1. Objektorientierte Implementierung endlicher Automaten

a) faOf()

Lösungsidee:

Beim Durchdenken des Algorithmus habe ich festgestellt, dass Epsilon-Alternativen in der reguläre Grammatik die Umsetzung erschweren könnten. Deshalb wird zunächst die reguläre Grammatik mit der Methode **newEpsilonFreeGrammarOf** aus dem ersten Übungszettel Epsilon-frei gemacht. Man kann dann die reguläre Grammatik relativ einfach in einen endlichen Automaten umwandeln, indem man jede Alternative einer Regel (z.B. **S -> a B**) in einen Zustandsübergang vom NTSymbol der Regel (**S**) auf den Zustand (NTSymbol an zweiter Stelle in der Alternative **B**) mit dem Band-Symbol (TSymbol an erster Stelle in der Alternative **a**) umwandelt. Alle Alternativen, die nur aus einem TSymbol bestehen, bekommen einen Zustandsübergang auf einen allgemeinen End-Zustand (**END**). Da die reguläre Grammatik Epsilon-frei ist, gibt es höchstens ein Epsilon, welches sich im Satz-Symbol befinden könnte. In diesem Fall wird der Zustand für das Satz-Symbol auch als End-Zustand markiert.

Code:

Tests:

b) grammarOf()

Lösungsidee:

Es kann ein beliebiger **FA** übergeben werden. Damit ich DFA und NFA bei der Implemtierung des Algorithmus gleich behandeln kann, habe ich die Sichtbarkeit der Methode **virtual StateSet FA::deltaAt(const State &src, TapeSymbol tSy) const = 0;** auf public geändert und immer mit **StateSet**s gearbeitet.

Es werden alle Kombinationen von Zuständen und Band-Symbolen auf die Zustandsüberführungsfunktion angewandt (verschachtelte Schleife). Um nicht zwischen DFA und NFA unterscheiden zu müssen, wurde die Sichtbarkeit von der Methode **deltaOf() der Klasse FA** von **protected** auf **public** geändert und immer mit **StateSet**s gearbeitet. Bei jeder Anwendung der Zustandsüberführungsfunktion ist der **State** (Variable in der Schleife) die Regel der Grammatik. Das Band-Symbol (Variable in der Schleife => TSymbol) und die von **deltaOf()** gelieferten Zustände (NTSymbol) bilden Sequenzen dieser Regel. Falls die **deltaOf()** von gelieferten Zustände keine ausgehenden Zustandsüberführungen mehr haben, dann werden diese in der Grammatik einfach ignoriert und das Band-Symbol hat in der Alternative kein nachfolgendes NTSymbol. Falls der Start-Zustand bereits ein End-Zustand ist, bekommt die Regel dieses Zuständes eine Epsilon-Alternative.

Code:

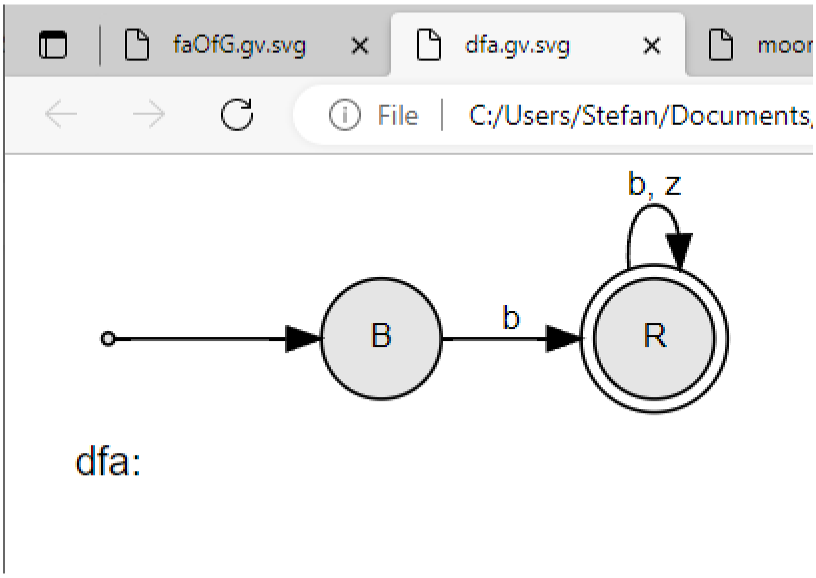
Tests:

2. DFA, Erkennung und Mealy- oder Moore-Automat

a)

Code:

```
cout << "2.a) DFA" << endl;
cout << "------" << endl;
cout << endl;

fab = new FABuilder();
fab->setStartState("B").
  addFinalState("R").
  addTransition("B", 'b', "R").
  addTransition("R", 'b', "R").
  addTransition("R", 'z', "R");

dfa = fab->buildDFA();
vizualizeFA("dfa", dfa);

cout << "dfa->accepts(\"bzb\") = " << boolalpha << dfa->accepts("bzb") << endl;
cout << "dfa->accepts(\"bbbbzbzz\") = " << boolalpha << dfa->accepts("bbbbzbzz")
<< endl;
cout << "dfa->accepts(\"zbb\")   = " << boolalpha << dfa->accepts("z")  << endl;
cout << endl;

delete fab;
```



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
VT  = { b, z }

2.a) DFA
------

writing    dfa to dfa.gv ...
rendering  dfa.gv to dfa.gv.svg ...
displaying dfa.gv.svg ...

dfa->accepts("bzb")      = true
dfa->accepts("bbbbzbzz") = true
dfa->accepts("zbb")      = false

2.b) MooreDFA
```

Tests:

b)

Eigentlich benötigt der MooreDFA nur eine Map, in der definiert ist, bei welchem State durch welches Symbol ausgegeben wird. Diese Map muss dann beim Erkennen des Band-Inhaltes verwendet werden (hier geben wir den Wert für den State einfach auf der Konsole aus).

Code:

```
// MooreDFA.h:                                                   SWE, 2022
// -----
// Objects of class MooreDFA represent deterministic finite automata.
//=====================================================================

#ifndef MooreDFA_h
#define MooreDFA_h

#include "ObjectCounter.h"
#include "TapeStuff.h"
#include "StateStuff.h"
#include "DFA.h"


class FABuilder;

class MooreDFA: public DFA
 /*OC+*/ , private ObjectCounter<MooreDFA> /*+OC*/ {

  friend class FABuilder;
```

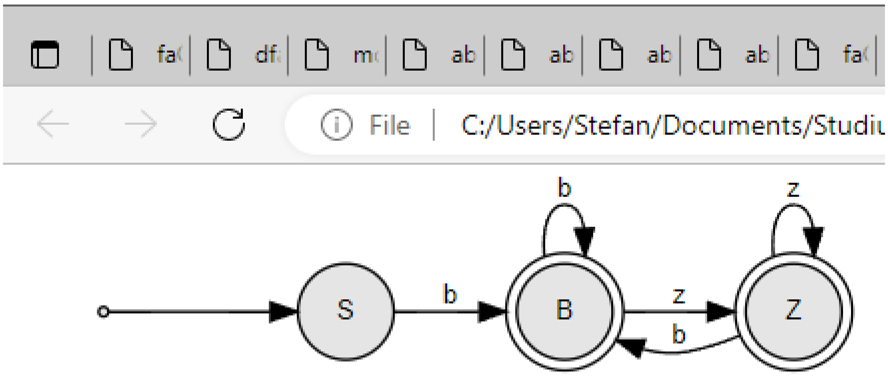
```
  private:
    typedef DFA Base;

  protected: // allows derived classes, e.g., for Mealy and or Moore
    // constructor called by FABuilder::build... methods and derived classes
    MooreDFA(const StateSet &S,  const TapeSymbolSet &V,
        const State    &s1, const StateSet      &F,
        const DDelta   &delta,
        const std::map<State, char> &mooreLambda);

  public:
    const std::map<State, char> mooreLambda;
    MooreDFA(const MooreDFA  &mooredfa) = default;
    MooreDFA(      MooreDFA &&mooredfa) = default;

    virtual ~MooreDFA() = default;
    virtual bool accepts(const Tape &tape) const;

}; // DFA

#endif

// end of MooreDFA.h
//======================================================================

// MooreDFA.cpp:                                              SWE, 2022
// -------
// Objects of class MooreDFA represent deterministic finite automata.
//======================================================================

#include <cmath>
#include <cstring>

#include <iostream>
#include <fstream>
#include <map>
#include <sstream>

using namespace std;

#include "TapeStuff.h"
#include "StateStuff.h"
#include "MbMatrix.h"
#include "FABuilder.h"
#include "MooreDFA.h"

// --- implementation of class MooreDFA ---

MooreDFA::MooreDFA(const StateSet &S,  const TapeSymbolSet &V,
         const State    &s1, const StateSet      &F,
         const DDelta   &delta,
         const std::map<State, char> &mooreLambda)
: DFA(S, V, s1, F, delta), mooreLambda(mooreLambda) {
} // MooreDFA::MooreDFA
```

```
bool MooreDFA::accepts(const Tape &tape) const {
  int       i   = 0;       // index of first symbol
  TapeSymbol tSy = tape[i]; // fetch first tape symbol
  State     s   = s1;      // start state
  cout << mooreLambda.at(s);
  while (tSy != eot) {      // eot = end of tape
    s = delta[s][tSy];
    if (!defined(s))
      return false;         // s undefined, so no acceptance
    cout << mooreLambda.at(s);
    i++;
    tSy = tape[i];          // fetch next symbol
  } // while
  cout << " ";
  return F.contains(s);     // accepted <==> s element of F
} // MooreDFA::accepts

// end of MooreDFA.cpp
//======================================================================
```

Tests:

```
  cout << "2.b) MooreDFA" << endl;
  cout << "------" << endl;
  cout << endl;

  fab = new FABuilder();
  fab->setStartState("S").
    addFinalState("B").
    addFinalState("Z").
    addTransition("S", 'b', "B").
    addTransition("B", 'b', "B").
    addTransition("B", 'z', "Z").
    addTransition("Z", 'z', "Z").
    addTransition("Z", 'b', "B").
    setSetMooreLambda({
      {"S", ' '},
      {"B", 'c'},
      {"Z", 'd'}
    });

  MooreDFA* mooreDfa = fab->buildMooreDFA();

  cout << "mooreDfa->accepts(\"bzb\") = " << boolalpha << mooreDfa->accepts("bzb")
<< endl;
  cout << "mooreDfa->accepts(\"bbbbzbzz\") = " << boolalpha << mooreDfa-
>accepts("bbbbzbzz") << endl;
  cout << "mooreDfa->accepts(\"zbb\")   = " << boolalpha << mooreDfa->accepts("z")
<< endl;
  cout << endl;
```

```
vizualizeFA("mooreDfa", mooreDfa);

delete mooreDfa;
delete fab;
```



mooreDfa:

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

```
2.b) MooreDFA
------

mooreDfa->accepts("bzb")      =  cdc true
mooreDfa->accepts("bbbbzbzz") =  ccccdcdd true
mooreDfa->accepts("zbb")      =  false

writing    mooreDfa to mooreDfa.gv ...
rendering  mooreDfa.gv to mooreDfa.gv.svg ...
displaying mooreDfa.gv.svg ...
```

3. NFA, Transformation NFA -> DFA und Zustandsminimierung

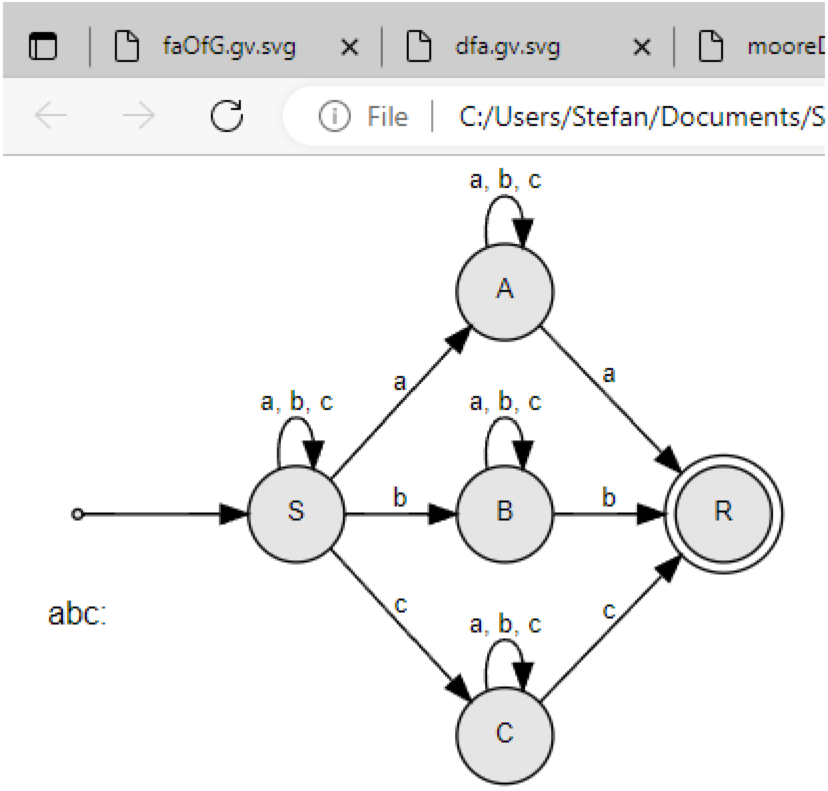
a)

Code:

```
cout << "3.a)" << endl;
cout << "------" << endl;
```

```
    cout << endl;

    fab = new FABuilder();
    fab->setStartState("S").
      addFinalState("R").
      addTransition("S", 'a', "S").
      addTransition("S", 'b', "S").
      addTransition("S", 'c', "S").
      addTransition("S", 'a', "A").
      addTransition("S", 'b', "B").
      addTransition("S", 'c', "C").
      addTransition("A", 'a', "A").
      addTransition("A", 'b', "A").
      addTransition("A", 'c', "A").
      addTransition("B", 'a', "B").
      addTransition("B", 'b', "B").
      addTransition("B", 'c', "B").
      addTransition("C", 'a', "C").
      addTransition("C", 'b', "C").
      addTransition("C", 'c', "C").
      addTransition("A", 'a', "R").
      addTransition("B", 'b', "R").
      addTransition("C", 'c', "R");

    NFA* abc = fab->buildNFA();
    cout << "abc->accepts1(\"cabcabcabcabcc\") = " << abc-
  >accepts1("cabcabcabcabcc") << endl;
    cout << "abc->accepts2(\"cabcabcabcabcc\") = " << abc-
  >accepts2("cabcabcabcabcc") << endl;
    cout << "abc->accepts3(\"cabcabcabcabcc\") = " << abc-
  >accepts3("cabcabcabcabcc") << endl;

    cout << "abc->accepts1(\"caaaaaaaaaaaaaaad\") = " << abc-
  >accepts1("caaaaaaaaaaaaaaad") << endl;
    cout << "abc->accepts2(\"caaaaaaaaaaaaaaad\") = " << abc-
  >accepts2("caaaaaaaaaaaaaaad") << endl;
    cout << "abc->accepts3(\"caaaaaaaaaaaaaaad\") = " << abc-
  >accepts3("caaaaaaaaaaaaaaad") << endl;

    delete abc;
    delete fab;
```

Tests:




```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

3.a)
------

abc->accepts1("cabcabcabcabcc") = true
abc->accepts2("cabcabcabcabcc") = true
abc->accepts3("cabcabcabcabcc") = true
abc->accepts1("caaaaaaaaaaaaaad") = false
abc->accepts2("caaaaaaaaaaaaaad") = false
abc->accepts3("caaaaaaaaaaaaaad") = false
3.b)
------
```

b)

Code:

```
void TimeAccept(NFA* abc, void (*func)(NFA* abc)) {
  stopwatch::Stopwatch sw{}; // not going to post this entire class here
  sw.start();
  func(abc);
  cout << "Elapsed time: " << sw.elapsed<TimeFormat::MICROSECONDS>() << " micro
```

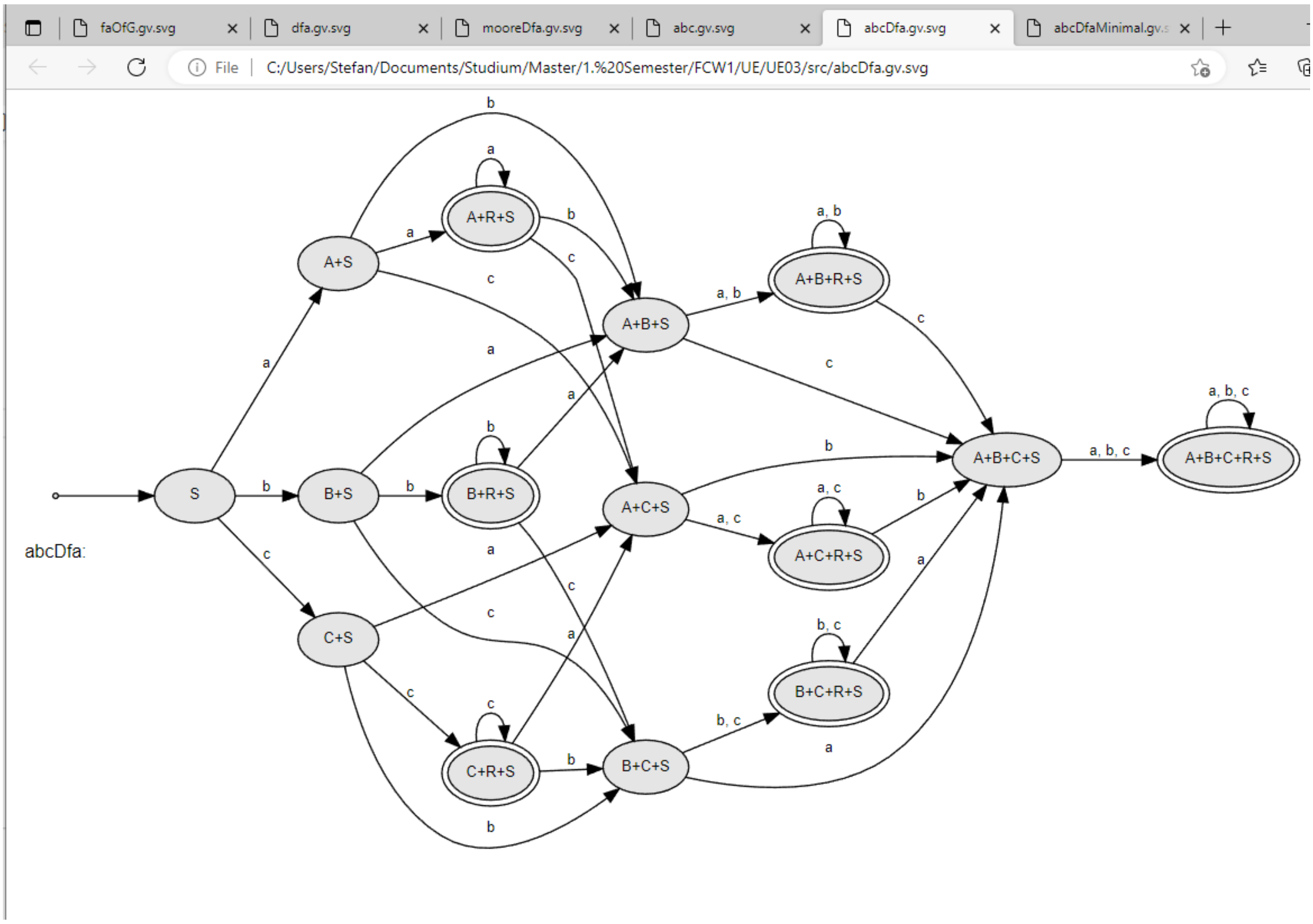
```
sec" << endl;
  }

// ... same as 3a)

  TimeAccept(abc, [](NFA* abc) {
    cout << "abc->accepts1(\"cabcabcabcabcc\") = " << abc-
>accepts1("cabcabcabcabcc") << endl;
  });
  TimeAccept(abc, [](NFA* abc) {
    cout << "abc->accepts2(\"cabcabcabcabcc\") = " << abc-
>accepts2("cabcabcabcabcc") << endl;
  });
  TimeAccept(abc, [](NFA* abc) {
    cout << "abc->accepts3(\"cabcabcabcabcc\") = " << abc-
>accepts3("cabcabcabcabcc") << endl;
  });

  TimeAccept(abc, [](NFA* abc) {
    cout << "abc->accepts1(\"caaaaaaaaaaaaaaad\") = " << abc-
>accepts1("caaaaaaaaaaaaaaad") << endl;
  });
  TimeAccept(abc, [](NFA* abc) {
    cout << "abc->accepts2(\"caaaaaaaaaaaaaaad\") = " << abc-
>accepts2("caaaaaaaaaaaaaaad") << endl;
  });
  TimeAccept(abc, [](NFA* abc) {
    cout << "abc->accepts3(\"caaaaaaaaaaaaaaad\") = " << abc-
>accepts3("caaaaaaaaaaaaaaad") << endl;
  });

// ... same as 3a)
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL

abc->accepts2("caaaaaaaaaaaaaaad") = false
abc->accepts3("caaaaaaaaaaaaaaad") = false
3.b)
------

abc->accepts1("cabcabcabcabcc") = true
Elapsed time: 7849 micro sec
abc->accepts2("cabcabcabcabcc") = true
Elapsed time: 383 micro sec
abc->accepts3("cabcabcabcabcc") = true
Elapsed time: 482 micro sec
abc->accepts1("caaaaaaaaaaaaaaad") = false
Elapsed time: 9202 micro sec
abc->accepts2("caaaaaaaaaaaaaaad") = false
Elapsed time: 396 micro sec
abc->accepts3("caaaaaaaaaaaaaaad") = false
Elapsed time: 467 micro sec
3.c)
------
```

c)

Code:

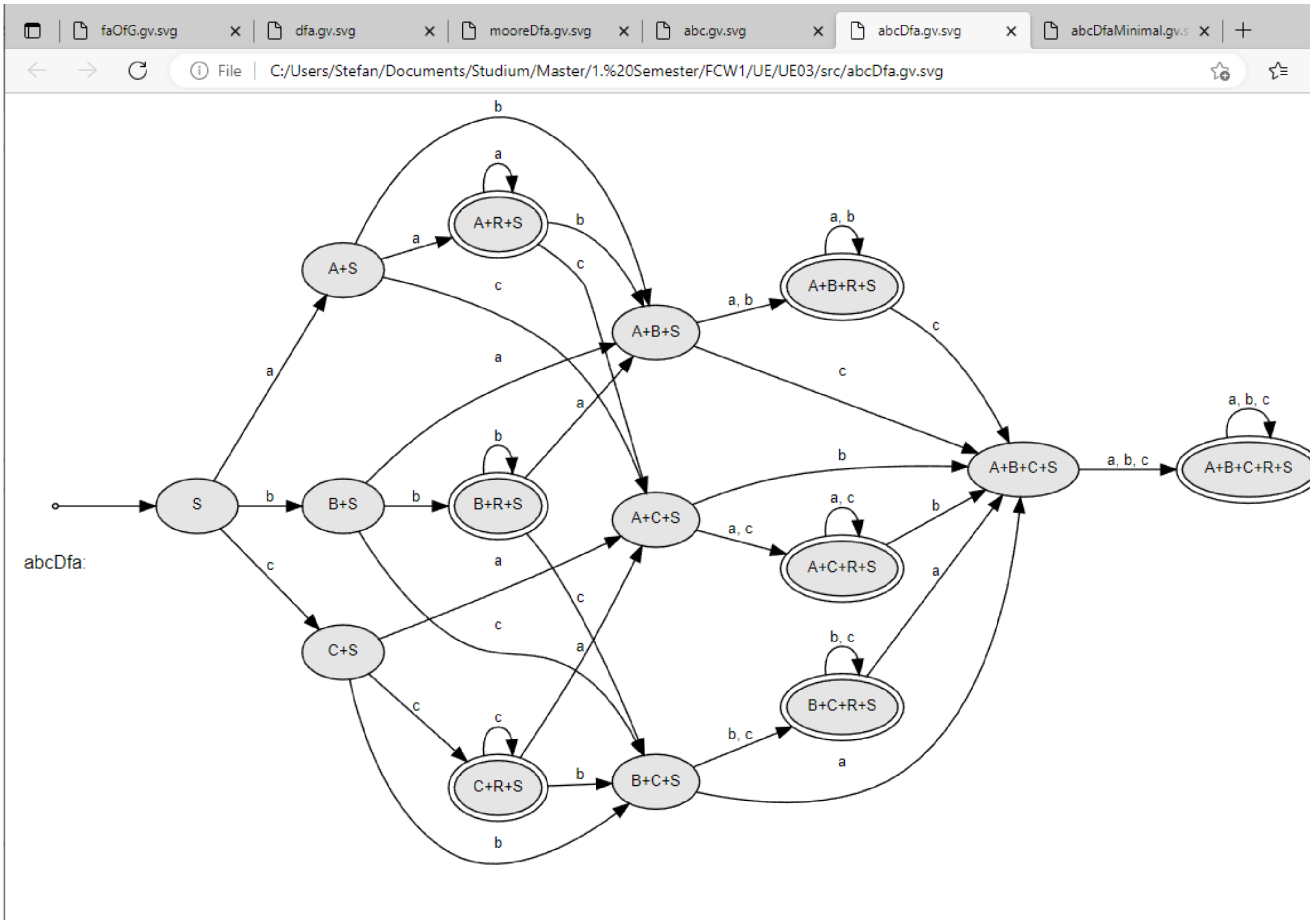
```
// ... same as 3a)

  DFA* abcDfa = abc->dfaOf();
  vizualizeFA("abcDfa", abcDfa);
  delete abcDfa;

// ... same as 3a)
```

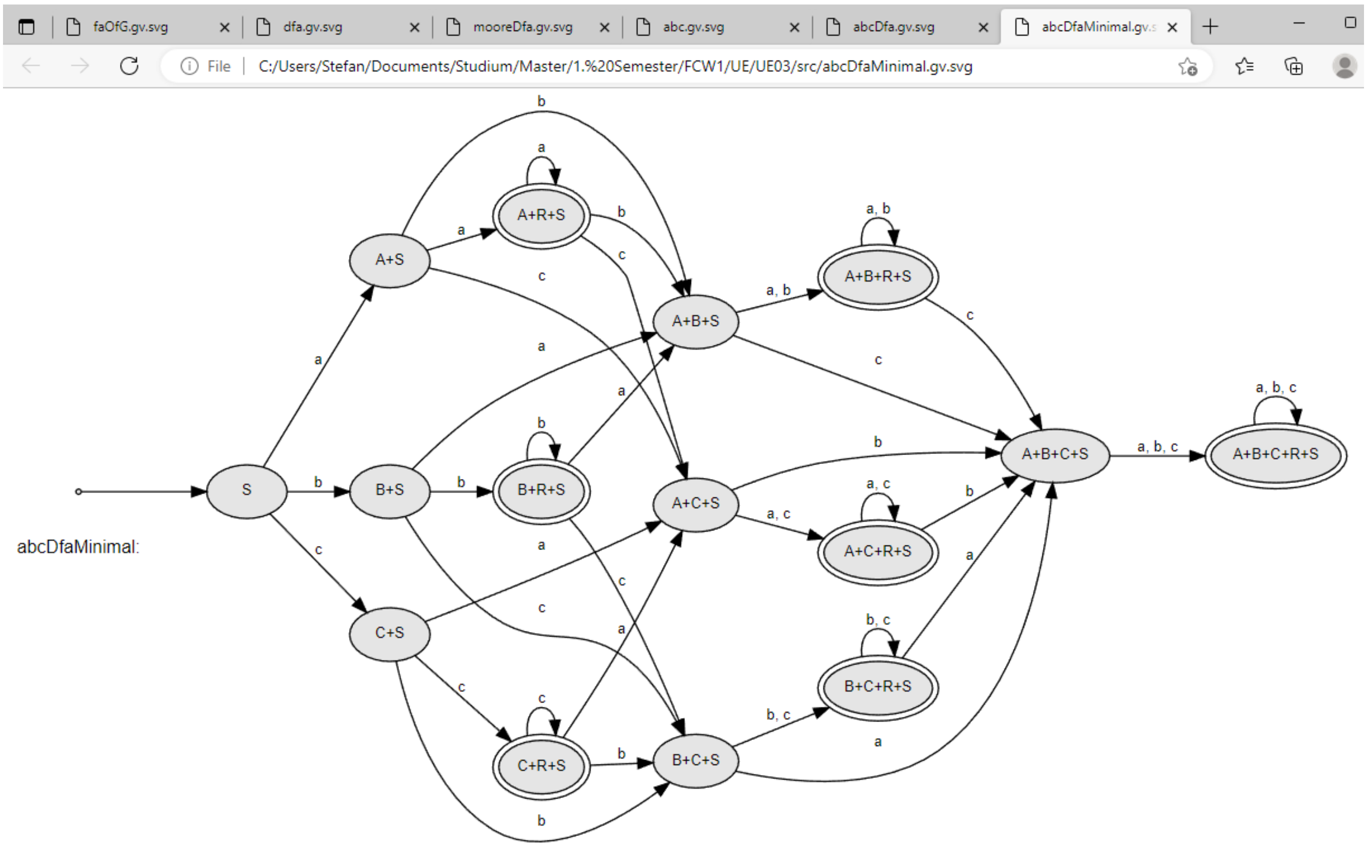


d)

Code:

```
// ... same as 3a)

   abcDfa = abc->dfaOf();
   DFA* abcDfaMinimal = abcDfa->minimalOf();

   vizualizeFA("abcDfa", abcDfa);
   vizualizeFA("abcDfaMinimal", abcDfaMinimal);

   delete abcDfaMinimal;
   delete abcDfa;

// ... same as 3a)
```





abcDfa war bereits minimal, da sich **abcDfa** nach der Minimierung nicht verändert hat.

4. Kellerautomat und erweiterter Kellerautomat

a)

```
Declaration     -> VAR | VAR VarDeclList .
VarDeclList     -> VarDecl ";" | VarDecl ";" VarDeclList .
VarDecl         -> IdentList ":" Type .
IdentList       -> ident | ident "," IdentList .
Type            -> ARRAY "(" number ")" OF TypeIdent | TypeIdent .
TypeIdent       -> INTEGER | BOOLEAN | CHAR .
```

b)

```
S1:
d(Z, e, Declaration)    = (Z, VAR)
d(Z, e, Declaration)    = (Z, VarDeclList VAR)
d(Z, e, VarDeclList)    = (Z, ";" VarDecl)
d(Z, e, VarDeclList)    = (Z, VarDeclList ";" VarDecl)
d(Z, e, VarDecl)        = (Z, Type ":" IdentList)
d(Z, e, IdentList)      = (Z, ident)
d(Z, e, IdentList)      = (Z, IdentList "," ident)
d(Z, e, Type)           = (Z, TypeIdent OF ")" number "(" ARRAY)
d(Z, e, Type)           = (Z, TypeIdent)
d(Z, e, TypeIdent)      = (Z, INTEGER)
d(Z, e, TypeIdent)      = (Z, BOOLEAN)
d(Z, e, TypeIdent)      = (Z, CHAR)

S2:
d(Z, VAR, VAR)          = (Z, e)
d(Z, ";", ";")          = (Z, e)
d(Z, ident, ident)      = (Z, e)
d(Z, ":", ":")          = (Z, e)
d(Z, ",", ",")          = (Z, e)
d(Z, ARRAY, ARRAY)      = (Z, e)
d(Z, number, number)    = (Z, e)
d(Z, OF, OF)            = (Z, e)
d(Z, ")", ")")          = (Z, e)
d(Z, "(", "(")          = (Z, e)
d(Z, INTEGER, INTEGER)  = (Z, e)
d(Z, BOOLEAN, BOOLEAN)  = (Z, e)
d(Z, CHAR, CHAR)        = (Z, e)
```

d(Z, e, TypeIdent) = (Z, INTEGER) d(Z, e, TypeIdent) = (Z, BOOLEAN) d(Z, e, TypeIdent) = (Z, CHAR) d(Z, INTEGER, INTEGER) = (Z, e) => reduktion (Z, VAR, VAR) = (Z, e)

(Z, Declaration .VAR a, b: INTEGER;) (Z, VarDeclList **VAR** . VARa, b: INTEGER;) (Z, VarDeclList . a, b: INTEGER;)

5. Term. Anfänge/Nachfolger, LL(k)-Bedingung u. Transformation