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

15. Dezember 2016

Logik in der Informatik

Wintersemester 2016/17

Übungsblatt 9

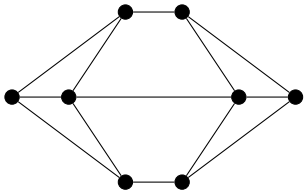
Abgabe: *10. Januar 2017*

Aufgabe 1: (25 Punkte)

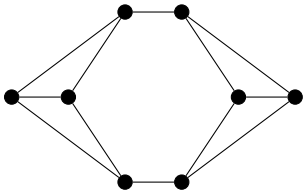
Sei $\sigma := \{E/2\}$. In der folgenden Darstellung von Graphen repräsentiert jede ungerichtete Kante zwischen zwei Knoten u und v die beiden gerichteten Kanten (u, v) und (v, u).

Betrachten Sie die folgenden Graphen $\mathcal{A}$ und $\mathcal{B}$:

$\mathcal{A}$:



$\mathcal{B}$:



Sei $\varphi := \exists x \, \exists y \, \forall z \, (E(x, z) \vee E(y, z))$. Dann gilt: $\mathcal{A} \models \varphi$ und $\mathcal{B} \not\models \varphi$.

(a) Leiten Sie aus dem $\mathsf{FO}[\sigma]$-Satz φ eine Gewinnstrategie für Spoiler im EF-Spiel auf $\mathcal{A}$ und $\mathcal{B}$ her. Geben Sie an, wie viele Runden Spoiler benötigt, wenn er dieser Strategie folgt. Beschreiben Sie die Strategie ähnlich wie in der in der Vorlesung behandelten Beweisidee zu Satz 3.52.

(b) Existiert eine bessere Gewinnstrategie für Spoiler, d.h. eine Strategie, mit der er in weniger Runden das Spiel gewinnt?

Aufgabe 2: (25 Punkte)

Beweisen Sie folgende Aussage:
Für alle $m \in \mathbb{N}$, alle relationalen Signaturen σ, alle σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$,
alle $k \in \mathbb{N}$, alle $\overline{a} := a_1, \ldots, a_k \in A$ und alle $\overline{b} := b_1, \ldots, b_k \in B$ gilt:

> Genau einer der beiden Spieler hat eine Gewinnstrategie im m-Runden EF-Spiel auf $(\mathcal{A}, \overline{a})$ und $(\mathcal{B}, \overline{b})$.

Hinweis: Per Induktion nach m ist der Beweis einfach und kurz.

Aufgabe 3: (25 Punkte)

(a) Sei $\sigma := \{f, R, S, c\}$ eine Signatur mit einem 1-stelligen Funktionssymbol f, einem 2-stelligen Relationssymbol R, einem 3-stelligen Relationssymbol S und einem Konstantensymbol c.

Überprüfen Sie für jedes der folgenden Worte, ob es sich jeweils um eine aussagenlogische Formel in AL, um einen σ-Term, um eine atomare σ-Formel und/oder um eine FO[σ]-Formel gemäß der Definitionen 2.4, 3.11 und 3.15 aus dem Skript handelt. Begründen Sie gegebenenfalls, warum ein Wort keine Formel in AL, keinen σ-Term, keine atomare σ-Formel bzw. keine FO[σ]-Formel darstellt. Für jedes der Worte, das eine FO[σ]-Formel ist, geben Sie bitte außerdem auch die Menge aller freien Variablen an und entscheiden Sie, ob das Wort ein FO[σ]-Satz ist.

(i) $(A_1 \wedge A_2)$

(ii) $(v_1 \wedge v_2)$

(iii) $f(f(v_2))$

(iv) $f^{\mathcal{A}}(f^{\mathcal{A}}(\beta(v_2)))$

(v) $R(f(v_2), v_3)$

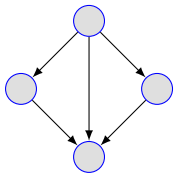
(vi) $R^{\mathcal{A}}(f^{\mathcal{A}}(\beta(v_2)), \beta(v_3))$

(vii) $\exists v_7 \neg f(f(f(v_7)))=f(v_7 \vee v_7)$

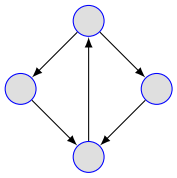
(viii) $\exists v_2 \forall v_3 \forall v_2 (f(v_2)=v_1 \rightarrow \forall v_2 (S(f(v_1), v_3, v_5) \wedge R(v_2, v_3)))$

(b) Sei nun $\sigma := \{E\}$ die Signatur, die aus einem 2-stelligen Relationssymbol E besteht. Betrachten Sie die σ-Strukturen $\mathcal{A} = (A, E^{\mathcal{A}})$ und $\mathcal{B} = (B, E^{\mathcal{B}})$, die durch die beiden gerichteten Graphen in der unten angegebenen Abbildung repräsentiert werden.
Geben Sie einen FO[σ]-Satz φ an, für den gilt: $\mathcal{A} \models \varphi$ und $\mathcal{B} \models \neg\varphi$.
Begründen Sie, warum Ihr φ von $\mathcal{A}$ erfüllt wird, aber nicht von $\mathcal{B}$.

$\mathcal{A}$:



$\mathcal{B}$:



— *auf der nächsten Seite geht's weiter* —

Aufgabe 4: **(25 Punkte)**

Lesen Sie Kapitel 11 aus dem Buch „Learn Prolog Now!“.

Achtung: *Geben Sie Ihre Lösungsansätze in einer Datei* `blatt09.pl` *über Moodle ab!*
Außerdem gilt: *Lösungsansätze, die von SWI-Prolog nicht geladen werden können, werden nicht bewertet!*

Speichern Sie die Dateien `al.pl` und `kinodb.pl`, die Sie auf der Webseite unter der URL `https://www2.informatik.hu-berlin.de/logik/lehre/WS16-17/Logik/#prolog-uebung` finden, und beginnen Sie die Datei `blatt09.pl` mit der Zeile:

```
:- ensure_loaded([kinodb, al]).
```

(a) Schreiben Sie ein Prädikat `george/1`, so dass die Anfrage

```
?- george(R).
```

in der Liste `R` die Menge aller Tupel `(K, Z)` zurückgibt, so dass gilt: Zum Zeitpunkt `Z` läuft im Kino `K` ein Film, in dem George Clooney mitspielt.

(b) Wir implementieren eine Reservierungsverwaltung für das Kino Babylon (in dem zur Zeit aus technischen Gründen nur ein Saal in Betrieb ist). Der Umstand, dass eine Person `P` für den Film, der um `Z` Uhr beginnt, Sitzplatz `S` reserviert hat, soll durch einen Fakt `belegt(P, Z, S)` in der Wissensbasis ausgedrückt werden. Stellen Sie zu diesem Zweck Ihrer Datei `blatt09.pl` die Zeile

```
:- dynamic belegt/3.
```

voran. Schreiben Sie ein Prädikat `reservieren/3`, so dass die Anfrage

```
?- reservieren(P, Z, S).
```

den Sitzplatz `S` für Person `P` und `Z` Uhr reserviert, d.h., der Wissensbasis einen Fakt `belegt(P, Z, S)` hinzufügt. Dies soll allerdings nur möglich sein, wenn um `Z` Uhr im Babylon tatsächlich ein Film läuft, und wenn der Sitzplatz für diese Uhrzeit noch nicht belegt ist. Anderenfalls soll die Anfrage scheitern.

(c) Schreiben Sie ein Prädikat `stornieren/2`, so dass die Anfrage

```
?- stornieren(Z, S).
```

die Reservierung für den Sitzplatz `S` zur Zeit `Z` aufhebt, d.h., einen entsprechenden Fakt in der Wissensbasis löscht. Gibt es eine solche Reservierung nicht, so soll die Anfrage scheitern.

(d) Wir repräsentieren im Folgenden Klauseln als Listen von Literalen und Klauselmengen als Listen von Klauseln. Beispielsweise repräsentiert der Prolog-Term

```
[[~x1, ~x2, ~x3, x4], [x1, ~x2], [x2]]
```

die Klauselmenge $\{\{\neg X_1, \neg X_2, \neg X_3, X_4\}, \{X_1, \neg X_2\}, \{X_2\}\}$.
Schreiben Sie ein Prädikat `unit_propagation/2`, das die Vereinfachungsheuristik *Unit Propagation* des DPLL-Algorithmus implementiert. D.h., ist `K` eine Klauselmenge, dann sollte die Anfrage

```
?- unit_propagation(K, K2).
```

in `K2` die Klauselmenge zurückgeben, die aus `K` entsteht, indem die Unit Propagation so lange auf `K` angewendet wird, bis keine „Einerklauseln“ mehr vorhanden sind. Beispielsweise sollte die Anfrage

```
?- unit_propagation([[~x1, ~x2, ~x3, x4], [x1, ~x2], [x2]], K2).
```

zu der folgenden (oder einer äquivalenten) Antwort führen:

```
K2 = [[~x3, x4]].
```

Hinweis: Führen Sie geeignete Hilfsprädikate ein.