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

5. Januar 2016

Logik in der Informatik

Wintersemester 2015/2016

Übungsblatt 10

Abgabe: *bis 14. 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)

Beweisen oder widerlegen Sie die folgenden Aussagen:

(a) Sei $\sigma = \emptyset$. Es gibt einen $\mathsf{FO}[\sigma]$-Satz φ_a, so dass für jede σ-Struktur $\mathcal{A}$ gilt:

$$\mathcal{A} \models \varphi_a \quad \Longleftrightarrow \quad |A| \text{ ist eine Primzahl.}$$

(b) Sei $\sigma = \{E/2, F/2\}$ eine relationale Signatur. Es gibt einen $\mathsf{FO}[\sigma]$-Satz φ_b, so dass für jede σ-Struktur $\mathcal{A} = (A, E^{\mathcal{A}}, F^{\mathcal{A}})$, bei der $\mathcal{G} := (A, E^{\mathcal{A}})$ ein endlicher gerichteter Pfad ist, gilt:

$$\mathcal{A} \models \varphi_b \quad \Longleftrightarrow \quad F^{\mathcal{A}} \text{ ist der reflexive und transitive Abschluss von } E^{\mathcal{A}}.$$

Dabei nutzen wir folgende Begriffe:

- Ein Graph $\mathcal{G} = (V^{\mathcal{G}}, E^{\mathcal{G}})$ ist ein endlicher gerichteter Pfad, falls es ein $n \in \mathbb{N}$ mit $n \geq 1$ gibt, so dass $|V^{\mathcal{G}}| = n$, $V^{\mathcal{G}} = \{v_1, \ldots, v_n\}$ und $E^{\mathcal{G}} = \{\, (v_i, v_{i+1}) \,:\, 1 \leq i < n \,\}$.
- Seien $E^{\mathcal{A}}, F^{\mathcal{A}} \subseteq A \times A$. $F^{\mathcal{A}}$ heißt transitiver und reflexiver Abschluss von $E^{\mathcal{A}}$, wenn für alle $(a, a') \in A \times A$ gilt:

$$(a, a') \in F^{\mathcal{A}} \quad \Longleftrightarrow \quad a = a' \text{ oder es gibt im gerichteten Graphen } \mathcal{G} = (A, E^{\mathcal{A}}) \text{ einen Weg vom Knoten } a \text{ zum Knoten } a'.$$

Aufgabe 2: (24 Punkte)

Beweisen Sie folgende Aussage:
Für alle relationalen Signaturen σ, alle σ-Strukturen $\mathcal{A}$ und $\mathcal{B}$,
alle $k \in \mathbb{N}$, alle $\overline{a} := a_1, \ldots, a_k \in A$, alle $\overline{b} := b_1, \ldots, b_k \in B$ und alle $m \in \mathbb{N}$ 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: **(27 Punkte)**

Katzen äußern sich bekanntlich mit Hilfe der Laute „M“, „I“ und „U“.

Die *Katzensprache* K ist eine Menge von Worten über dem Alphabet $A := \{M, I, U\}$, die durch die folgenden Regeln rekursiv definiert ist:

Basisregel: (B) $MI \in K$.

Rekursive Regeln: Für alle $v \in A^*$ und alle $w \in A^*$ gilt:

(R1) Ist $vI \in K$, so ist auch $vIU \in K$.

(R2) Ist $Mv \in K$, so ist auch $Mvv \in K$.

(R3) Ist $vIIIw \in K$, so ist auch $vUw \in K$.

(R4) Ist $vUUw \in K$, so ist auch $vw \in K$.

(a) Geben Sie für jedes der folgenden Worte aus A^* an, ob es zur Menge K gehört oder nicht. Begründen Sie jeweils Ihre Antwort!

(i) MIU

(ii) UMII

(iii) MUII

(iv) MU

(b) Beweisen Sie mit einer Induktion über den Aufbau der Menge K, dass für jedes Wort $w \in K$ gilt: Die Anzahl $|w|_I$ der Vorkommen des Lauts I in w ist *nicht* durch 3 teilbar (d.h. es gibt eine Zahl $k \in \mathbb{N}$, so dass gilt: $|w|_I = 3k + 1$ oder $|w|_I = 3k + 2$).

(c) Kann eine Katze „MUUU“ machen? D.h., ist $MUUU \in K$?

(d) Geben Sie einen Kalkül $\mathfrak{K}$ über der Menge A^* an, so dass gilt: $\text{abl}_{\mathfrak{K}} = K$.

Aufgabe 4 finden Sie auf der Rückseite

Aufgabe 4: (25 Punkte)

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

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

(a) Machen Sie sich mit den Prolog-Modulen `al_def`, `al_literals` und `al_nf` vertraut, die Sie unter der URL

`http://www2.informatik.hu-berlin.de/logik/lehre/WS15-16/Logik/downloads/al/`

finden können. Laden Sie diese Prolog-Module in ein Verzeichnis Ihrer Wahl.

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

(c) Importieren Sie im Modul `pure_literal` die Prädikate aus den Prolog-Modulen `al_def`, `al_literals` und `al_nf`, die Sie für die Lösung der folgenden beiden Teilaufgaben benötigen.

(d) Wir kodieren Klauselmengen wie auf Blatt 9 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. 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) 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 anwendet 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 sich geeignete Hilfsprädikate. *Beispielsweise* bietet es sich an, Prädikate `literal/2` und `pure_literal/2` zu definieren, so dass `literal(L, KM)` für jedes in der Klauselmenge `KM` vorkommende Literal `L` erfüllt ist, und `pure_literal(L, KM)` für jedes in der Klauselmenge `KM` vorkommende Literal `L` erfüllt ist, dessen Negat nicht in `KM` vorkommt.