

- ☐ Gr. 1, Dr. H. Dobler
☐ Gr. 2, Dr. G. Kronberger

Name _____ Aufwand in h _____

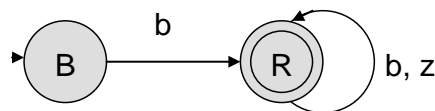
Punkte _____ Übungsleiter _____

1. Objektorientierte Implementierung endlicher Automaten (1 + 4 Punkte)

- a) Machen Sie sich mit der objektorientierten Implementierung endlicher Automaten in C++ vertraut. Sie finden diese im *moodle*-Kurs in der Datei *AutomataForStudents*, im Wesentlichen in den beiden Klassen *DFA* und *NFA*: Studieren Sie die Quelltexte anhand der beiden Testprogramme *DFATest* und *NFATest*.
- b) Um das Verständnis (auch der oo Implementierung von Grammatiken) weiter zu festigen, erstellen Sie eine Funktion zur *XFA* **xfaOf(const Grammar *g)* Transformation einer regulären Grammatik (gegeben in Form eines *Grammar*-Objekts) in einen endlichen Automaten (also in ein Objekt der Klasse *DFA* oder *NFA*, je nachdem welche Klasse Ihnen dafür besser geeignet erscheint) sowie eine Funktion *Grammar *grammarOf(const XFA *xfa)* für die umgekehrte Transformation (also *NFA* oder *DFA* nach *Grammar*).

2. DFA, Erkennung und Mealy- oder Moore-Automat (1 + 2 Punkte)

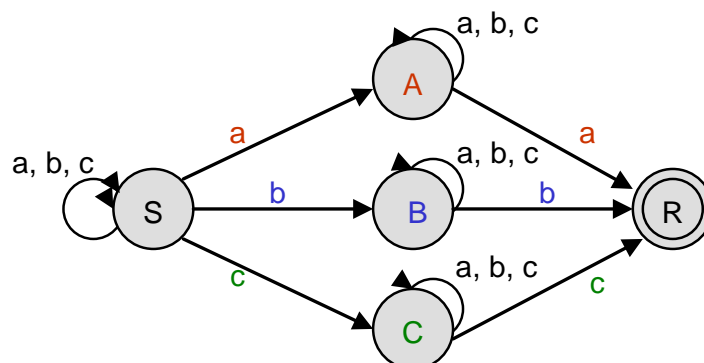
- a) Schreiben Sie ein Programm, das den unten dargestellten Automaten für einfache Bezeichner erzeugt (in Form eines Objekts der Klasse *DFA*) und versuchen Sie, mit der *accepts*-Methode sowohl gültige als auch ungültige Bandinhalte zu erkennen.



- b) Entwickeln Sie ausgehend von der Klasse *DFA* eine neue Klassen (*Mealy* oder *Moore*), die einen endlichen Transformationsautomaten (nach Mealy oder Moore) simuliert. Testen Sie Ihre Klasse, indem Sie einfache Bezeichner (*b* steht für *Buchstabe*, *z* steht für *Ziffer*, z. B. *bzzb*) in's Englische übersetzen (*c* für *character* und *d* für *digit*, also z.B. *bzzb* -> *cddc*).

3. NFA, Transformation NFA -> DFA und Zustandsminimierung (2 + 2 + 2 Punkte)

- a) Schreiben Sie ein Programm, das den unten dargestellten Automaten für spezielle *abc*-Folgen erzeugt (in Form eines Objekts der Klasse *NFA*) und versuchen Sie mit den beiden Methoden *accepts* (verwendet Backtracking) und *accepts2* (verwendet Zustandsmengen) sowohl gültige als auch ungültige Bandinhalte zu erkennen.



- b) Instrumentieren Sie die beiden *accepts*-Methoden so, dass Sie zur Laufzeit Maßzahlen für den Zeitaufwand der Erkennung ermitteln können.
- c) Berechnen Sie mit der Methode *NFA::dfaOf* den deterministischen Automaten für obigen nichtdeterministischen Automaten und stellen Sie diesen graphisch (am besten mittels *GraphViz* von www.graphviz.org) dar.
- d) Stellen Sie fest, ob der in c) berechnete deterministische Automat minimal ist, indem Sie dafür, mit der Methode *DFA::minimalDfaOf* den Minimalautomaten berechnen und schauen, ob ...

4. Kellerautomat und erweiterter Kellerautomat

(1 + 1 + 1 + 2 Punkte)

Die Grammatik für Variablendeklarationen in der Sprache MiniModula-2 lautet:

```
Declaration = VAR { VarDecl ";" } .
VarDecl    = IdentList ":" Type .
IdentList   = ident { "," ident } .
Type        = [ ARRAY "(" number ")" OF ] TypeIdent .
TypeIdent   = INTEGER | BOOLEAN | CHAR .
```

- a) Transformieren Sie diese Grammatik in die Schreibweise der formalen Sprachen.
- b) Konstruieren Sie einen *Kellerautomaten* für Sätze dieser Grammatik. (Algorithmus siehe unten.)
- c) Konstruieren Sie einen *erweiterten Kellerautomaten* für die Sätze dieser Grammatik. (Algorithmus siehe unten.)
- d) Geben Sie die Zugfolgen der beiden Kellerautomaten aus b) und c) an, die sie bei der Erkennung des Satzes

```
VAR a, b: INTEGER;
```

durchlaufen.

Algorithmus Kellerautomat aus Grammatik (nichtdeterministisch, *top-down*):

Der Kellerautomat besitzt nur einen einzigen Zustand Z (Start- und Endzustand), zu Beginn enthält der Keller nur das Satzsymbol S und erkennt Sätze durch leeren Keller.

S.1: Erzeuge für jede Regel $A \rightarrow \alpha$ einen Übergang $\delta(Z, \varepsilon, A) = (Z, \alpha^R)$.

Hierbei ist α^R die Umkehrung von α .

S.2: Erzeuge für jedes Terminalsymbol a einen Übergang $\delta(Z, a, a) = (Z, \varepsilon)$.

Algorithmus erweiterter Kellerautomat aus Grammatik (nichtdeterministisch, *bottom-up*):

Der erweiterte Kellerautomat besitzt zwei Zustände, Z und R . Dabei ist R Endzustand. Sein Keller enthält im Startzustand das nicht zur Grammatik gehörende Symbol $\$$.

S.1: Erzeuge für jede Regel $A \rightarrow \alpha$ einen Übergang $\delta(Z, \varepsilon, \alpha) = (Z, A)$.

S.2: Erzeuge für jedes Terminalsymbol a einen Übergang $\delta(Z, a, x) = (Z, xa)$ für alle $x \in V \cup \{\$\}$.

S.3: Erzeuge den Übergang $\delta(Z, \varepsilon, \$S) = (R, \varepsilon)$.

5. Term. Anfänge/Nachfolger, LL(k)-Bedingung u. Transformation (2 + 2 + 1 Punkte)

Wir betrachten eine abgeänderte und vereinfachte Form von Modula-2-Programmoduln und beschreiben sie durch folgende Grammatik:

```
progmod  → MODULE id : priority ; imppart block id .
priority → const | ε
imppart  → FROM id IMPORT implist | IMPORT implist
implist  → id | id , implist
block    → dclpart statpart | statpart
dclpart  → DECL | DECL ; dclpart
statpart → BEGIN statseq ; END
statseq  → STAT | STAT ; statseq
```

- Bestimmen Sie die terminalen Anfänge und Nachfolger der Länge 1 aller Nonterminalsymbole dieser Grammatik.
- Ist diese Grammatik LL(k)? Wenn ja, wie groß ist k ; wenn nein, warum nicht?
- Transformieren Sie diese Grammatik in eine äquivalente LL(1)-Grammatik und zeigen Sie, dass Ihre Grammatik tatsächlich LL(1) ist.