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

17. Januar 2025

Logik in der Informatik

Wintersemester 2024/2025

Übungsblatt 12

Abgabe: *bis 27. Januar 2025, 13.00 Uhr*

Aufgabe 1: (Moodle-Quiz)

Absolvieren Sie das Quiz 12 auf der Moodle-Plattform.

Aufgabe 2: (Präsenzaufgabe)

(a) Sei $\sigma := \{ E \}$ die Signatur, die aus dem 2-stelligen Relationssymbol E besteht.

- **(i)** Zeigen Sie, dass die Klasse aller azyklischen (endlichen oder unendlichen) Graphen erststufig axiomatisierbar ist.
- **(ii)** 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.

Hinweis: Ein gerichteter Graph ist azyklisch, falls es kein $\ell \in \mathbb{N}_{\geqslant 1}$ und keine Knoten $a_1, \ldots, a_\ell$ gibt, sodass (a_ℓ, a_1) und (a_i, a_{i+1}) für alle $i \in [\ell - 1]$ Kanten im Graphen bilden.

(b) Sei $\sigma := \{R, f_0, f_1, c\}$, wobei c ein Konstantensymbol, R ein 2-stelliges Relationssymbol und f_0, f_1 zwei 1-stellige Funktionssymbole sind.

Beweisen Sie folgende Aussagen aus Korollar 4.41 aus dem Vorlesungsskript:

- **(i)** Das Unerfüllbarkeitsproblem für $\mathsf{FO}[\sigma]$ ist nicht entscheidbar.
- ~~**(ii)** Das Erfüllbarkeitsproblem für $\mathsf{FO}[\sigma]$ ist nicht semi-entscheidbar.~~

Aufgabe 3: (40 Punkte)

(a) Sei $\sigma := \{ E \}$ die Signatur, die aus dem 2-stelligen Relationsymbol E besteht und seien $\varphi, \psi \in \mathsf{FO}[\sigma]$. Seien $x, y \in \mathsf{VAR}$ zwei verschiedene Variablen. Leiten Sie ähnlich wie in Beispiel 4.19 aus dem Skript die folgenden beiden Sequenzen im Sequenzenkalkül $\mathfrak{K}_S$ ab.

- **(i)** $\varphi \vdash \neg\neg\varphi$
- **(ii)** $\varphi, (\neg\varphi \vee \psi) \vdash \psi$

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

(b) Sei $\sigma := \{ E, f \}$ die Signatur, die aus dem 2-stelligen Relationssymbol E und dem 1-stelligen Funktionssymbol f besteht. Betrachten Sie das Alphabet $A := A_{\mathsf{FO}[\sigma]}$ und die Menge $M := A^*$.

Geben Sie einen Kalkül $\mathfrak{K}$ über der Menge M an, sodass gilt: $\mathrm{abl}_{\mathfrak{K}} = \mathsf{FO}[\sigma]$. Das heißt, $\mathrm{abl}_{\mathfrak{K}}$ soll aus genau denjenigen Elementen von M bestehen, die gemäß Definition 3.15 syntaktisch korrekte Formeln der Logik erster Stufe über der Signatur σ sind.

(c) Sei $\Sigma := \{ a, b \}$ und $\sigma_\Sigma := \{ \leqslant, P_a, P_b \}$ die Signatur mit dem 2-stelligen Relationssymbol $\leqslant$ und den zwei 1-stelligen Relationssymbolen P_a und P_b.

Definition: Eine Sprache $L \subseteq \Sigma^*$ ist *erststufig axiomatisierbar*, falls es eine (möglicherweise unendliche) Menge Φ von $\mathsf{FO}[\sigma_\Sigma]$-Sätzen gibt, sodass für alle nicht-leeren Worte $w \in \Sigma^*$ gilt:

$$w \in L \quad \Longleftrightarrow \quad \mathcal{A}_w \models \Phi$$

Zur Erinnerung: $\mathcal{A}_w$ ist die zu w gehörende Wortstruktur.

(i) Zeigen Sie, dass die Sprache $L_1 \subseteq \Sigma^*$, die durch den regulären Ausdruck $(ab)^*$ beschrieben wird, erststufig axiomatisierbar ist.

(ii) Sei für ein Wort $u \in \Sigma^*$ die Sprache $L(u)$ definiert als die Menge $\{ w \in \Sigma^* \ : \ w \neq u \}$. Zeigen Sie, dass die Sprache $L(u)$ für jedes $u \in \Sigma^*$ erststufig axiomatisierbar ist.

(iii) Nutzen Sie **(ii)** um zu zeigen, dass die Sprache $L_2 \subseteq \Sigma^*$, die durch den regulären Ausdruck $(aa)^*$ beschrieben wird, erststufig axiomatisierbar ist.
Beachten Sie, dass L_2 *nicht* FO-definierbar ist.

(iv) Zeigen Sie mithilfe von **(ii)**, dass *jede* Sprache $L \subseteq \Sigma^*$ erststufig axiomatisierbar ist.

Aufgabe 4: (20 Punkte)

Lesen Sie Kapitel 12 aus dem Buch „Learn Prolog Now!“.

Achtung: *Die Bearbeitung der Aufgabe ist unter Beachtung der bekannten Abgabehinweise über Moodle abzugeben! Analog zu früheren Blättern finden Sie die benötigten Dateien auf der Seite zur Prolog-Übung.*

(a) Machen Sie sich mit den Prolog-Modulen `al_def`, `al_literals` und `al_nf` vertraut, welche Sie auf der Seite zur Prolog-Übung finden können. Laden Sie diese Prolog-Module in ein Verzeichnis Ihrer Wahl.

(b) Erstellen Sie (im selben Verzeichnis) in einer Datei `blatt12.pl` ein Modul mit dem Namen `pure_literal`, das die Prädikate `knf_shell/0` und `pure_literal/2` exportiert.

(c) Importieren Sie im Modul `pure_literal` genau die Prädikate aus den Modulen `al_def`, `al_literals` und `al_nf`, die Sie zum Lösen der folgenden beiden Teilaufgaben benötigen.

(d) Wir kodieren Klauselmengen wie auf Blatt 11 als Listen von Listen von Literalen.

Implementieren Sie das Prädikat `knf_shell/0`, so dass eine Anfrage

```
?- knf_shell.
```

eine Eingabeaufforderung zur Konstruktion von Klauselmengen aus aussagenlogischen Formeln startet.

— auf der nächsten Seite geht's weiter —

D.h., wenn über die Tastatur eine aussagenlogische Formel als Prolog-Term eingegeben wird, dann soll nach Ende der Eingabe (durch `.` und die Taste „Enter“) eine zu der Formel äquivalente Klauselmenge ausgegeben werden. Dies soll so lange wiederholt werden, bis statt einer aussagenlogischen Formel das Atom `bye` (wieder gefolgt durch `.` und die Taste „Enter“) eingegeben wird.

Hinweise: Definieren Sie sich gegebenenfalls geeignete Hilfsprädikate. Verwenden Sie für die Eingabe das Prädikat `read/1` und für die Ausgabe das Prädikat `write/1`. Beide Prädikate sind in SWI-Prolog vordefiniert. Sie müssen sich nicht um die Behandlung von Eingabefehlern kümmern.

(e) *Zur Erinnerung: Pure Literal Rule*

> Literale λ, deren Negat $\overline{\lambda}$ nirgendwo in der Klauselmenge auftaucht, können auf 1 gesetzt werden. Alle Klauseln, die ein solches Literal enthalten, sind dann wahr und können gestrichen werden. Wiederhole dies, so lange es Literale gibt, deren Negat nirgendwo in der Klauselmenge auftaucht.

Implementieren Sie ein Prädikat `pure_literal/2`, so dass eine Anfrage von der Form

```
?- pure_literal(KM, KM2).
```

auf die Klauselmenge `KM` die *Pure Literal Rule* des DPLL-Algorithmus solange anwendet, wie es Literale gibt, deren Negat nirgendwo in der Klauselmenge auftaucht und die entstehende Klauselmenge in `KM2` zurückgibt. Beispielsweise sollte die Anfrage

```
?- pure_literal([[~x1, x2, ~x5], [x1, x2, ~x4, x7], [x3, ~x5, x7],
                 [x3, ~x4, ~x5], [x5,x4,~x8], [x1,x3,x5,x7],
                 [~x7,x8]], KM2).
```

zu der Antwort

```
KM2 = [].
```

führen.

Hinweise: Definieren Sie geeignete Hilfsprädikate. *Beispielsweise* bietet es sich an, Prädikate `is_literal/2` und `is_pure_literal/2` einzuführen, so dass das Ziel `is_literal(L, KM)` für jedes in der Klauselmenge `KM` vorkommende Literal `L` erfüllt ist, und so dass das Ziel `is_pure_literal(L, KM)` für jedes in der Klauselmenge `KM` vorkommende Literal `L` erfüllt ist, dessen Negat nicht in `KM` vorkommt.