Software-Modellierung und Verifikation
Informatik 2
Prof. J.-P. Katoen
RWTH Aachen

Priv.-Doz. T. Noll noll@cs.rwth-aachen.de
Ch. Jansen christina.jansen@cs.rwth-aachen.de

1. Exercise sheet *Compiler Construction 2010*

Due Wed., 27 October 2010, *before* the exercise course begins.

Exercise 1.1: **(4 points)**

Assume we want to analyse the two program fragments given below. List reasonable symbol classes and corresponding tokens useful for the lexical analysis. Think in terms of a scanner and decompose the fragments according to your categorisation.

(a)
```
float limitedSquare(x) float x;{
  return (x<=-10.0||x>=10.0)?100:x*x;
}
```

(b)
```
<div id="logo">
<a href="./"><img src="./logos/openlogo-nd-50.png" width="50" alt=""></a>
</div>
```

Exercise 1.2: **(2+1+2+4 points)**

Considering todays situation, you could ask: Why worrying about compiler construction? There exist already many and highly optimised compilers, even compiler compilers. But indeed new compilers are needed in everydays situation. Assume you might want to write your very own command line tool, which should accept – and maybe later on analyse – logic formulae as inputs (specified in some kind of "logic programming language"). Here of course you need your own compiler! And designing it is what we want to do...

To make life easy we restrict ourselves to propositional logic, where we could have propositions of the form

$$\Phi :== tt \,|\, ff \,|\; \text{words over the alphabet } \Sigma = \{a, \ldots, z\}$$

Formulae are then inductively defined by: If Φ, Ψ propositional formulae, then so are

$$(\neg\Phi), (\Phi \wedge \Psi), (\Phi \vee \Psi), (\Phi \rightarrow \Psi)$$

For now we only want to allow propositions starting with s or t and ending on *ing*.

a) Give regular expressions and languages for all symbol classes that will be needed for a lexical analysis of elements of $\mathcal{L}_{propLogic}$. Denote the languages of each symbol class by the associated token, e.g. for tokens and, or, proposition, ... write $\mathcal{L}_{\text{and}}, \mathcal{L}_{\text{or}}, \mathcal{L}_{\text{proposition}}, \ldots$.

b) Give the sequence of lexemes for $(\neg(\mathit{talking} \wedge (tt \rightarrow \mathit{singing})))$ and translate each lexeme into a symbol.

c) Derive a NFA that accepts $w \in \mathcal{L}_{\text{tok}}$ in state q_{tok} for all tok $\in \{\text{and}, \text{or}, \text{proposition}, \ldots\}$.

d) Solve the simple matching problem for the input string *singing* by using the algorithm given in the lecture. Apply the DFA- as well as the NFA-method.