Humboldt-Universität zu Berlin
Lehrstuhl Logik in der Informatik
Prof. Dr. Nicole Schweikardt

19. Januar 2016

Logik in der Informatik

Wintersemester 2015/2016

Übungsblatt 12

Abgabe: *bis 28. Januar, 13.15 Uhr (vor der Vorlesung oder im Briefkasten zwischen den Räumen 3.401 und 3.402 im Johann von Neumann-Haus (Rudower Chaussee 25))*

Aufgabe 1: (24 Punkte)

Seien $\varphi, \psi \in \mathsf{FO}[\sigma]$. Leiten Sie ähnlich wie in Beispiel 4.19 aus dem Skript die folgenden beiden Sequenzen im Sequenzenkalkül $\mathfrak{K}_S$ ab.

(a) $\varphi,\ (\neg\varphi \vee \psi) \ \vdash\ \psi$

(b) $\varphi \ \vdash\ \neg\neg\varphi$

Aufgabe 2: (24 Punkte)

Sei $\sigma := \{E\}$ die Signatur, die aus einem 2-stelligen Relationssymbol E besteht.

(a) Zeigen Sie, dass die Klasse aller azyklischen (endlichen oder unendlichen) Graphen erststufig axiomatisierbar ist.

(b) Nutzen Sie den Endlichkeitssatz der Logik erster Stufe, um zu zeigen, dass die Klasse aller *nicht* azyklischen (endlichen oder unendlichen) Graphen *nicht* erststufig axiomatisierbar ist.

Zur Erinnerung: Ein gerichteter Graph ist azyklisch, falls er keinen Kreis endlicher Länge besitzt.

Aufgabe 3: **(27 Punkte)**

Wir betrachten im Folgenden Kalküle über der Menge $M := \mathsf{AL}(\{\neg, \rightarrow\})$.

Ein Kalkül $\mathfrak{K}$ über M heißt *korrekt*, falls für jede Menge $\Phi \subseteq M$ und jede Formel $\psi \in M$ gilt: Wenn $\psi \in \mathrm{abl}_{\mathfrak{K}}(\Phi)$, dann gilt $\Phi \models \psi$. Ein Kalkül $\mathfrak{K}$ über M heißt *vollständig*, falls für jede Menge $\Phi \subseteq M$ und jede Formel $\psi \in M$ gilt: Wenn $\Phi \models \psi$, dann ist $\psi \in \mathrm{abl}_{\mathfrak{K}}(\Phi)$.

Seien $\mathfrak{K}_{Syl}$ und $\mathfrak{K}_{Abd}$ die beiden folgenden Kalküle über der Menge M: Beide Kalküle enthalten für jede *allgemeingültige* aussagenlogische Formel $\varphi \in M$ das Axiom $\dfrac{}{\varphi}$.

Außerdem enthält

- $\mathfrak{K}_{Abd}$ für alle $\varphi, \psi \in M$ die Ableitungsregel

$$\frac{\psi \qquad (\varphi \rightarrow \psi)}{\varphi},$$

  die *Abduktion* genannt wird,

- $\mathfrak{K}_{Syl}$ für alle $\varphi, \psi, \chi \in M$ die Ableitungsregel

$$\frac{(\varphi \rightarrow \psi) \qquad (\psi \rightarrow \chi)}{(\varphi \rightarrow \chi)},$$

  die *Syllogismus* genannt wird.

(a) Geben Sie für $\varphi := \neg(A_0 \rightarrow A_0)$ und $\Phi := \emptyset$ eine möglichst kurze Ableitung von φ aus Φ in $\mathfrak{K}_{Abd}$ an.

(b) Beweisen Sie, dass $\mathfrak{K}_{Abd}$ vollständig, aber nicht korrekt ist.

(c) Beweisen Sie, dass $\mathfrak{K}_{Syl}$ korrekt, aber nicht vollständig ist.

(d) Betrachten Sie den Kalkül $\mathfrak{K}$ über M, der alle Ableitungsregeln aus $\mathfrak{K}_{Syl}$ und alle Ableitungsregeln aus $\mathfrak{K}_{Abd}$ enthält. Ist $\mathfrak{K}$ korrekt bzw. vollständig? Begründen Sie Ihre Antwort.

Weitere Aufgaben finden Sie auf der Rückseite

Aufgabe 4: (25 Punkte)

Lesen Sie Kapitel 8 aus dem Buch „Learn Prolog Now!“.

Achtung: *Geben Sie Ihre Lösungsansätze für die Teilaufgaben (a)–(c) in einer Datei mit dem Namen* `blatt12.pl` *über das GOYA-System ab!* ***Es gilt:*** *Lösungsansätze, die von SWI-Prolog nicht geladen werden können, werden nicht bewertet!*

(a) Implementieren Sie ein Prädikat `sat/1`, so dass eine Anfrage `?- sat(F).` für eine aussagenlogische Formel `F` genau dann erfolgreich ist, wenn `F` erfüllbar ist.

Hinweise: Ihr Prädikat soll zu der Formel `F` zuerst eine *erfüllbarkeitsäquivalente* 3-*KNF* konstruieren, und anschließend deren Erfüllbarkeit mit dem DPLL-Algorithmus testen. Es macht nichts, wenn Ihr Prädikat für eine erfüllbare Formel mehrfrach `true.` ausgibt.

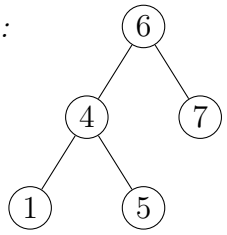
Sie können zur Lösung dieser Aufgabe alle Prolog-Module verwenden, die Sie unter `http://www2.informatik.hu-berlin.de/logik/lehre/WS15-16/Logik/downloads/al/` vorfinden. Dies gilt insbesondere für die Module `tseitin.pl` und `dpll.pl`.

In den folgenden Teilaufgaben betrachten wir *geordnete und beschriftete Binärbäume*, die folgende Eigenschaften erfüllen:

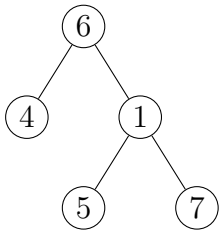
- Jeder Binärbaum enthält mindestens einen Knoten.
- Jeder Knoten in einem Binärbaum ist entweder ein *Blattknoten*, d.h. er besitzt keine Kinderknoten, oder er ist ein *innerer Knoten*, d.h. er besitzt sowohl einen linken als auch einen rechten Teilbaum, die beide nicht leer sind.
- Jeder Knoten in einem Binärbaum ist mit einem Prolog-Term beschriftet.

Betrachten Sie beispielsweise folgende Binärbäume:

Binärbaum 1:



Binärbaum 2:



Wir repräsentieren Binärbäume wie folgt durch Prolog-Terme: Ist `T` ein Prolog-Term, dann entspricht `node(T, nil, nil)` dem Binärbaum, der aus genau einem, mit `T` beschrifteten, Blattknoten besteht. Repräsentieren `L` und `R` Binärbäume, dann ist `node(T, L, R)` der Binärbaum, der aus einem mit `T` beschrifteten inneren Knoten besteht, an den als linker Teilbaum `L` und als rechter Teilbaum `R` angehängt ist. Beispielsweise wird *Binärbaum 1* repräsentiert durch den Term

```
node(6, node(4, node(1, nil, nil), node(5, nil, nil)), node(7, nil, nil))
```

(b) Eine Liste `X` nennen wir die *pre-order-Traversierung* eines Binärbaums `B`, wenn das folgende Prädikat `preorder/2` die Anfrage `?- preorder(X, B).` erfüllt:

```
preorder(node(T, nil, nil), [T]) :- !.
preorder(node(T, L, R), [T|X]) :-
    preorder(L, LX), preorder(R, RX), append(LX, RX, X).
```

Obwohl jeder Binärbaum eine eindeutige pre-order-Traversierung besitzt, gibt es umgekehrt Listen, die pre-order-Traversierungen von keinem oder auch von mehr als einem Binärbaum sind. Beispielsweise ist die Liste `[6, 4, 1, 5, 7]` sowohl für *Binärbaum 1* als auch für *Binärbaum 2* eine pre-order-Traversierung.

Schreiben Sie eine *Definite Clause Grammar mit einem zusätzlichen Argument*, so dass die Anfrage `?- bt(B, X, []).` für einen Binärbaum `B` und eine Liste `X` genau dann erfüllt ist, wenn `X` eine pre-order-Traversierung ist.

(c) Wir nennen einen Binärbaum einen *binären Suchbaum*, wenn seine Knoten nur mit Ganzzahlen beschriftet sind und für jeden seiner inneren Knoten `node(Z, L, R)` gilt: Jeder Knoten in `L` ist mit einer Zahl $<$ `Z`, und jeder Knoten in `R` mit einer Zahl $>$ `Z` beschriftet. Beispielsweise ist *Binärbaum 1* ein binärer Suchbaum, *Binärbaum 2* jedoch nicht.

Schreiben Sie eine *Definite Clause Grammar mit drei zusätzlichen Argumenten*, so dass die Anfrage `?- bst(B, Min, Max, X, []).` genau dann erfüllt ist, wenn `X` die pre-order-Traversierung für den *binären Suchbaum* `B` ist, und zusätzlich `Min` und `Max` die kleinste, beziehungsweise größte Ganzzahl ist, die als Beschriftung in `B` vorkommt.

Hinweis: Nutzen Sie die Möglichkeit, Ihrer Definite Clause Grammar mit Hilfe der Notation {...} zusätzliche Ziele hinzuzufügen. Verwenden Sie gegebenfalls das eingebaute Prädikat `integer/1`.