Humboldt-Universität zu Berlin
Lehrstuhl Logik in der Informatik
Dr. Christoph Berkholz

8. Januar 2019

Logik in der Informatik

Wintersemester 2018/2019

Übungsblatt 10

Abgabe: *bis 15. Januar 2019, 11.¹⁵ 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: (25 Punkte)

(a) Sei $\Sigma = \{a, b\}$ und $\sigma_\Sigma = \{\leqslant, P_a, P_b\}$ die Signatur, die aus dem 2-stelligen Relationssymbol $\leqslant$, sowie zwei 1-stelligen Relationssymbolen P_a und P_b besteht. Beweisen Sie, dass es keinen $\mathsf{FO}[\sigma_\Sigma]$-Satz gibt, der die Sprache aller nicht-leeren Worte aus $\{a, b\}^*$ beschreibt, in denen die Anzahl der in ihnen vorkommenden as gerade ist.

Zur Erinnerung: Ein $\mathsf{FO}[\sigma_\Sigma]$-Satz φ *beschreibt* eine Sprache $L \subseteq \Sigma^*$, falls für jedes nicht-leere Wort $w \in \Sigma^*$ gilt: $w \in L \iff \mathcal{A}_w \models \varphi$.

(b) Sei 2-*COL* die Klasse aller gerichteten zweifärbbaren Graphen, d.h. aller $\{E/2\}$-Strukturen $\mathcal{A} = (A, E^{\mathcal{A}})$ für die gilt:

> Es gibt eine Funktion $f : A \to \{rot, blau\}$, so dass für jede Kante (a, b) in $E^{\mathcal{A}}$ gilt: $f(a) \neq f(b)$.

Zeigen Sie: Die Klasse 2-*COL* ist *nicht FO-definierbar*.

Aufgabe 2: (25 Punkte)

(a) Sei die Signatur $\sigma := \{E, f\}$. Hierbei ist E ein zweistelliges Relationssymbol und f ein einstelliges Funktionssymbol. Welche der folgenden Aussagen sind korrekt, welche nicht? (Sie brauchen Ihre Antwort nicht zu begründen.)

(i) $\forall x\, \exists y\ E(x, y) \quad \equiv \quad \exists y\, \forall x\ E(x, y)$

(ii) $\forall x\, \forall y \left(f(x) = y \ \rightarrow \ E(y, x) \right) \quad \equiv \quad \forall y\, \forall x \left(\neg\, E(y, x) \ \rightarrow \ \neg\, f(x) = y \right)$

(iii) $\forall x\, \forall y\ \neg\, f(x) = y \quad \equiv \quad \neg \forall x\, \exists y \left(\left(x = y \ \vee \ E(x, y) \right) \ \rightarrow \ \exists z \left(z = y \ \vee \ E(z, y) \right) \right)$

(b) Welche der folgenden Aussagen sind für alle Signaturen σ und alle $\mathsf{FO}[\sigma]$-Formeln φ und ψ korrekt, welche nicht? (Sie brauchen Ihre Antwort nicht zu begründen.)

(i) $(\forall x\, \varphi \ \vee \ \forall x\, \psi) \ \equiv \ \forall x\, (\varphi \vee \psi)$

(ii) $(\forall x\, \varphi \ \wedge \ \forall x\, \psi) \ \equiv \ \forall x\, (\varphi \wedge \psi)$

(iii) $(\forall x\, \varphi \ \rightarrow \ \exists x\, \psi) \ \equiv \ \exists x\, (\varphi \rightarrow \psi)$

(iv) $(\forall x\, \varphi \ \rightarrow \ \exists x\, \psi) \ \equiv \ \forall x\, (\varphi \rightarrow \psi)$

(c) Beweisen Sie, dass Ihre Antworten zu **(i)** und **(iii)** in Aufgabenteil **(b)** korrekt sind.

Aufgabe 3: (25 Punkte)

(a) Welche der beiden folgenden Aussagen ist für jede Signatur σ und jede $\mathsf{FO}[\sigma]$-Formel φ korrekt, welche nicht? Beweisen Sie, dass ihre Antworten korrekt sind.

(i) $\exists x \forall y \varphi \models \forall y \exists x \varphi$ **(ii)** $\forall y \exists x \varphi \models \exists x \forall y \varphi$

(b) Sei $\sigma = \{E/2\}$. Betrachten Sie die $\mathsf{FO}[\sigma]$-Formel

$$\varphi(x,z) := \exists y \Big(E(z,y) \rightarrow \big(\forall y\, E(x,y) \wedge \neg \exists x\, E(x,y) \big) \Big)$$

(i) Berechnen Sie eine zu φ äquivalente $\mathsf{FO}[\sigma]$-Formel in Negationsnormalform.

(ii) Berechnen Sie eine zu φ äquivalente $\mathsf{FO}[\sigma]$-Formel in Pränex-Normalform.

Gehen Sie hierbei wie in Beispiel 3.71 vor. Machen Sie pro Zwischenschritt nur eine Umformung und kommentieren Sie Ihre Zwischenschritte.

(c) Beweisen Sie Satz 3.68 aus der Vorlesung, das heißt zeigen Sie:

Jede $\mathsf{FO}[\sigma]$-Formel φ ist äquivalent zu einer Formel in NNF.

Aufgabe 4: (25 Punkte)

Lesen Sie Kapitel 11 aus dem Buch „Learn Prolog Now!“.
Achtung: *Die Bearbeitung der Aufgabe ist unter Beachtung der bekannten Abgabehinweise über Moodle abzugeben! Importieren Sie in Ihrer Abgabedatei* `blatt10.pl`*, analog zu Blatt 7, die Dateien* `al.pl` *und* `kinodb.pl`*.*

(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 für den George Clooney Regie geführt hat.

(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. So repräsentiert `[[~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.