1. Objektorientierte Implementierung endlicher Automaten

a) faOf()

Lösungsidee:

Ich bin dabei so vorgegangen, wie ich die Grammatik händisch in einen endlichen Automaten umwandelt würde. Man betrachtet jede Regel der Grammatik. Das TSymbol auf der linken Seite der Regel (NTLeft) wäre beim Automaten der aktuelle Zustand, das Terminal-Symbol bei jeder Alternative wäre das Band-Symbol. Wenn ein Band-Symbol von einem NTSymbol gefolgt ist, dann entspricht es dem Anwenden der Zustandsüberführungsfunktion von (NTLeft, Band-Symbol) => NTNext, also wäre das eine Transition, die im Automaten einzutragen ist.

Das gleiche Band-Symbol kann bei regulären Sprachen in den Alternativen einer Regeln maximal 2x vorkommen (einmal alleine, einmal gefolgt von einem). Falls ein Band-Symbol 2x vorkommt, dann ist es einmal von einem NTSymbol gefolgt (NTNext, bei rechtsregulären Sprachen) und einmal steht es alleine da. Wenn das Band-Symbol von keinem NTSymbol gefolgt ist, dann muss es zumindest die Alternative in der aktuellen Regel geben, wo das Band-Symbol von einem NTSymbol (NTNext) gefolgt wird. Zu Beginn werden die Folgezustände von den Band-Symbol ohne folgenden NTSymbol ermittelt. Kann also ein End-Zustand sein.

Code:

Testfälle:

b) grammarOf()

Lösungsidee:

Es werden alle Kombinationen von Zuständen und Band-Symbolen auf die Zustandsüberführungsfunktion angewandt. 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 erfolgreichen Anwendung der Zustandsüberführungsfunktion ist der **State**, der als Parameter für **deltaOf()** verwendet wurde, die Regel der Grammatik und das Band-Symbol und die von **deltaOf()** gelieferten Zustande die Sequenz dieser Regel. Falls der **deltaOf()** gelieferten Zustand ein End-Zustand ist, dann muss auch eine Alternative ohne dem von **deltaOf()** gelieferten Zustand in die Regeln des verwendeten **State** (nur dem Band-Symbol als Sequenz) eingetragen werden.

Code:

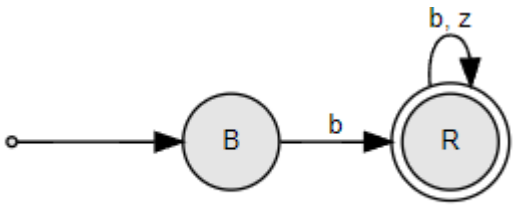
Testfälle:

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

a)

Code:

```
fab = new FABuilder(); // example from FS slides p. 47
fab->setStartState("B").
  addFinalState("R").
  addTransition("B", 'b', "R").
  addTransition("R", 'b', "R").
  addTransition("R", 'z', "R");
dfa = fab->buildDFA();
delete fab;
```



dfa:

Tests:

b)

3. NFA, Transformation NFA -> DFA und Zustandsminimierung

4. Kellerautomat und erweiterter Kellerautomat

für jede regel übeführung machen (in VO unterlagen nachsehen) Arguments: Zustand NT vom stack (liest aber nix vom band) => legt alpha uaf stack in umgekehrter reihenfolge

d(Z, e, Declaration) = (Z, VarDeclList VAR) => VarDeclList erstellen d(Z, e, VarDecl) = (Z, Type ":" IdentList) ...

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