

<b>Name:</b>	_____
<b>Vorname:</b>	_____
<b>Matrikel-Nr.:</b>	_____

**Klausur Formale Systeme**  
Fakultät für Informatik  
SS 2023

Prof. Dr. Bernhard Beckert  
4. September 2023

*Die Bearbeitungszeit beträgt 60 Minuten.*

A1 (9)	A2 (12)	A3 (8)	A4 (7)	A5 (8)	A6 (9)	A7 (7)	$\Sigma$ (60)

**Bewertungstabelle bitte frei lassen!**

**Gesamtpunkte:**

# 1 Zur Einstimmung ((2 + 2) + (1 + 1 + 1) + 2 = 9 Punkte)

- a. Seien  $p, q$  einstellige Prädikatensymbole,  $r$  ein zweistelliges Prädikatensymbol und  $f$  ein einstelliges Funktionssymbol.

Geben Sie für folgende prädikatenlogische Formeln – **falls möglich** – jeweils zwei Interpretationen  $I$  über dem Universum  $D = \{a, b\}$  an, und zwar jeweils

- eine Interpretation, in der die Formel **wahr** ist, und
- eine Interpretation, in der die Formel **falsch** ist.

In den Fällen, in denen eine Interpretation mit der gesuchten Eigenschaft **nicht existiert**, geben Sie dies an.

**Hinweis:** Es muss explizit angegeben werden, wenn eine passende Interpretation nicht existiert (schreiben Sie „existiert nicht“ neben den Kasten). Es genügt nicht, die Beschreibung der Interpretation leer zu lassen.

- i.  $\forall x ((\exists y p(y)) \wedge q(x)) \rightarrow \neg \forall z (p(z) \vee q(z))$

Interpretation, in der die Formel wahr ist:

$I(p)(a) = \mathbf{F}$	$I(p)(b) = \mathbf{F}$	$I(q)(a) = \mathbf{F}$	$I(q)(b) = \mathbf{F}$
------------------------	------------------------	------------------------	------------------------

Interpretation, in der die Formel falsch ist:

$I(p)(a) = \mathbf{W}$	$I(p)(b) = \mathbf{W}$	$I(q)(a) = \mathbf{W}$	$I(q)(b) = \mathbf{W}$
------------------------	------------------------	------------------------	------------------------

- ii.  $(\exists x \forall y r(x, f(y))) \rightarrow (\exists x \forall y r(f(x), y))$

Interpretation, in der die Formel wahr ist:

$I(r)(a, a) = \mathbf{W}$	$I(r)(a, b) = \mathbf{W}$	$I(r)(b, a) = \mathbf{W}$	$I(r)(b, b) = \mathbf{W}$
---------------------------	---------------------------	---------------------------	---------------------------

$I(f)(a) = a$	$I(f)(b) = b$
---------------	---------------

Interpretation, in der die Formel falsch ist:

$I(r)(a, a) = \mathbf{W}$	$I(r)(a, b) = \mathbf{W}$	$I(r)(b, a) = \mathbf{W}$	$I(r)(b, b) = \mathbf{F}$
---------------------------	---------------------------	---------------------------	---------------------------

$I(f)(a) = b$	$I(f)(b) = b$
---------------	---------------

## Fortsetzung 1 Zur Einstimmung

b. Geben Sie kurze Antworten zu folgenden Fragen bzw. Aufgaben:

i. Nennen Sie eine Konsequenz des Gödelschen Unvollständigkeitssatzes.

Jede konsistente, rekursiv aufzählbare Axiomenmenge für die Arithmetik natürlicher Zahlen ist unvollständig. (Es gibt viele weitere.)

ii. Erläutern Sie den Unterschied zwischen  $\models$  und  $\vdash$ .

$\vdash$  bezieht sich auf die Ableitung (Syntaxumformung) im Kalkül.

iii. In der Vorlesung haben Sie Konfluenz als Eigenschaft von Termersetzungssystemen  $(D, \rightarrow^*)$  kennengelernt. Konfluenz lässt sich aber auch auf Kalküle anwenden, indem  $D$  die Menge der Beweissituationen (z.B. Tableaus) und der Übergang  $\rightarrow^*$  die Anwendung einer Kalkülregel ist. Begründen Sie warum die Konfluenz für die Beweissuche eine wichtige Eigenschaft darstellt.

Für konfluente Kalküle ist die Reihenfolge der Regelanwendungen nicht entscheidend. Niemals wird eine Sackgasse erreicht, in welcher man eine wahre Aussage nicht mehr zeigen kann.

Hinweis: Vollständigkeit sagt nur aus, dass es für jede wahre Aussage einen Beweis gibt.

c. Zeigen Sie die Unerfüllbarkeit folgender aussagenlogischer Klauselmengen über  $\Sigma = \{P, \dots, T\}$  mit dem **Markierungsalgorithmus**:

$$(Q \wedge P \rightarrow R) \wedge (S \wedge P \rightarrow Q) \wedge (R \wedge T \wedge P \rightarrow \mathbf{0}) \wedge S \wedge (S \rightarrow P) \wedge (P \wedge Q \wedge R \rightarrow \mathbf{0})$$

Geben Sie dabei an, in welcher Reihenfolge die Atome markiert werden.

Markierungsreihenfolge: S, P, Q, R

## 2 Abstrakte Datentypen und Prädikatenlogik

(4 + 3 + 3 + 2 = 12 Punkte)

Man kann abstrakte Datentypen (ADTs) in Prädikatenlogik formalisieren. Dazu sei eine Signatur  $\Sigma = (P_\Sigma, F_\Sigma, \alpha_\Sigma)$  gegeben, und wir gehen davon aus, dass die Menge  $K \subseteq F_\Sigma$  der Konstruktoren des ADTs eine Teilmenge der Funktionssymbole  $F_\Sigma$  ist. Zudem sei ein prädikatenlogisches Modell  $M = (D, I)$  über der Signatur  $\Sigma$  gegeben.

Beispielsweise kann man mit folgenden Konstruktoren den ADT *Binärbaum* formalisieren:

- *leaf*: eine Konstante (für Blätter)
- *tree*( $x, y$ ): eine zweistellige Funktion ( $x$  und  $y$  sind der linke bzw. der rechte Teil-Baum)

**Hinweis:** Die Signatur kann neben Konstruktoren weitere Funktionen enthalten, wie bspw. *left*, *right*, die man auf Bäume anwenden kann. Das ist für diese Aufgabe aber nicht relevant.

- a. Ein ADT heißt **generiert**, wenn

für alle  $d \in D$  gilt: es gibt einen Grundterm  $t$ , der nur Konstruktoren enthält, mit  $I(t) = d$

Man könnte meinen, dass Generiertheit in Prädikatenlogik formalisierbar ist. Das ist tatsächlich aber im allgemeinen nicht möglich. Versuchen könnte man es aber (vergeblich) für den Beispiel-ADT *Binärbaum* mit der Formel

$$\forall tr (tr \doteq leaf \vee \exists x \exists y (tr \doteq tree(x, y))) .$$

Begründen Sie kurz, warum diese Formel nicht formalisiert, dass der obige Beispiel-ADT generiert ist.

Die obige Formel lässt zu, dass Bäume Zyklen haben. Es könnte sein, dass  $tr \doteq tree(tr, leaf)$ . Für solch ein  $tr$  wäre die Formel wahr, ohne dass  $tr$  notwendig durch einen Grundterm dargestellt werden kann.

- b. Ein ADT heißt **frei generiert**, wenn er generiert ist und zudem für alle Grundterme  $t_1, t_2$ , die nur Konstruktoren enthalten, gilt:

$$\text{wenn } t_1 \neq t_2 \text{ dann } I(t_1) \neq I(t_2)$$

Man kann in Prädikatenlogik formalisieren, dass ein gegebener ADT, von dem man weiß, dass er generiert ist, darüber hinaus auch *frei* generiert ist. Zeigen Sie für den Beispiel-ADT *Binärbaum*, dass das geht, indem Sie folgende prädikatenlogische Formel passend ergänzen:

$$\forall x \forall y ( \neg (tree(x, y) \doteq leaf) \wedge \boxed{\forall u \forall v (tree(x, y) \doteq tree(u, v) \rightarrow ((x \doteq u) \wedge (y \doteq v)))} )$$

- c. Ein wesentlicher Vorteil generierter ADTs ist, dass man eine korrekte und vollständige Induktionsregel angeben kann. Vervollständigen Sie in der Box die folgende Induktionsregel für den Beispiel-ADT *Binärbaum*.

**Hinweis:**  $\phi(u)$  ist dabei eine beliebige prädikatenlogische Formel mit einer freien Variable  $u$ .

$$\frac{\phi(\text{leaf}) \quad \boxed{\forall x \forall y ((\phi(x) \wedge \phi(y)) \rightarrow \phi(\text{tree}(x, y)))}}{\forall u \phi(u)}$$

- d. Wenn der ADT nicht generiert ist, ist die Regel aus Teilaufgabe c nicht korrekt. Begründen Sie kurz, warum das so ist.

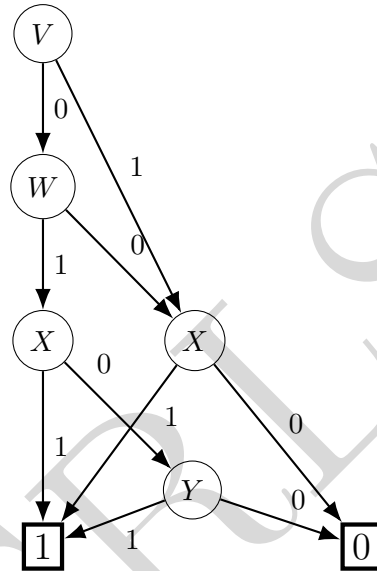
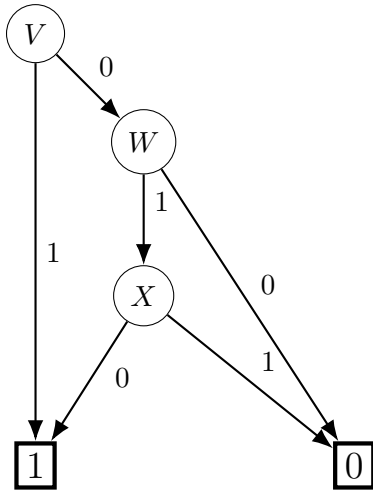
Die Prämissen der Regel stellen sicher, dass  $\phi$  für alle Elemente gilt, die als Grundterme über den Konstruktoren darstellbar sind. Wenn der ADT generiert ist, sind das alle Elemente in  $D$ . Wenn er nicht generiert ist, kann es weitere Elemente geben, für die dann  $\phi$  unter Umständen (je nach Modell) nicht gilt, so dass die Konklusion der Regel falsch ist.

### 3 Binary Decision Diagrams

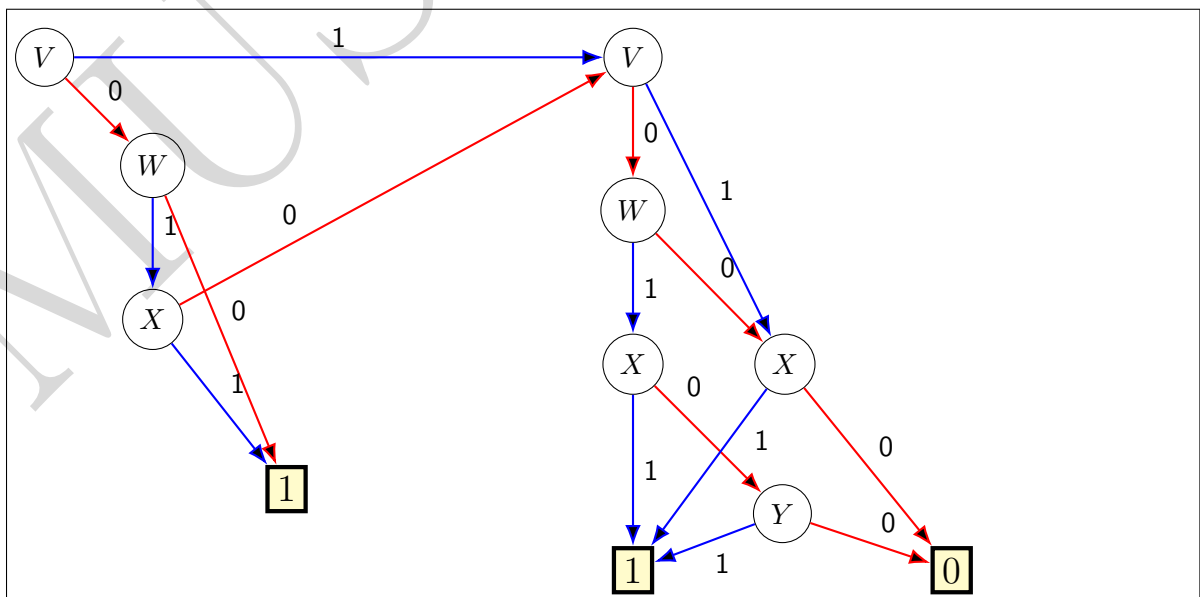
(3 + 3 + 2 = 8 Punkte)

Diese Aufgabe beschäftigt sich mit Binary Decision Diagrams (auch Shannongraphen genannt).

- a. Unten sehen Sie den Shannongraphen  $G$  für die Formel  $f_G \equiv V \vee (W \wedge \neg X)$ . Zeichnen Sie einen Shannongraphen  $H$  für die Formel  $f_H \equiv (\neg V \wedge W \wedge Y) \vee X$ . Nutzen Sie dabei die folgende Ordnung der Variablen:  $V < W < X < Y$ .



- b. Seien  $f_G$  und  $f_H$  die von Ihren Shannongraphen in a) beschriebenen Formeln. Geben Sie einen Shannongraphen für  $f_G \rightarrow f_H$  an. Sie müssen Ihre Shannongraphen aus a) dabei wiederververwenden. Eine Reduktion des Graphen ist nicht notwendig; die Ordnung der Variablen aus a) muss **nicht** beachtet werden.



### Fortsetzung 3 Binary Decision Diagrams

- c. Geben Sie eine allgemeine Konstruktionsvorschrift an um aus zwei beliebigen gegebenen Shannongraphen  $G, H$  einen Shannongraphen für  $f_G \rightarrow f_H$  zu konstruieren. Beschreiben sie explizit die Transformation des Graphen und *nicht* die Transformation als Abfolge logischer Operationen.

Für die Konstruktion der Implikation ist die Schreibweise der Implikation als Disjunktion sehr hilfreich:  $f_G \rightarrow f_H$  ist äquivalent zu  $\neg f_G \vee f_H$ .

Auf Basis dieser Umformung lässt sich nun ein Verfahren angeben:

- i. Man leitet alle eingehenden Kanten des 1-Terminalknotens von  $G$  um: Diese müssen auf die Wurzel von  $H$  zeigen.
- ii. Man entfernt den alten 1-Terminalknoten von  $G$
- iii. Man tauscht den alten 0-Terminalknoten von  $G$  durch einen 1-Terminalknoten aus.

Ggf. kann man den Graphen im Anschluss reduzieren.

## 4 Formalisieren in Prädikatenlogik (PL1)

(1 + 1 + 2 + 3 = 7 Punkte)

Gegeben sei die prädikatenlogische Signatur  $\Sigma = (\{mother, author\}, \{person, program, trusts\}, \alpha)$ . Sie enthält die einstellige Prädikatensymbole  $person(\cdot)$  und  $program(\cdot)$ , das zweistellige Prädikatensymbol  $trusts(\cdot, \cdot)$ , sowie die einstelligen Funktionssymbole  $mother(\cdot)$  und  $author(\cdot)$ .

Zur Auswertung der Formeln werden nur solche Interpretationen  $(D, I)$  über  $\Sigma$  verwendet, in denen

- das Universum  $D$  eine Menge von Personen und Computerprogrammen ist,
- das Prädikat  $person(x)$  genau dann wahr ist, wenn  $x$  eine Person ist,
- das Prädikat  $program(x)$  genau dann wahr ist, wenn  $x$  ein Computerprogramm ist,
- das Prädikat  $trusts(x, y)$  genau dann wahr ist, wenn  $x$  dem Element  $y$  vertraut,
- die Funktion  $mother(x)$  eine Person  $x$  auf die Mutter von  $x$  abbildet und
- die Funktion  $author(x)$  ein Computerprogramm  $x$  auf das Element abbildet, das  $x$  geschrieben hat.

Geben Sie jeweils eine Formel der Prädikatenlogik mit Gleichheit über  $\Sigma$  an, die folgende Sachverhalte darstellt:

- a. Personen sind keine Computerprogramme.

$$\forall x (person(x) \rightarrow \neg program(x))$$

- b. Es gibt eine Person, die keiner Person außer sich selbst vertraut.

$$\exists x (person(x) \wedge \forall y (person(y) \wedge trusts(x, y) \rightarrow x \doteq y))$$

- c. Es gibt Personen, die einer Person genau dann vertrauen, wenn ihre Mutter dieser Person vertraut.

$$\exists x \left( person(x) \wedge \forall y \left( person(y) \rightarrow (trusts(x, y) \leftrightarrow trusts(mother(x), y)) \right) \right)$$

- d. Computerprogramme vertrauen grundsätzlich nur Computerprogrammen und Personen, die ein Computerprogramm geschrieben haben.

$$\forall x \forall y \left( program(x) \wedge trusts(x, y) \rightarrow \left( program(y) \vee \left( person(y) \wedge \exists z (program(z) \wedge y \doteq author(z)) \right) \right) \right)$$



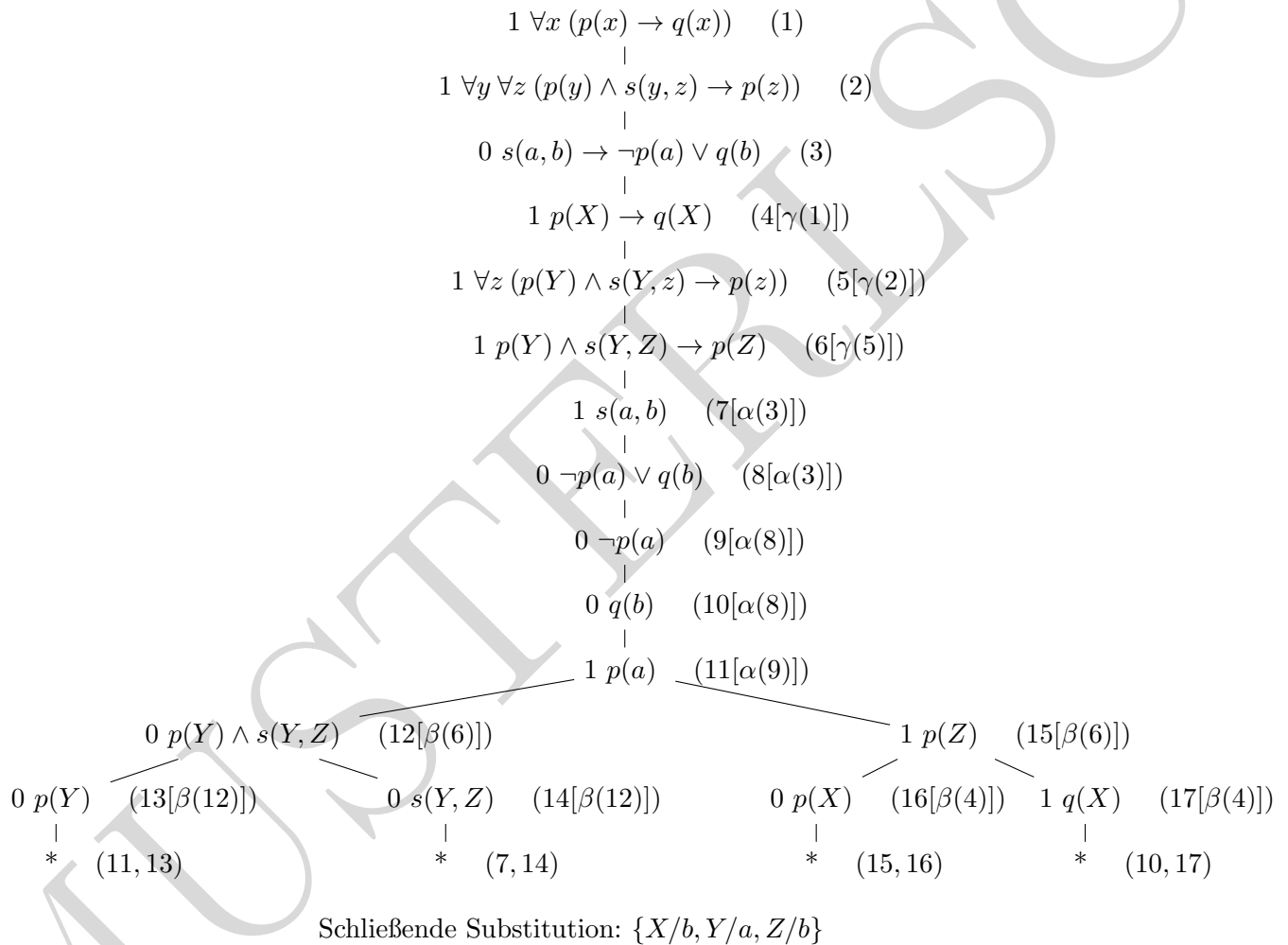
## 5 Tableaukalkül

(8 Punkte)

Vervollständigen und schließen Sie den folgenden Tableaubeweis.

Notieren Sie

- bei jeder Erweiterung den Regeltyp ( $\alpha, \beta, \gamma, \delta$ ) und die Formel, auf die die Regel angewendet wird.
- bei jedem Abschluss, welche Formeln verwendet wurden.
- die schließende Substitution.



## 6 Spezifikation mit der Java Modeling Language

(4 + (3 + 2) = 9 Punkte)

- a. Geben Sie in natürlicher Sprache wieder, was der folgende JML-Methodenvertrag für die Methode `m` aussagt.

**Hinweis 1:** Achten Sie auf die Klammerung!

**Hinweis 2:** Der Ausdruck  $(\text{\product int } j; a \leq j \ \&\& \ j < b; f(j))$  für gegebene Integer `a` und `b` steht für das Produkt  $\prod_{j=a}^{b-1} f(j)$ .

```
/*@ normal_behavior
   @ requires b >= 2;
   @ ensures (\exists int r; 0 <= r && r < x;
             x == (\product int i; 0 <= i && i < r; b) && \result == r)
   @ || ( !(\exists int r; 0 <= r && r < x;
           x == (\product int i; 0 <= i && i < r; b)) && \result == -1);
   @ assignable \nothing;
   @*/
static int m(int x, int b) { ... }
```

Wenn `m` mit dem Parameter `b` größer oder gleich 2 und einem beliebigen `int x` aufgerufen wird, terminiert die Methode und ändert keine (vor Aufruf der Methode bereits existierenden) Heap-Stellen. Der Rückgabewert von `m` ist dann der Logarithmus zur Basis `b` von `x`, sofern dieser eine ganze Zahl ist, und ansonsten -1.

- b. Gegeben seien die folgenden Klassen, die einen gerichteten Graphen implementieren. Die Knoten (Node) sowie die Kanten (Edge) werden dabei jeweils in einem Array gespeichert, wobei jede Kante eine Referenz auf ihren Startknoten (`start`) und ihren Endknoten (`end`) enthält. Um z.B. eine Straßenkarte darzustellen, sei die Objektivariante des Graphen, dass es keine un erreichbaren "Inseln" gibt, dass also jeder Knoten von jedem anderen Knoten aus erreichbar ist.
- Für die Formalisierung der Erreichbarkeit steht die Methode `reachable(Node a, Node b, int k)` zur Verfügung, die genau dann `true` zurückgeben soll, wenn der Knoten `b` von Knoten `a` aus in **genau** `k` Schritten erreichbar ist. Formulieren Sie diese Nachbedingung der Methode `reachable`.
  - Formulieren Sie die oben gegebene Invariante.

**Hinweis:** Es ist erlaubt, in der `ensures`-Klausel die Methode `reachable` rekursiv zu benutzen, da sie mit dem Schlüsselwort `pure` markiert ist.

```
class Node { ... }

class Edge {
    Node start;
    Node end;
}

class DirectedGraph {
    Node[] nodes;
    Edge[] edges;

    /*@ invariant (\forall int i; 0 <= i < nodes.length;
    @             (\forall int j; 0 <= j < nodes.length;
    @             i != j ==> (\exists int k; 0 <= k < edges.length;
    @             reachable(nodes[i], nodes[j], k)))));
    @*/

    /*@ normal_behavior
    @ requires k >= 0;
    @ requires (\exists int i; 0 <= i < nodes.length; nodes[i] == a);
    @ requires (\exists int i; 0 <= i < nodes.length; nodes[i] == b);
    @ ensures \result <==>
    @         (k == 0 && a == b)
    @         || (k != 0
    @             && (\exists int n; 0 <= n < nodes.length;
    @                 reachable(a, nodes[n], k - 1)
    @                 && (\exists int e; 0 <= e < edges.length;
    @                     edges[e].start == nodes[n] && edges[e].end == b));
    @*/
    static boolean /*@ pure @*/ reachable(Node a, Node b, int k) { ... };
}
```

## 7 Lineare Temporale Logik (LTL) und Büchautomaten

**((1,5 + 1,5 + 1,5) + 2,5 = 7 Punkte)**

- a. Im folgenden soll eine Auktion mit zwei Bietern  $A$  und  $B$  mittels LTL formalisiert werden. Die beiden Bieter können unter gewissen Bedingungen Gebote abgeben oder die Auktion beenden. Gegeben ist folgende LTL-Signatur:

$$\Sigma = \{bidA, bidB, close\}$$

Die Variablen sollen ausdrücken, ob eine Aktion zu einem Zeitpunkt ausgeführt wird. Es gilt also:  $bidA$  ist wahr gdw.  $A$  ein Gebot abgibt.  $bidB$  ist wahr gdw.  $B$  ein Gebot abgibt.  $close$  ist wahr gdw. die Auktion beendet wird.

Formalisieren Sie:

- i. Keine zwei Aktionen können gleichzeitig ausgeführt werden.

$$\boxed{\square(\neg((bidA \wedge bidB) \vee (bidA \wedge close) \vee (bidB \wedge close)))}$$

- ii. Die Auktion kann erst geschlossen werden, wenn mindestens zwei Zeitschritte lang kein Gebot abgegeben wurde.

$$\boxed{\square((bidA \vee bidB) \Rightarrow \neg(\mathbf{X}close \vee \mathbf{XX}close))}$$

- iii. Bieter  $A$  darf keine zwei Gebote hintereinander abgeben, d.h. zwischen zwei Geboten von  $A$  muss immer ein Gebot von  $B$  liegen.

$$\boxed{\square(bidA \Rightarrow \mathbf{X}(\neg bidA \mathbf{U}_w bidB))}$$

- b. Geben Sie einen nicht-deterministischen Büchi-Automaten an, dessen akzeptierte Sprache den Modellen ( $\omega$ -Wörtern) der LTL-Formel

$$\square b \wedge \square \diamond a$$

über der Signatur  $\Sigma = \{a, b\}$  entspricht.

Für das Vokabular  $V = \mathbb{P}(\Sigma)$  (Potenzmenge von  $\Sigma$ ) werden die folgenden, aus der Vorlesung bekannten, Abkürzungen definiert:

$$A = \{M \in V \mid a \in M\} \subset V$$

$$B = \{M \in V \mid b \in M\} \subset V$$

$$\bar{A} = \{M \in V \mid a \notin M\} \subset V$$

$$\bar{B} = \{M \in V \mid b \notin M\} \subset V$$

$$AB = A \cap B$$

$$\bar{A}\bar{B} = \bar{A} \cap \bar{B}$$

$$A\bar{B} = A \cap \bar{B}$$

$$\bar{A}B = \bar{A} \cap B$$

