Prof. Dr. A. Poetzsch-Heffter
Peter Zeller, M. Sc.

TU Kaiserslautern
Fachbereich Informatik
AG Softwaretechnik

Sheet 14: Logik (SS 2017)

Abgabe: Montag, 17. Juli, 14:00
Abgabekästen neben Raum 34-401.7 (bei AG Softwaretechnik)
Bitte geben Sie zu dritt ab.

With this sheet you can get additional points for the admission to the final exam. For the admission you only need 50% of the points up until exercise sheet 12, so 46 (out of 92) points or more.

For exercise 1-3 you only have to hand in part a) to get the full points for the exercise. You can do the remaining tasks on your own as preparation for the exam.

Please note that the deadline is already on Monday this time, so that we can discuss the sheet in the last exercise sessions this semester.

Aufgabe 1 Tableaux

Use the Tableaux method for first-order predicate logic to show that:

a) $\Big(\big(\forall x.\forall y.\ \neg\neg p(x,y) \to p(y,x)\big) \wedge \forall x.\exists y.\ p(x,y)\Big) \to \forall x.\exists y.\ p(y,x)$ is a tautology.

b) $\neg\Big(\big(\forall x.\ p(x) \to p\big(f(x)\big)\big) \to \big(\forall x.\ p(x) \to p\big(f\big(f(x)\big)\big)\big)\Big)$ is unsatisfiable.

c) $\neg\Big(\big(\big(\exists x.\ p(x) \wedge q(x)\big) \to \big(\big(\exists x.\ p(x)\big) \wedge \big(\exists y.\ q(y)\big)\big)\big) \wedge \big(\big(\big(\exists x.\ p(x)\big) \wedge \big(\forall y.\ q(y)\big)\big) \to \big(\exists z.\ p(z) \wedge q(z)\big)\big)\Big)$
is unsatisfiable.

Aufgabe 2 Berechnung von MGUs

Consider the following sets of literals and decide for each set if it is unifiable. If it is unifiable, give a most general unifier (MGU). Use the algorithm from the lecture step by step. For each loop iteration give two literals $L_k\Theta$ und $L_m\Theta$ and mark the position in which they differ.

In this task x, y, z_1, z_2, z_3, z_4 are variables, p, q are predicate-symbols, and f, g, h, a, b function-symbols.

a)
$$\Big\{q\big(f(a,x),z_1\big),\ q\big(f\big(y,g(z_1)\big),h(z_3)\big),\ q\big(z_2,h(b)\big)\Big\}$$

b)
$$\Big\{p\big(x,f(y)\big),\ p\big(f(a),y\big)\Big\}$$

c)
$$\Big\{p\big(x,\ f(x),\ h(y)\big),\ p\big(g(z_1,a),\ y,\ z_2\big),\ p\big(g(b,z_3),\ f(z_4),\ h\big(f(z_4)\big)\big)\Big\}$$

Aufgabe 3 Resolution

Use the resolution-method (of first-order predicate logic) to show that the following formulas are unsatisfiable. Start by transforming the formulas into a suitable form.

a)

$$\exists z.\forall x.\forall y.\ \Big(p(y) \to q(x,x)\Big) \wedge \Big(q(x,y) \to r(y)\Big) \wedge p(z) \wedge \Big(\neg r(z) \vee \neg p(z)\Big)$$

b)

$$\neg\Bigg(\Big(\forall y.\ q(y)\Big) \vee \neg\Bigg(\forall x.\Big(q(x) \vee r(x)\Big) \wedge \Big(\exists z.\ \neg p(z) \wedge \Big(p(z) \vee \neg r(x)\Big)\Big)\Bigg)\Bigg)$$

Aufgabe 4 Logisches Programmieren mit Prolog

Das Programmieren mit Prolog wird nicht Teil der Abschlussklausur sein. Mit dieser Aufgabe können Sie aber auch allgemein das Modellieren von Aussagen mit Logik üben.

Installieren Sie zunächst die Prolog-Variante SWI-Prolog (http://www.swi-prolog.org/). Auf den Terminal-Rechnern des SCI ist diese bereits für Sie installiert.

Erstellen Sie dann eine Datei `lists.pl` mit folgendem Inhalt:

```
% ein Element X ist Element der Liste, die X als erstes Element hat
elem(X, [X|Xs]).
% wenn X ein Element der Liste L ist, dann ist X auch
% ein Element der Liste mit einem weiteren Element Y am Anfang vor L
elem(X, [Y|L]) :- elem(X, L).
```

Die Zeilen, die mit einem Prozent-Zeichen starten sind Kommentare und können ausgelassen werden. Laden Sie dann Die Datei in den Prolog-Interpreter, indem Sie folgenden Befehl auf der Konsole ausführen:

```
swipl lists.pl
```

Im Interpreter können Sie dann die in der Datei definierten Prädikate mit Abfragen testen, zum Beispiel:

```
?- elem(1, [1,2,3]).
true .

?- elem(77, [1,2,3]).
false.
```

Abfragen können Variablen verwenden. Falls es mehrere mögliche Ergebnisse gibt, kann durch drücken von "n" das nächste mögliche Ergebnis angezeigt werden.

```
?- elem(X, [1,2,3]).
X = 1 ;
X = 2 ;
X = 3 ;
false.
```

Hinweise zu Prolog und zur verwendeten Syntax:

- Variablen beginnen mit einem Großbuchstaben.
- Der Unterstrich wird als Platzhalter-Variable verwendet, falls eine Variable sonst nirgendwo verwendet wird.
- Prädikate und Funktionssymbole beginnen mit einem Kleinbuchstaben.
- In einer Datei steht eine Liste von Aussagen, die jeweils mit einem Punkt enden. Die Aussagen sind entweder atomar oder eine Implikation der Form `A :- B1, ..., Bn`, was der Generalisierung der logischen Formel $(B_1 \wedge \cdots \wedge B_n) \to A$ entspricht.

- Prolog unterstützt Negation, wir werden diese aber hier nicht verwenden, da die Negation in Prolog gewisse Unterschiede zur Negation in der Logik hat.
- Mit dem Befehl "`make.`" kann die Prolog-Datei im Prolog Interpreter neu geladen werden.
- Es gibt in Prolog zwei Funktionssymbole um Listen aufzubauen. Die leere Liste wird als `[]` geschrieben und die Liste mit erstem Element `X` und Rest-Liste `Xs` wird als `[X|Xs]` geschrieben.

  Um die Notation zu vereinfachen gibt es außerdem zwei abkürzende Schreibweisen: Statt `[X|[Y|Ys]]` können die ersten Elemente auch zu `[X,Y|Ys]` zusammengefasst werden. Wenn der Rest der Liste die leere Liste ist, kann diese auch weggelassen werden. Die folgenden 3 Notationen bezeichnen also alle den gleichen Term: `[1|[2|[3|[]]]]`, `[1,2,3|[]]` und `[1,2,3]`.

Prolog bietet einige eingebaute Prädikate auf Listen. Wir wollen in dieser Aufgabe zum Üben unsere eigenen Prädikate definieren. Falls Sie eine Teilaufgabe nicht lösen können, dürfen Sie in nachfolgenden Teilaufgaben auch auf die entsprechenden eingebauten Prädikate zurückgreifen (siehe http://www.swi-prolog.org/pldoc/man?section=lists).

a) Definieren Sie ein Prädikat `app(X,Y,Z)`, welches wahr ist, wenn die Liste `Z` durch das zusammenfügen der Listen `X` und `Y` entstanden ist.

Hinweis: Verwenden Sie zwei Aussagen:
1. Die leere Liste vorne anfügen ergibt die gleiche Liste.
2. Wenn `Xs` angefügt an `Ys` die Liste `Zs` ergibt, dann ergibt `[X|Xs]` angefügt an `Ys` die Liste `[X|Zs]`.

Beispiele:

```
?- app([1,2,3],[4,5],[1,2,3,4,5]).
true.

?- app([a,b],[c,d,e],X).
X = [a, b, c, d, e].
```

b) Definieren Sie ein Prädikat `start(X,Y)`, das wahr ist, wenn die Liste `Y` mit der Liste `X` anfängt.

Hinweis: Wenn es eine Liste `R` gibt, so dass `R` angehängt hinter `X` die Liste `Y` ergibt, dann gilt `start(X,Y)`.

Beispiele:

```
?- start([], [1,2,3]).
true.

?- start([1,2], [1,2,3]).
true.

?- start([2,3], [1,2,3]).
false.

?- start(X, [1,2,3]).
X = [] ;
X = [1] ;
X = [1, 2] ;
X = [1, 2, 3] ;
false.
```

c) Definieren Sie ein Prädikat `mid(X,Y)`, das wahr ist, wenn die Liste `Y` die Liste `X` als Teilliste enthält.

Beispiele:

```
?- mid([],[1,2,3]).
true .

?- mid([1,2],[1,2,3,4]).
true .

?- mid([2,3],[1,2,3,4]).
```

```
true .

?- mid([1,3],[1,2,3,4]).
false.
```

d) Definieren Sie ein Prädikat `rem(X,L1,L2)`, welches aussagt, dass `L2` die Liste ist, die man durch Entfernen eines Vorkommens von `X` aus `L1` erhalten kann.

Hinweis: Unterscheiden Sie die Fälle `rem(X,[X|Ys],...)` und `rem(X,[Y|Ys],...)` und formulieren Sie jeweils Bedingungen für den dritten Parameter.

Beispiele:

```
?- rem(2,[1,2,1,3],X).
X = [1, 1, 3] ;
false.

?- rem(1,[1,2,1,3],X).
X = [2, 1, 3] ;
X = [1, 2, 3] ;
false.

?- rem(4,[1,2,3],X).
false.
```

e) Definieren Sie ein Prädikat `remFirst(X,L1,L2)`, welches aussagt, dass `L2` die Liste ist, die man durch Entfernen **des ersten** Vorkommens von `X` aus `L1` erhalten kann.

Verwenden Sie hierzu das eingebaute Prädikat `dif(X, Y)`, welches aussagt, dass `X` und `Y` unterschiedliche Terme sind.

Beispiel:

```
?- remFirst(2,[1,2,3,2],X).
X = [1, 3, 2] ;
false.

?- remFirst(3,[1,2,3],X).
X = [1, 2] ;
false.

?- remFirst(2,X,[1,2,3]).
X = [2, 1, 2, 3] ;
X = [1, 2, 2, 3] ;
false.
```

f) Definieren Sie ein Prädikat `perm(L1,L2)`, welches aussagt, dass die Liste `L1` eine Permutation der Liste `L2` ist, d.h. dass die zwei Listen die gleichen Elemente enthalten, aber eventuell in verschiedener Reihenfolge.

Hinweis: Die leere Liste ist eine Permutation der leeren Liste und die Liste `[X|Xs]` ist eine Permutation einer Liste `Ys` wenn `Xs` eine Permutation von `Ys` ohne das erste `X` ist.

Beispiel:

```
?- perm([1,2,3],[2,3,1]).
true .

?- perm([1,2,3],X).
X = [1, 2, 3] ;
X = [2, 1, 3] ;
X = [2, 3, 1] ;
X = [1, 3, 2] ;
X = [3, 1, 2] ;
X = [3, 2, 1] ;
false.
```

g) Schreiben Sie ein Prädikat `sorted(L)`, welches aussagt, dass die Liste `L` aufsteigend sortiert ist. Verwenden Sie dazu das eingebaute Prädikat `X =< Y` für den Vergleich von Listen-Einträgen.

Beispiel:

```
?- sorted([]).
true.

?- sorted([1,2,3]).
true .

?- sorted([1,3,2]).
false.
```

h) Verwenden Sie die Prädikate `sorted` und `perm`, um ein Prädikat `slowSort(L1,L2)` zu definieren. Das Prädikat soll aussagen, dass `L2` der Liste `L1` in aufsteigend sortierter Reihenfolge entspricht. Das heißt `L2` muss sortiert und eine Permutation von `L1` sein.

Beispiel:

```
?- slowSort([2,3,1],X).
X = [1, 2, 3] ;
false.

?- slowSort([2,4,3,1],X).
X = [1, 2, 3, 4] ;
false.
```

i) Schreiben Sie ein Prädikat `insertionSort(L1,L2)`. Dieses soll immer das gleiche Ergebnis wie `slowSort` liefern, allerdings der Strategie “Sortieren durch Einfügen” folgen. Definieren Sie sich dazu zuerst ein Hilfs-Prädikat für das sortierte Einfügen in eine Liste.