TDT4205 Compilers

Exercise 1

Stian Hvatum (hvatum)
MTDT

January 30, 2012

Task 1

Version info of *gcc*, *flex* and *bison*

- gcc (GCC) 4.2.3 (Ubuntu 4.2.3-2ubuntu7)
- flex 2.5.34
- bison (GNU Bison) 2.3

Task 2

A lexical analyzer is a program that accepts the source code of a program as a stream of characters, and identifying them as lexemes and outputs this as tokens.

An acceptor is a program or machine that inputs something, like the token stream generated by the lexical analyzer, and desides if it satisfies a given syntax, like a programming language. If the input is accepted, the acceptor returns *true*, else it returns *false*.

Task 3

3.1

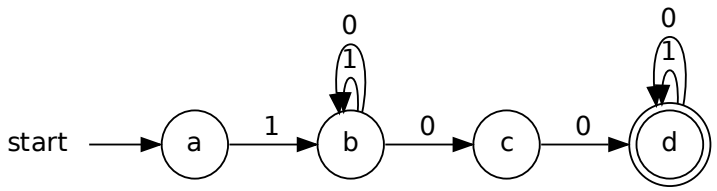

Figure 1: NDF for 1(0—1)*00(0—1)*

3.2

This language includes:

- The word 100
- All words starting with 100
- All words starting with 1 and ending with 00
- All words starting with 1 and containing 00

given the alphabeth is the set of 0 1
Examples are

- 11111111111111111001111111111111111111
- 1001111111111111111111
- 10011001100110011001
- 10001

3.3

	a	b	c	d
1	b	b	-	d
0	-	{b c}	d	d

Figure 2: Transition table for 1(0—1)*00(0—1)*

Task 4

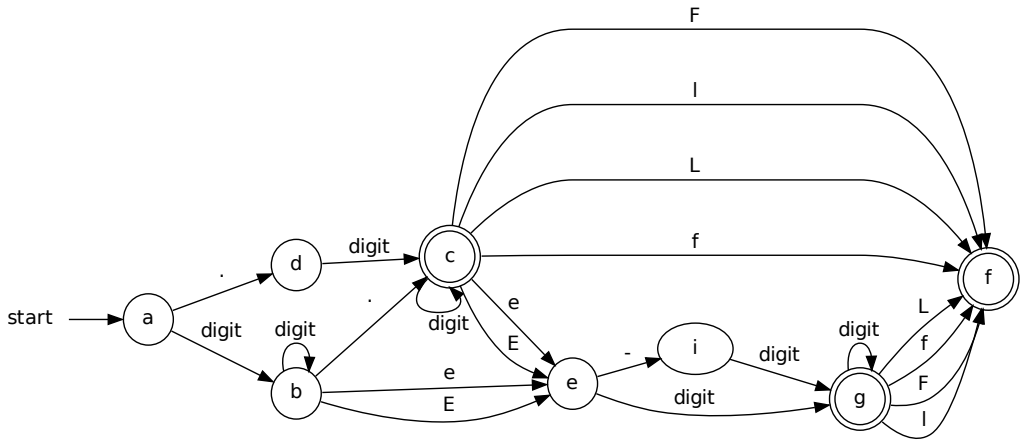

Figure 3: DFA for float

Appendix

DOT-code for NDFA 3.1

```
1  //NFA for lang 1(0|1)*00(0|1)*
2  digraph G {
3     rankdir=LR;
4     shape=circle;
5     start->a
6     a->b [label="1"];
7     b->b [label="1" dir=back];
8     b->b [label="0" dir=back];
9     b->c [label="0"];
10    c->d [label="0"];
11    d->d [label="1" dir=back];
12    d->d [label="0" dir=back];
13
14    a[shape=circle];
15    b[shape=circle];
16    c[shape=circle];
17    d[shape=doublecircle];
18    start[shape=plaintext];
19 }
```

DOT-code for NDFA 4

```
//NFA for lang 1(0|1)*00(0|1)*
digraph G {
    rankdir=LR;
   shape=circle;
   start->a
   a ->b  [label="digit"];
   a ->d  [label="."];
   d ->c  [label="digit"];
   b->b [label="digit"];
   b->c [label="."];
   b->e [label="e"];
   b->e [label="E"];
   c->c [tailport=sw headport=s label="digit"];
   c->e [label="e"];
   c->e [label="E"];
   e->g [label="digit"];
   e->i [label="-"];
   i->g [label="digit"]
   g->g [label="digit"]
   g->f [label="f"];
   g->f [label="F"];
   g->f [label="l"];
   g->f [label="L"];
   c->f [label="f"];
   c->f [label="F"];
   c->f [label="l"];
   c->f [label="L"];


   a[shape=circle];
   b[shape=circle];
   c[shape=doublecircle];
   d[shape=circle];
   e[shape=circle];
   f[shape=doublecircle];
   g[shape=doublecircle];
   start[shape=plaintext];
}
```