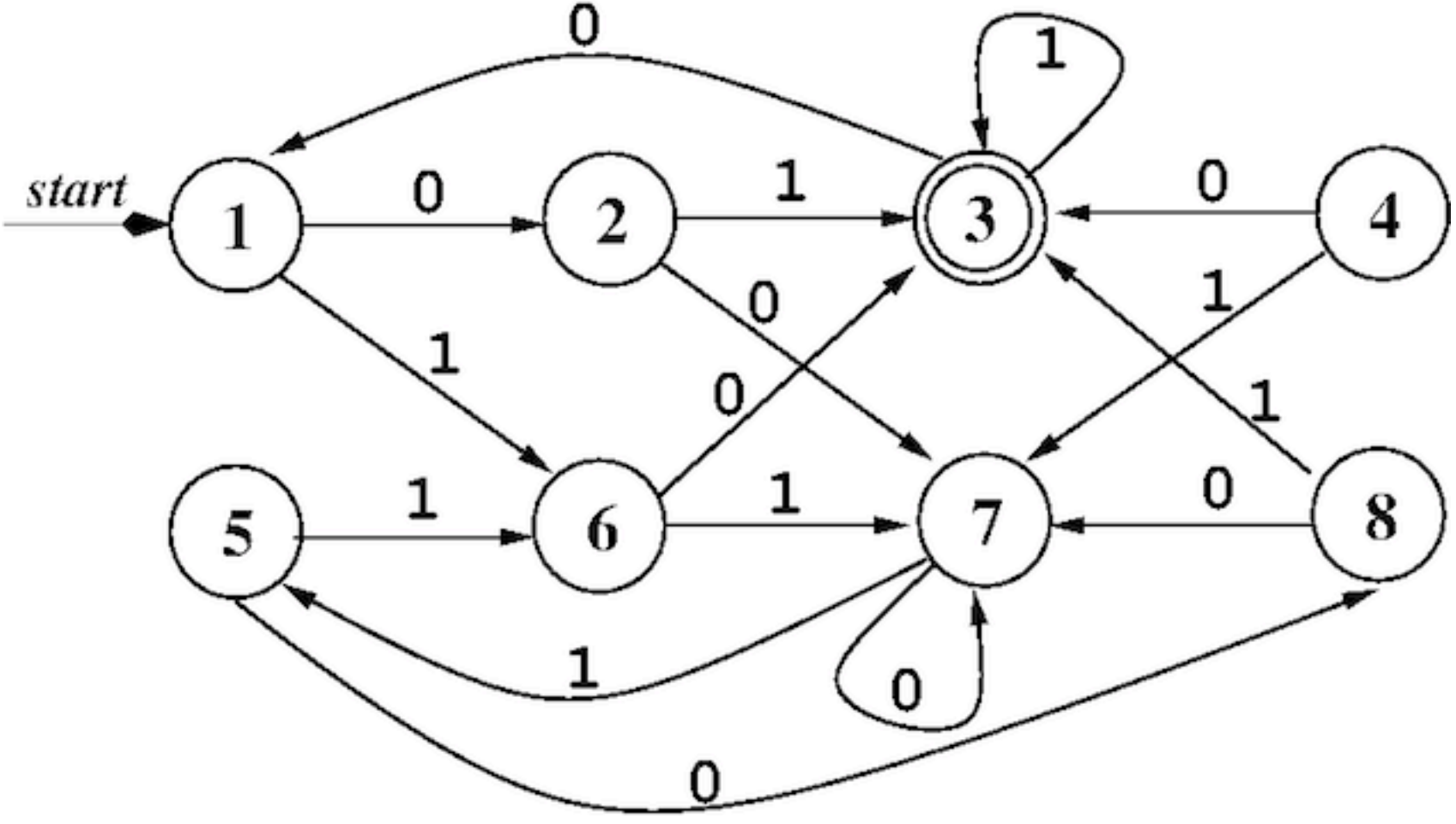
Convert the non-deterministic finite automaton (NFA) below into a deterministic finite automaton (DFA) using the subset-construction algorithm:



- Derive the move function for all (about `10`) pairs of DFA state and input character.
- Then draw the resulting DFA on the right-hand side of the original NFA. (In the Figure, eps stands for ε.)

Task 2 (DFA minimization)

Minimize the DFA given below, following the algorithm presented in the lecture.



Task 3 (regular languages and tokenization)

- a.) Using the alphabet of decimal digits, give regular expression describing the following languages:
  - (i) Numbers divisible by `5`.
  - (ii) Numbers in which digit `5` occurs exactly three times.
- b.) Are the following languages over the alphabet of decimal digits regular? Give short convincing reasons for your answers:
  - (i) Numbers of arbitrary length which contain digit `1` exactly as many times as digit `2`.
  - (ii) Numbers N < 1.000.000 which contain digit `1` exactly as often as digit `2`.

Task 4

This task refers to how to disambiguate a grammar. See class lecture "Syntax Analysis" ~slides 15-20, or Section 2.3 of "Introduction to Compiler Design" book.

The following ambiguous grammar describes Boolean expressions:

```
B -> B && B
B -> B || B
B -> (B)
B -> true
B -> false
```

Rewrite the grammar to be unambiguous, assuming that:

- `&&` and `||` are single-terminal symbols and that
- `&&` (conjunction) binds tighter than `||` (disjunction), and that
- `&&` is right associative and `||` is left associative,

Task 5

This task refers to LL(1) parser construction, including the construction of Nullable, First, Follow sets and eliminating left recursion and identical prefixes. See class lecture "Syntax Analysis" slides ~ 22-37, or Section 2.8 to 2.12 of "Introduction to Compiler Design" book.

Consider the following grammar for postfix expressions:

```
E -> E E +
E -> E E *
E -> num
```

- (a) Eliminate Left-Recursion in the grammar
- (b) Perform Left Factorization of the grammar obtained in question (a)
- (c) On the grammar obtained in question (b) compute the Nullable and First sets for every production and the Follow sets for every nonterminal
- (d) Compute the Look-Ahead sets for every non-terminal, Make a LL(1) parse table, and Implement a Recursive-Descent Parser (pseudocode). (See Slides~33-36 in class lecture "Syntax Analysis" or Section 2.11 in book)

Task 6

See class lecture "Syntax Analysis" slides~38-end, or Section 2.13 to 2.15 of "Introduction to Compiler Design" book.

Consider the grammar below, where `=>` is considered a single terminal symbol:

```
T   ->   T => T
T   ->   T * T
T   ->   int
```

- (a) Add a new start production and compute Follow(T) (and remember to add yet another start production to account for the end of file token `$`).
- (b) Construct an SLR parser table for the grammar.
- (c) Eliminate conflicts using the following precedence rules:
  - (i) Operator `*` binds tighter than `=>`
  - (ii) Operator `*` is left associative, and `=>` is right associative.

Last updated 2016-04-11 22:31:09 CEST