

☐ Gr. 1, Dr. H. DoblerName Yvonne MarnethAufwand in h 20h☒ Gr. 2, Dr. G. Kronberger

Punkte _____

Übungsleiter _____

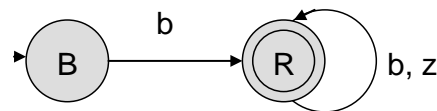
1. Objektorientierte Implementierung endlicher Automaten**(6 Punkte)**

Machen Sie sich zuerst mit der oo Implementierung endlicher Automaten in C++ vertraut. Sie finden diese im *moodle*-Kurs in der Datei *FiniteAutomataForStudents*, im Wesentlichen in den beiden Klassen *DFA* und *NFA*: Studieren Sie die Quelltexte anhand des Testprogramms in *Main.cpp*.

Um das Verständnis (auch der oo Implementierung von Grammatiken) weiter zu festigen, erstellen Sie eine Funktion *XFA *faOf(const Grammar *g)* zur Transformation einer regulären Grammatik (gegeben in Form eines *Grammar*-Objekts *g*) in einen endlichen Automaten (also in ein Objekt der Klasse *DFA* oder *NFA*, je nachdem welche Klasse Ihnen dafür besser geeignet erscheint) sowie eine zweite Funktion *Grammar *grammarOf(const XFA *xfa)* für die umgekehrte Transformation (also von *NFA* oder *DFA* nach *Grammar*, wobei diese dann aber regulär sein muss).

2. DFA, Erkennung und Mealy- oder Moore-Automat**(1 + 3 Punkte)**

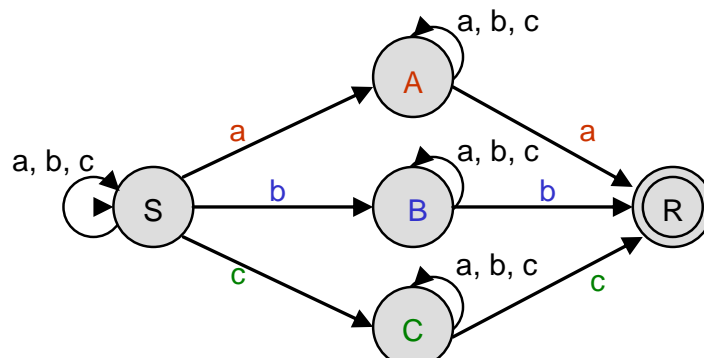
- a) Schreiben Sie ein Programm, das den unten dargestellten Automaten für einfache Bezeichner erzeugt (in Form eines Objekts der Klasse *DFA*) und testen Sie die *accepts*-Methode sowohl für gültige als auch für ungültige Bandinhalte.



- b) Entwickeln Sie ausgehend von der Klasse *DFA* eine neue Klasse (*Mealy* oder *Moore*), die einen endlichen Transformationsautomaten (nach George H. Mealy oder Edward F. Moore) simuliert. Testen Sie Ihre Klasse, indem Sie Bezeichner gemäß des obigen Automaten (*b* steht für *Buchstabe*, *z* steht für *Ziffer*, z. B. *bzzb*) in's Englische übersetzen (*c* für *character* und *d* für *digit*), also damit z.B. den dt. Bezeichner *bzzb* in den engl. Bezeichner *cddc* übersetzen.

3. NFA, Transformation NFA -> DFA und Zustandsminimierung (1 + 2 + 1 + 1 Punkte)

- a) Schreiben Sie ein Programm, das den unten dargestellten Automaten für spezielle *abc*-Folgen in Form eines Objekts der Klasse *NFA* erzeugt, und versuchen Sie mit den drei Methoden *accepts1* (mit Multithreading), *accepts2* (mit Backtracking) und *accepts3* (durch Verfolgung von Zustandsmengen) sowohl gültige als auch ungültige Bandinhalte zu erkennen.



- b) Instrumentieren Sie die drei *acceptsX*-Methoden so, dass Sie Maßzahlen für den Zeitaufwand und die Laufzeit (mittels *Timer.h* + *.cpp*) der Erkennung ermitteln können.
- c) Berechnen Sie mit der Methode *NFA::dfaOf* den deterministischen Automaten für obigen nichtdeterministischen Automaten und stellen Sie diesen graphisch 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)

Hier die Grammatik für die Definition von Konstanten in der Sprache MiniC++ (einer kleinen Teilmenge von C++):

```
ConstDef = 'const' Type ident Init { ',' ident Init } ';' .
Type =     'bool' | 'int' .
Init =     '=' ( false | true | [ '+' | '-' ] number ) .
```

- 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

```
const int max = 100;
```

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 ist 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 (1 + 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 der Länge 1 ($First_1$) aller Alternativen und terminalen Nachfolger der Länge 1 ($Follow_1$) 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 dann, dass Ihre Grammatik tatsächlich $LL(1)$ ist.