Aufgabe 1: 3-Schichten-Architektur

Benennen Sie die drei Schichten einer 3-Schichten-Architektur und beschreiben Sie kurz deren jeweilige Funktion.

Schicht 1: Präsentationsschicht

Die Präsentationsschicht (engl. client layer oder presentation layer) ist für die Darstellung und Entgegennahme von der Software bearbeiteten Daten, sowie der von der Software bereitgestellten Funktionen verantwortlich.

Schicht 2: Geschäftslogikschicht

Die Geschäftslogikschicht (auch Verarbeitungsschicht, Anwendungs-logikschicht, Domänenschicht, application layer oder middle tier) realisiert das eigentliche Geschäftsmodell, indem die am Geschäftsmodell beteiligten Geschäftsobjekte mit der zugehörigen Logik implementiert werden.

Schicht 3: Persistenzschicht

Die Persistenzschicht ist für die persistente Ablage von Daten(-objekten) zuständig. Die Anwendungskomponenten greifen mit Hilfe der Datenzugriffsfunktionen auf die Persistenzschicht zu und abstrahieren so von der konkreten technischen Persistenz.

Name:_____ Matr.-Nr: _____

Aufgabe 2: Grammatiken

Gegeben sei eine Grammatik wie folgt (ε ist Leersymbol):

G = <Startsymbol, Terminale, Nicht-Terminale, Regeln> mit

Startsymbol = $Start$

Terminale = $\{a, b, c, d\}$

Nicht-Terminale = $\{X, Y\}$

Regeln = $\{ Start \rightarrow XY,$
$X \rightarrow aX,$
$X \rightarrow bX,$
$X \rightarrow b,$
$Y \rightarrow cY$
$Y \rightarrow d \}$

Entscheiden Sie, ob folgende Wörter in der durch die Grammatik definierten Sprache enthalten sind:

- ε ~~ja~~ / nein
- aabccb ~~ja~~ / nein
- abcd ja / ~~nein~~
- aabbaacb ~~ja~~ / nein
- acdc ~~ja~~ / nein
- aabbabcd ja / ~~nein~~
- abc ~~ja~~ / nein
- aabccd ja / ~~nein~~
- bd ja / ~~nein~~
- aabbbc ~~ja~~ / nein

Bestimmen Sie die Wörter, die durch die Grammatik akzeptiert werden in der Art $a^n b^n$:

$[a,b]^* bc^* d$

bzw.

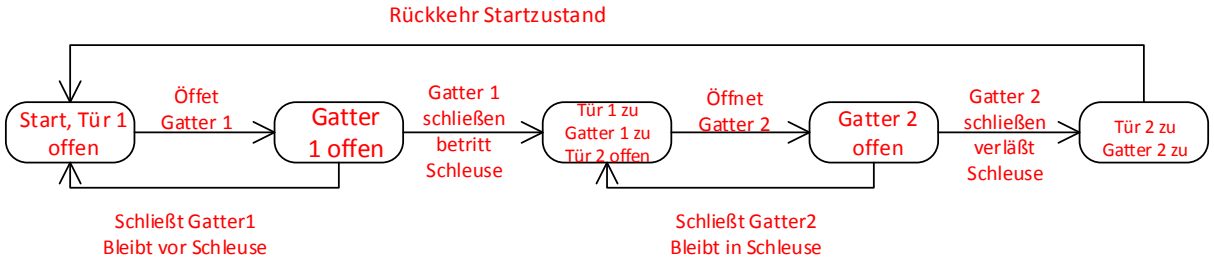
$[a,b]^m bc^n d$ mit $m,n \geq 0$

Name:_____ Matr.-Nr: _____

Aufgabe 3: Endliche Automaten

Gegeben sei eine Doppeltürschleuse, wo die zweite Tür sich erst öffnen lässt, wenn sich eine Person durch die erste Tür in die Schleuse begeben hat und die erste Tür durch Schließen des Gatters hinter sich geschlossen ist. Erst wenn die die Person die Schleuse verlassen hat und zweite Tür wieder geschlossen ist, ist die Schleuse wieder im Startzustand. Ein Sensor überprüft, ob sich eine Person in der Schleuse befindet. Bevor ein Gatter geschlossen ist, steht die jeweilige Tür offen. Als Beispiel sei gegeben:

KABA BEYOND SECURITY



Name:_____ Matr.-Nr: _____

Aufgabe 4: Prolog Unifikation

Gegeben seien jeweils 2 Prolog-Ausdrücke. Sie sollen entscheiden, ob diese beiden Ausdrücke unifizierbar sind und, falls dieses der Fall ist, den durch Substitution erzeugten Ausdruck angeben. Sollte die Unifikation misslingen, geben Sie den Grund an. Als Beispiele sind gegeben (Konstanten fangen mit kleinem, Variablen mit großem Buchstaben an):

- test (X, foo, Y) = test (M, N, M) ☺ mit test (X, foo, X).
- test (kai, X) = test1(kai, udo) ☹ weil test ≠ test1.

Hier nun die Ausdrücke:

- comp(Mainboard, intel(i7)) = comp(asus(p6t), CPU).

☺ mit Mainboard = asus(p6t), CPU = intel(i7).

- comp(Mainboard, intel(i7)) = comp(asus(p6t), amd(Type)).

☹ intel ≠ amd.

- append(X, [4, 5, 7], Y) = append([1,2,3], Z, [a, b, c]).

☺ mit X = [1, 2, 3], Y = [a, b, c], Z = [4, 5, 6].

- append([1, 2, 3], [4, 5, 6], X), X = [a, b, c].

☹ mit [1, 2, 3, 4, 5, 6] ≠ [a, b, c].

Name:_____ Matr.-Nr: _____

Aufgabe 5: Prolog Programmanalyse

Gegeben sei ein Prolog-Programm `foo(List1, List2)` wie folgt (`reverse/2` dreht die Reihenfolge einer Liste um):

```
foo(Arg1, Arg2) :-
    reverse(Arg1, Temp1),
    append(Arg1, Temp1, Temp2),
    append(Temp2, Arg1, Arg2).
```

Beschreiben Sie kurz, was das Programm macht und geben Sie jeweils ein Beispiel mit belegtem 1. Argument und variablem 2. Argument wie belegtem 2. Argument und variablem 1. Argument:

Das Programm `foo/2` evaluiert zu wahr, wenn beide Argumente Listen sind und das 2. Argument eine Konkatenation der ersten Liste mit ihrer Umkehrung gefolgt mit der ursprünglichen ersten Liste ist.

Beispiele:

```
foo([1, 2, 3], Bar)
-> Bar = [1, 2, 3, 3, 2, 1, 1, 2, 3].

foo(Bar, [1, 2, 3, 3, 2, 1, 1, 2, 3].
-> Bar = [1, 2, 3].
```

Bonus-Aufgabe: was macht folgende Funktion:

```
cool(X,Y) :-
    cool(X, [], Y, Y).

cool([], List, List, []).
cool([First|Rest], List1, List2, [_|Rest2]) :-
    cool(Rest, [First|List1], List2, Rest2).
```

`cool/2` implementiert `reverse/2`

Name:_____ Matr.-Nr: _____

Bewertung der Aufgaben

Aufgabe 1:	irgendwie zutreffende Beschreibung (maximal 3 Punkte)	=	1 Punkt
Aufgabe 2:	pro richtige Wortentscheidung richtige regulärer Ausdruck (maximal 6 Punkte)	= =	0.5 Punkte 1 Punkt
Aufgabe 3:	richtiger Automat fast richtiger Automat richtiger Ansatz fast richtiger Ansatz (maximal 4 Punkte)	= = = =	4 Punkte 3 Punkt 2 Punkt 1 Punkt
Aufgabe 4:	jeweils richtiges Ergebnis (maximal 4 Punkte)	=	1 Punkt
Aufgabe 5:	richtige Codebeschreibung fast richtige Codeskizze richtige Beispiele Bonusfrage (maximal 5 Punkte)	= = = =	3 Punkte 2 Punkte 1 Punkt 1 Punkt
Maximale Punkte aller Aufgaben:		=	22 Punkte

Notengebung: Maximum(Minimum(Aufrunden(Summe aller Punkte - 5),15),0)

Name:_____ Matr.-Nr: _____