Exercise 2: Automata Theory

Formal Methods II, Fall Semester 2013

Distributed: 11.10.2013
Due Date: 25.10.2013
Send your solutions to: tobias.klauser@uzh.ch or deliver them in the class.

Finite State Automata – Theoretical Exercises

1. (2 points) What kind of strings are accepted by the following Deterministic Finite Automata (DFA)? Give a short verbal description of the language defined by each DFA:

(a)

	0	1
$\rightarrow q_0$	q_0	q_1
$*q_1$	q_0	q_0

(b)

	a	b	c
$\rightarrow q_0$	q_0	q_1	q_0
q_1	q_2	q_1	q_0
$*q_2$	q_2	q_2	q_2

2. (2 points) Draw a state diagram for a DFA that defines each of the following languages:
   (a) All strings over {a,b} that contain an even number of a's
   (b) All strings over {a,b} that contain an even number of a's and an odd number of b's

Bonus (3 points) Give a DFA that accepts all strings over {0,1} that represent binary numbers that are divisible by 5. For example, 0, 101, 1010, 1111, 11001 are all in the language, while 1, 10, 11, 110, 1011, 11000 are all not in the language.

Hint: Consider creating a machine with 5 states. Think about having your machine keep track of (number so far) mod 5.

3. (6 points) Give a Non-deterministic Finite Automaton (NFA), with a minimal number of states and transitions, for each of the following languages:
   (a) All strings over {a,b} that have an a as one of the last 3 characters in the string. For example, a, baab, bbbab, aabbaabb are all in the language, while bb, babbb, bbabbbbb are all not in the language.
   (b) All strings over {E, G, L, O, X} that *contain* the regular expression GO*GLE. For example, GGLE, GOOGLE, XXGOGOOOGLEGLXX are all in the language, while GLE, GOOGLLE are all not in the language.

Give the simplest possible DFA that corresponds to the NFA of point (b).

Parsers – Practical Exercise

4. The goal of this practical exercise is to familiarize yourself with parsers. In this exercise, you'll write a simple parser, which evaluates simple mathematical expressions such as:

$$(2+3) \times 4$$

Don't worry if this is the first time you're exposed to programming: you'll only have to understand and slightly *modify* existing code! Moreover, the assistants are eager to help you if you have any question or problem!

The programming language used for this exercise is called Python. It comes with a very simple and intuitive syntax, and therefore does not require a lot of prior programming experience to understand! The appendix provides a crash course in the Python programming language.

(a) The calculator code file (`calc.py`) can be found inside "`Exercise2_material.zip`", which is available on the class website. This initial calculator only recognizes the sum operator "$+$" and the subtraction operator "$-$". In other words, it only implements the following grammar:

$$\begin{aligned} <\text{command}> &= <\text{expression}> ; \\ <\text{expression}> &= <\text{factor}> \\ & \quad \mid <\text{expression}> \text{`}+\text{'} <\text{factor}> \\ & \quad \mid <\text{expression}> \text{`}-\text{'} <\text{factor}> ; \\ <\text{factor}> &= <\text{number}> ; \\ <\text{number}> &= (\text{`0'} \mid \text{`1'} \mid \text{`2'} \mid \text{`3'} \mid \ldots \mid \text{`9'})^{+} \; ; \end{aligned}$$

(b) Run the program, and try typing a few expressions, such as "$2+3-1$". Then look at the code and try to understand its general structure. Pay particular attention to how the functions `scanCommand()`, `scanExpression()` and `scanFactor()` work. Concerning the other functions, you just have to understand *what* they do (i.e. what is described in the comments), not necessarily *how* they do it.

(c) (5 points) Modify the program so that the calculator also accepts the multiplication sign "$\times$" (i.e. the letter "x"), as well as the parentheses "(" and ")". In other words, modify the code so that it implements the following grammar:

$$
\begin{aligned}
<\text{command}> &= <\text{expression}> \\
<\text{expression}> &= <\text{term}> \\
&\quad |\ <\text{expression}>\ \text{'+'}\ <\text{term}> \\
&\quad |\ <\text{expression}>\ \text{'}-\text{'}\ <\text{term}>\ ; \\
<\text{term}> &= <\text{factor}> \\
&\quad |\ <\text{term}>\ \text{'}\times\text{'}\ <\text{factor}>\ ; \\
<\text{factor}> &= <\text{number}>\ |\ \text{'('}\ <\text{expression}>\ \text{')'} \\
<\text{number}> &= (\ \text{'0'}\ |\ \text{'1'}\ |\ \text{'2'}\ |\ \text{'3'}\ |\ \ldots\ |\ \text{'9'}\ )^{+}
\end{aligned}
$$

To test your program, type in a few expressions such as "$2 - 3 \times (4 + 5)$" and check the results!

(d) (5 points) Modify the program so that the calculator now also accepts the usage of the variables "a", "b" and "c". In other words, modify the code so that it implements the following grammar:

$$
\begin{aligned}
<\text{command}> &= <\text{assignment}>\ |\ <\text{expression}> \\
<\text{assignment}> &= <\text{variable}>\ \text{'='}\ <\text{expression}> \\
<\text{expression}> &= <\text{term}> \\
&\quad |\ <\text{expression}>\ \text{'+'}\ <\text{term}> \\
&\quad |\ <\text{expression}>\ \text{'}-\text{'}\ <\text{term}>\ ; \\
<\text{term}> &= <\text{factor}> \\
&\quad |\ <\text{term}>\ \text{'}\times\text{'}\ <\text{factor}>\ ; \\
<\text{factor}> &= <\text{number}>\ |\ \text{'('}\ <\text{expression}>\ \text{')'}\ |\ <\text{variable}> \\
<\text{number}> &= (\ \text{'0'}\ |\ \text{'1'}\ |\ \text{'2'}\ |\ \text{'3'}\ |\ \ldots\ |\ \text{'9'}\ )^{+} \\
<\text{variable}> &= \text{'a'}\ |\ \text{'b'}\ |\ \text{'c'}
\end{aligned}
$$

To test your program, type in a few expressions such as "a $= 2 + 3 \times 4$" and then "a $\times$ 2", and check the results!

Bonus (2 points) Did you notice that the calculator also accepts malformed expressions such as "2 (", or does not complain if you use unknown symbols such as "/"? Find an elegant solution to fix this bug!

Have fun!

Appendix: Crash Course in Python

Installing Python

- Python is already included in standard distributions of UNIX operating systems, such as Linux or Mac OS X. If you're using such OS, you don't need to install anything.
- If you're using Windows, download and install Python from:

  `http://www.python.org/download/`

  Choose the link "Python 3.3.2 Windows installer" and follow the standard installation process.

Downloading tutorial code

You can find four tutorial code files inside "`Exercise2_material.zip`".

Understanding Python code

The four tutorials provide simple self-explanatory examples with increasing complexity. You may want to first execute a tutorial to see what it does, before reading the code and trying to understand it.

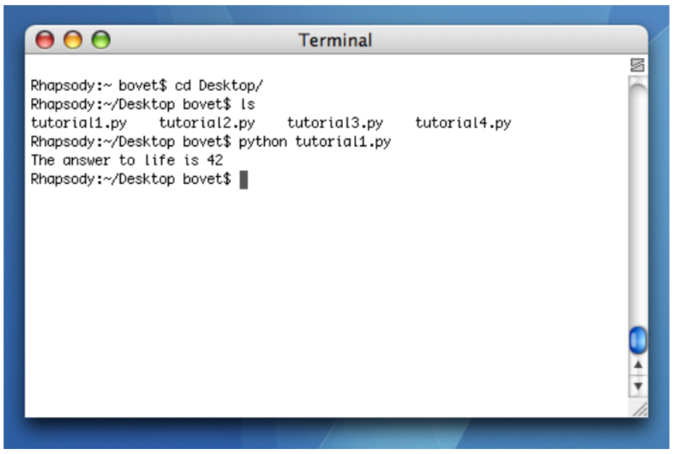
Comments are indicated in Python with "#". The tutorial code files contain many of them: they should help you to quickly understand the syntax of Python!

Running and modifying a Python program

- On Linux/Mac OS X:

  1. Open a Terminal: on Linux, click the Terminal icon in the dock; on Mac OS X, the Terminal is found in /Applications/Utilities/. The Terminal icon looks like this:

  2. Make sure that the current directory contains the Python program file that you want to execute. To change the directory, use the command `cd`. To list the directory contents, use the command `ls`. To run the first Python tutorial program, enter the command `python tutorial1.py`:



```
Rhapsody:~ bovet$ cd Desktop/
Rhapsody:~/Desktop bovet$ ls
tutorial1.py    tutorial2.py    tutorial3.py    tutorial4.py
Rhapsody:~/Desktop bovet$ python tutorial1.py
The answer to life is 42
Rhapsody:~/Desktop bovet$ 
```

  3. To modify a program, just open the corresponding .py file with your favorite editor. Make sure you save the modifications before running the program again!
  4. To abort a running program, press Control-C.

- On Windows:

  1. To start Python development environment, select Start > Programs > Python 3.3 > IDLE (Python GUI).
  2. To open a program file (e.g. tutorial1.py), select File > Open...
  3. To run the program, select Run > Run Module (F5).
  4. To abort a running program, press Control-C.

What’s special about Python

If you have prior experience with any other programming language, you might notice the following major syntactic features of Python:

- It’s *really* simple! For instance, you don’t have to declare variables.
- Indentation plays an important role! There is for example no “end” after an if-clause or after a while-loop.
- Don’t forget the colon “:” after if- or while-conditions. On the other hand, you don’t have to put semi-colons “;” after any statement. See for example:

```
if x > 0:
    s = "is greater than zero"
elif x < 0:
    s = "is less than zero"
else:
    s = "equals zero"
print x, s
```

- Functions are defined with the keyword `def`. They can return more than one value.
- Finally, the main part of the program has to be placed after the function declarations, i.e. at the end of the file:

```
def firstPerfectNumbers():
    return 6, 28

x, y = firstPerfectNumbers()
print "The two first perfect numbers are", x, "and", y
```

More information about Python

To learn more about Python, you may start by looking at the following tutorial:

`http://docs.python.org/tutorial/`