Universitetet i Oslo
Institutt for Informatikk
Reliable Systems
Martin Steffen

INF 5110: Compiler construction

Spring 2024 **Series 1** 22. 1. 2024

Topic: Scanning: automata and regular expressions

Issued: 22. 1. 2024

The exercises on this sheet refer to the lecture and material about *scanning* (or lexing). In general, when talking about letters, we assume for instance an English lettered alphabet, i.e., when speaking about "letters", we mean those letter characters as covered by the simple ASCII character set. Remember also: the ASCII character set is, technically, an *ordered* alphabet. Here, in this (and similar) exercises, we mostly restrict ourselves to alphabets of 2 or 3 different letters, only (like a, b, c).

Similarly, when talking about digits, we naturally refer to the numeric symbols of the base-10 numerical notation.

Exercise 1 (Regular expressions) Use *regular expressions* to capture the languages described informally as follows:

1. All strings of lowercase letters that begin and end with an a.
2. All strings of lowercase letters that begin or end with an a.
3. All strings of digits that do not contain leading zeroes.
4. All strings of digits that represents *even* numbers.
5. All strings of digits such that all the 2s precede all the 9s.
6. All strings of a's and b's (i.e., $\Sigma = \{a, b\}$) that don't contain 3 consecutive b's.
7. All strings of a's and b's that contain an odd number of a's *or* an odd number of b's.
8. All strings of a's and b's that contain an even number of a's *and* an even number of b's.
9. All strings of a's and b's that contain exactly as many a's than b's.

Exercise 2 (From regular expessions to automata) Assume the following regular expression

$$(\boldsymbol{a} \mid \boldsymbol{b})^* \boldsymbol{a} (\boldsymbol{a} \mid \boldsymbol{b} \mid \boldsymbol{\epsilon}) \tag{1}$$

Formulate a sentence (in English/Norwegian etc.) which captures the meaning of the regular expression. Now, turn the regular expression into the (deterministic, minimal) finite-state automaton which recognises that language. To do so, follow the 3 standard steps:

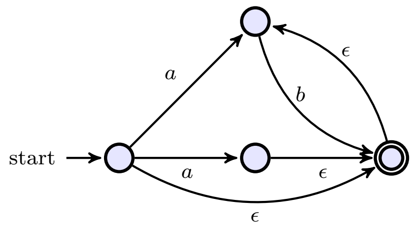
1. use *Thompson's construction* to obtain an non-deterministic finite-state automaton for the regular expression (1).

2. Use the subset construction to turn that into an equivalent DFA ("determinization").
3. Use the *partition refinement* algo to obtain the/a minimal equivalent DFA ("minimization")

Exercise 3 Find a few "tools", or libraries for certain languages etc. that use regular expressions. Check the "syntax" of them.

Exercise 4 (Reflection) The lecture and exercises teaches the standard way to turn regular expressions into deterministic and minimal finite-state automata, which goes through 3 separate steps. A lexer (as first stage of most compilers) is typically interested only in the end-result, a minimal, deterministic FSA. Why do we bother (and why do many lexer/scanner implementations bother) to go through these 3 different stages, why not go directly from regular expressions to the end result?

Exercise 5 (Determinization) Determinize the automaton visualized in the following graphics. Use for that the powerset construction.



□

Exercise 6 (Minimization) Minimize the automaton vizualized in the following figure.