SS2010
BAI2-LBP Gruppe 1 Team 07
Lösung zu Aufgabe 4

R. C. Ladiges, D. Fast

16. Juni 2010

Inhaltsverzeichnis

4 Aufgabe 4

In dieser Aufgabe wird mit einem in Prolog implementierten Beweiser (Prädikatenlogik) gearbeitet. Diese Aufgabe soll ermöglichen, sich mit Beweisverfahren in der Logik intensiver auseinander zu setzen und damit die Trennung von Syntax und Semantik wie auch den Aufbau logischer Systeme im Allgemeinen besser zu verstehen. Die Implementierung wie auch die Aufgabe sind im Original von Carola Eschenbach und Rüdiger Valk an der Universität Hamburg erstellt worden.

Für die Aufgabe ist das Paket prädikatenlogik.zip zu speichern.

4.1 Sich mit dem Programmpaket vertraut machen

4.1.1 Aufgabenstellung

Machen Sie sich mit dem Programmpaket vertraut. Lesen Sie die Definitionen der Prädikate durch, Probieren Sie die Prädikate aus und überzeugen Sie sich, dass Sie wissen, warum was passiert.

4.1.2 Entwurf

Ansehen aller Prädikate der vier Module und falls möglich / sinnvoll in den sechs anderen Aufgaben verwenden.

Zur Lösung:
Auflistung aller Prädikate mit einem Beschreibungstext in eigenen Worten. Hiermit ist nicht die ausführliche Beschreibung der Funktionsweise der Prädikate gemeint.

4.1.3 Lösung

main.pl :
`test_F/2`:
Prüft, ob eine Formel bei einem bestimmten Modell erfüllt, oder nicht erfüllt ist.
`test/2`:
Prüft eine bestimmte Beispielformel aus `exampleModels.pl` mit `test_F/2`.
`testAll/0`:
Prüft alle Beispielformeln aus `exampleModels.pl` mit `test_F/2`

formulae.pl :
`wff/1`:
Prüft ob die Eingabe eine korrekte Formel ist.
`wfts/1`:
Prüft ob die Eingabe eine korrekte Termliste ist.
`wft/1`:
Prüft ob die Eingabe ein gültiger Term ist.
`aussagensymbole_L/2`:
Liefert eine Liste aller Aussagensymbole einer Formel, ohne doppeltes Vorkommen derer.
`freieVariablen_F/2`:
Liefert eine Liste aller freien Variablen einer Formel, ohne doppeltes Vorkommen derer.

`freieVariablen_TL/2`:
Liefert eine Liste aller freien Variablen einer Termliste, ohne doppeltes Vorkommen derer.
`freieVariablen_T/2`:
Liefert eine Liste aller freien Variablen (max. ein Element) eines Terms.

modelChecker.pl :
`satisfy/3`:
Prüft eine Formel mit `evaluateF/4` auf den Wahrheitswert Wahr.
`dissatisfy/3`:
Prüft eine Formel mit `evaluateF/4` auf den Wahrheitswert Falsch.
`varBelegung/3`:
Belegt die freien Variablen mit Elementen des Universums (alle Möglichkeiten).
`asBelegung/2`:
Erzeugt alle möglichen Belegung einer eingehenden Liste von Aussagensymbolen als getrennte Ergebnisse.
`evaluateF/4`:
Dieses Prädikat ist Wahr, wenn eine eingegebene Formel bei angegebenem Modell und Variablenbelegung zum übergebenen Wahrheitswert evaluiert.
`evaluateF_all/5`:
Unterprädikat für den Allquantor, welches alle möglichen Variablenbelegungen für eine bestimmte Variable durchgeht.
`evaluateArgs/4`:
Wertet eine Termliste, zu einem bestimmten Modell, bei einer bestimmten Variablenbelegung, aus.
`evaluateArg/4`:
Wertet einen Term, zu einem bestimmten Modell, bei einer bestimmten Variablenbelegung, aus.

exampleModels.pl :
`constant/1`:
Dient zur Deklaration der in Formeln vorkommenden Konstanten.
`function/2`:
Dient zur Deklaration der in Formeln vorkommenden Funktionen.
`aussagensymbol/1`:
Dient zur Deklaration der in Formeln vorkommenden Aussagensymbolen.
`relation/2`:
Dient zur Deklaration der in Formeln vorkommenden Relationen.
`example_F/2`:
Wird für die Zuordnung von Beispielformeln zu Bezeichnern benutzt. Dadurch kann die Formel leichter an mehrere dafür vorgesehene Prädikate übergeben werden, in dem sie über den Bezeichner adressiert wird.
`domain/2`:
Wird für die Zuordnung von einem Universum zu einem Modellbezeichner benutzt.
`interpretation/2`:
Wird für die Zuordnung von einer Interpretation zu einem Modellbezeichner benutzt.

4.2 Anwendung des Paketes für bestimmte Fragestellungen

4.2.1 Aufgabenstellung

Ergänzen Sie Aussagensymbole und Beispielformeln (Datei: exampleModels.pl). (**Anwendung** des Paketes für bestimmte Fragestellungen, **erstellen von Modellen**)

4.2.2 Entwurf

Ergänzung des Prädikates `examples_F/2` in `exampleModels.pl` um weitere Fakten, welche als zweites Argument Formeln haben. Diese sind selbst ausgedachte simple Formeln, und/oder weitere komplexere Formeln aus der Vorlesung und/oder Literatur, die Quantoren, Variablen, Funktionen und Relationen in der Prädikatenlogik veranschaulichen. Um die Beispiele zu testen, und um Unterschiede zu zeigen, definieren wir uns eigene, dazu passende, Modelle (`domain/2` und `interpretation/2`).

4.2.3 Quelltext

```
constant(pa). %Person A
constant(pb). %Person B
constant(pc). %Person C

relation(liebt,2). %liebt(X,Y):= X liebt Y

%Es existiert jemand (>=1) der alle Liebt.
example_F(lf(1), exists(X, forall(Y, liebt(X,Y) ) ) ).

%Jeder hat jemanden (>=1) den er liebt.
example_F(lf(2), forall(X, exists(Y, liebt(X,Y) ) ) ).

%Es exisiert jemand (>=1) der von allen geliebt wird.
example_F(lf(3), exists(X, forall(Y, liebt(Y,X) ) ) ).

%Jeder wird von jemanden (>=1) geliebt.
example_F(lf(4), forall(X, exists(Y, liebt(Y,X) ) ) ).

%Jeder liebt jeden.
example_F(lf(5), forall(X, forall(Y, liebt(X,Y) ) ) ).


domain(lf(_), [a, b, c]).
interpretation(lf(1), [  % b wird von allen geliebt
   pa->a, pb->b, pc->c, liebt->[(a,b),(c,b),(b,b)]
   ]).
interpretation(lf(2), [  % b liebt alle
   pa->a, pb->b, pc->c, liebt->[(b,a),(b,b),(b,c)]
   ]).
interpretation(lf(3), [  % a und b lieben sich, und c liebt b
   pa->a, pb->b, pc->c, liebt->[(a,b),(b,a),(c,b)]
   ]).
```

4.3 Veränderung der Syntax und der Semantik

4.3.1 Aufgabenstellung

Verändern Sie die Sprache (**Veränderung der Syntax und der Semantik**): Ergänzen Sie die Sprache um mindestens einen Junktor (xor) und erzeugen Sie dazu Beispielformeln (Dateien: formulae.pl, modelChecker.pl, exampleModels.pl)

4.3.2 Entwurf

Für die Umsetzung der Kontravalenz können (weil in Prolog bereits deklariert) wir zuerst das *„Symbol“* des Junktors $\dot\vee$ am Anfang der drei Dateien, so wie die anderen Junktoren auch, mit dem Prädikat `op/1` als Operator **`xor`** definieren.

In `formulae.pl` ergänzen wir die Prädikate `wff/1`, `aussagensymbole_L/2` und `freieVariablen_F/2` um je einen weiteren Fall für das Auftreten des **`xor`** Operators. Was, bis auf den Operator, syntaktisch gleich mit den anderen Junktoren (mit Stelligkeit 2) ist.
Zusätzlich verändern wir den letzten Fall (atomare Formeln) für das Prädikat `freieVariablen_F/2` so, dass der Operator für die Kontravalenz zusammen mit den Operatoren für die anderen Junktoren ausgeschlossen wird.

In `modelChecker.pl` ergänzen wir das Prädikat `evaluateAsF/4` für den **`xor`** Operator um zwei Regeln. Die zwei Regeln bestimmen die Logik der Kontravalenz, durch die zwei Fälle, bei der sie zu Wahr evaluiert:
1. Fall: Wenn Â(A)=1 und Â(B)=0
2. Fall: Wenn Â(A)=0 und Â(B)=1
für die Formel: `A` **`xor`** `B`
Der einzige Unterschied hierbei zur Aussagenlogik ist das weiterreichen des Modells.

In `exampleModels.pl` fügen wir noch, wie in [4.2], einige Beispiele zur Kontravalenz hinzu.

4.3.3 Quelltext

formulae.pl:

```
wff(Formula1 xor Formula2):- !,
  wff(Formula1),
  wff(Formula2).

%Alles hier drunter ist neu im Vergleich zur AL

aussagensymbole_L(F1 xor F2, LIn-LOut) :-
  aussagensymbole_L(F1, LIn-VarMid),
  aussagensymbole_L(F2, VarMid-LOut).

freieVariablen_F(F1 xor F2, VarIn-VarOut) :-
  freieVariablen_F(F1, VarIn-VarMid),
  freieVariablen_F(F2, VarMid-VarOut).
```

```
/* atomare Formeln */
freieVariablen_F(Formula,  VIO):-
  Formula =..[Pred | ArgList],
  \+ member(Pred, [& , v , > , ~ , xor, exists, forall]),
  freieVariablen_TL(ArgList, VIO).
```

ModelChecker.pl:

```
evaluateF(Formula1 xor Formula2, Modell, VarBelegung, true):-
  evaluateF(Formula1, Modell, VarBelegung, true),
  evaluateF(Formula2, Modell, VarBelegung, false),
  !.
evaluateF(Formula1 xor Formula2, Modell, VarBelegung, true):-
  evaluateF(Formula1, Modell, VarBelegung, false),
  evaluateF(Formula2, Modell, VarBelegung, true),
  !.
%das weiterreichen des Modell ist anders als bei der AL,
%sonst ist alles identisch zur AL.
```

ExampleModels.pl:

```
example_F(lf(6), liebt(X,pb) xor liebt(pb,X) ).
% Interpretation -> Erfüllbarkeit
% lf(1) -> erfüllt
% lf(2) -> erfüllt
% lf(3) -> nicht erfüllt
```

4.4 Sammeln syntaktischer Informationen

4.4.1 Aufgabenstellung

Ergänzen Sie einfache rekursive Prädikate (**Sammeln syntaktischer Informationen**): zählen Sie die Junktoren einer Formel und bilden Sie eine Liste aller Teilformeln, die durch Quantoren gebildet werden.

4.4.2 Entwurf

Zählen der Junktoren: Wie bei Aufgabe [2.4], nur mit weiteren Regeln für Quantoren (Ausgabe: Rekursionsaufruf auf die Formel), Gleichheit (Ausgabe: 0) und Relationssymbole (Ausgabe: 0).

Liste aller Teilformeln: Junktoren wie bei Aufgabe [2.4], mit dem Unterschied, dass sie nur den Rest zurückgeben. Bei Aussagensymbolen, Gleichheit und Relationssymbolen wird eine leere Liste ausgegeben. Nur wenn die Formel ein Quantor ist, wird die Formel in die Ausgabeliste gegeben. Ein Rekursionaufruf auf die Formel des Quantors findet eventuelle weitere Quantorenteilformeln.

4.4.3 Quelltext

Zählen der Junktoren:

```
njunktoren(Var, _):-
  var(Var),
  !,
  fail.
njunktoren(F, N):-
  (F=(F1&F2);F=(F1 v F2);F=(F1>F2);F=(F1<>F2);F=(F1 xor F2)),
  !,
  njunktoren(F1,TmpA),
  njunktoren(F2,TmpB),
  N is 1 + TmpA + TmpB.
njunktoren(~ F, N):-
  !,
  njunktoren(F,Tmp),
  N is 1 + Tmp.
njunktoren(F, N):-                           %anders AL (neu)
  ( F=exists(X,Tmp) ; F=forall(X,Tmp) ), %anders AL (neu)
  var(X),                                    %anders AL (neu)
  !,                                         %anders AL (neu)
  njunktoren(Tmp,N).                         %anders AL (neu)
njunktoren(T1 = T2, 0):-                     %anders AL (neu)
  !,                                         %anders AL (neu)
  wft(T1),                                   %anders AL (neu)
  wft(T2).                                   %anders AL (neu)
njunktoren(Formula, 0):-                     %anders AL (neu)
  Formula =..[Pred | ArgList],               %anders AL (neu)
  relation(Pred, Arity),                     %anders AL (neu)
  length(ArgList, Arity),                    %anders AL (neu)
  !,                                         %anders AL (neu)
  wfts(ArgList).                             %anders AL (neu)
njunktoren(F, 0):-
  aussagensymbol(F),
  !.
njunktoren(F, _):-
  write('Ungueltige Formel:'),
  writeln(F),
  !,
  fail.
```

Liste aller Teilformeln:

```
teilformeln(Var, _):-
  var(Var),
  !,
  fail.
```

```
teilformeln(F, R):-                         %anders AL
  (F=(F1&F2);F=(F1 v F2);F=(F1>F2);F=(F1<>F2);F=(F1 xor F2)),
  !,
  teilformeln(F1,TmpA),
  teilformeln(F2,TmpB),
  append(TmpA,TmpB,R).
teilformeln(~ F, R):-                       %anders AL
  !,
  teilformeln(F,R).
teilformeln(F, [F|R]):-                     %anders AL (neu)
  ( F=exists(X,Tmp) ; F=forall(X,Tmp) ), %anders AL (neu)
  var(X),                                   %anders AL (neu)
  !,                                        %anders AL (neu)
  teilformeln(Tmp,R).                       %anders AL (neu)
teilformeln(T1 = T2, []):-                  %anders AL (neu)
  !,                                        %anders AL (neu)
  wft(T1),                                  %anders AL (neu)
  wft(T2).                                  %anders AL (neu)
teilformeln(Formula, []):-                  %anders AL (neu)
  Formula =..[Pred | ArgList],              %anders AL (neu)
  relation(Pred, Arity),                    %anders AL (neu)
  length(ArgList, Arity),                   %anders AL (neu)
  !,                                        %anders AL (neu)
  wfts(ArgList).                            %anders AL (neu)
teilformeln(F, []):-                        %anders AL
  aussagensymbol(F),
  !.
teilformeln(F, _):-
  write('Ungueltige Formel:'),
  writeln(F),
  !,
  fail.
```

4.5 Sammeln semantischer Informationen

4.5.1 Aufgabenstellung

Ergänzen Sie Prädikate zur Berechnung semantischer Eigenschaften und Relationen (**Sammeln semantischer Informationen**) unter Betrachtung der möglichen Belegungen: Implementieren Sie Prädikate zur Bestimmung der Äquivalenz von Formeln bei einer gegebenen Interpretation (`aequiv(Formel1,Formel2,Model)`), gilt, wenn beide Formeln bei der gleichen Belegung den gleichen Wahrheitsgehalt haben) und Prädikate zur Bestimmung der Folgerung (Implikation) von Formeln aus Formeln bei einer gegebenen Interpretation (`impl(Formel1,Formel2, Model)`), gilt nicht, wenn `Formel1` wahr und `Formel2` bei der gleichen Belegung falsch ist, sonst gilt sie immer). Beachten Sie: dazu müssen die freien Variablen alle möglichen Belegungen durchlaufen!

4.5.2 Entwurf

Wir definieren uns die beiden Prädikate welche Wahr werden, wenn die gegebenen Formeln mit dem jeweiligen Junktor verknüpft eine Tautologie ergibt. Dazu schreiben wir uns ein eigenes Prädikat `tautologie/2` für die Prädikatenlogik (ähnlich wie `test_F/2` aufgebaut). Dieses Prädikat ist Wahr, wenn alle Varianten der Variablenbelegung erfüllbar sind. Dazu nutzen wir das `evaluateF_all/5` Prädikat für Wahrheit, welches wir mit unserer verknüpften Formel, unserem Modell, der zum Modell passenden `domain/2`, und den `freieVariablen_F/2` der Formel füttern.

4.5.3 Quelltext

```
aequiv(F1,F2,M) :- tautologie(F1<>F2,M). %anders AL
impl(F1,F2,M) :- tautologie(F1>F2,M).    %anders AL

tautologie(F,M):-              %anders AL
  wff(F),                      %anders AL
  freieVariablen_F(F, []-VL),  %anders AL
  chngforall(F,VL,NF),         %anders AL
  evaluateF(NF, M, [], true).  %ähnlich AL
tautologie(F, _):-
  \+ wff(F),
  write('Der Ausdruck '),  writeln(F),
  write('ist fuer diese Version des Modell-'),
  write('Pruefers nicht evaluierbar ') .

chngforall(F,[],F).      %anders AL (neu)
chngforall(F,[K|R],NF):- %anders AL (neu)
  chngforall(F,R,Tmp),   %anders AL (neu)
  NF=(forall(K,Tmp)).    %anders AL (neu)
```

4.5.4 Anmerkungen

Da `evaluateF_all/5` leider nicht wie erwartet alle (sondern immer nur eine) freien Variablen auf alle möglichen Belegungen abbildet müssen wir dies manuell machen. Dazu ergänzen wir die Formel um einen Allquantor für jede freie Variable, rekursiv mit `chngforall/3`).

Anstatt die Formel mit Allquantoren zu ergänzen und als ganzes zu evaluieren, könnte man auch direkt bei jedem Rekursionsschritt `evaluateF_all/5` aufrufen, was im Prinzip ja auch beim evaluieren des Allquantors passiert.

4.6 Erweiterung der Verarbeitungsmöglichkeiten des Pakets

4.6.1 Aufgabenstellung

Ergänzen Sie Prädikate zur Bestimmung der **Äquivalenz** und **Folgerung** von Formeln aus Formellisten bei einer gegebenen Interpretation (**Erweiterung der Verarbeitungsmöglichkeiten** des Pakets). Testen Sie die Prädikate auch mit der leeren Liste. Eine Formelliste wird dabei als konjunktive Verknüpfung aller ihrer Formeln aufgefasst.

4.6.2 Entwurf

Wir definieren uns ein Unterprädikat, welches Rekursiv eine gegebene Liste aus Formeln verundet (Konjunktion aller Elemente zu einer Formel) und ausgibt.

Ob die zusammengebaute Formel mit der übergebenen Formel äquivalent ist überprüfen wir mit dem Prädikat `aequiv/3` aus [4.5].

Dass, aus der zusammengebauten Formel, die übergebene Formel folgt, fragen wir über das `impl/3` Prädikat aus [4.5] ab.

4.6.3 Lösung

Das Unterprädikat `alskonjunktion`

```
alskonjunktion([],_):-
  !,
  writeln('Keine korrekte Formelliste'),
  fail.
alskonjunktion([F],F):- !.
alskonjunktion([FMK|FMR],FMK & Tmp):-
  alskonjunktion(FMR,Tmp),
  !.
```

verwandelt unsere eingehende Liste in eine einzige Formel. Die Abbruchbedingung der Rekursion ist hier eine einzelne Formel in der Liste. Eine Abbruchbedingung für die leere Liste würde keinen Sinn ergeben, da es dafür keine logische Formel gibt. Die eingehende Liste muss also mindestens ein Element enthalten. Bei einer Liste aus mehreren Elementen verknüpfen wir die einzelnen Elemente der Liste rekursiv zu einer Formel mit dem **&** Operator.

Unsere verschiedenen Probleme sind mit `alskonjunktion` nun leicht zu lösen:

```
fmaequiv(FM,F):-
  alskonjunktion(FM,Tmp),
  aequiv(Tmp,F).
fmimpl(FM,F):-
  alskonjunktion(FM,Tmp),
  impl(Tmp,F).
```

4.7 Vergleich zur Aussagenlogik

4.7.1 Aufgabenstellung

Machen Sie in Ihrem Code als Kommentar deutlich, wie sich die Erweiterungen in diesem Programmpaket gegenüber den Erweiterungen aus dem Programmpaket in Aufgabe 2 unterscheiden.

4.7.2 Entwurf

Bei den Zeilen die sich unterscheiden einen Kommentar „`%anders AL`“ anhängen.

Informationen zur Signatur

	**Unterzeichner**	EMAILADDRESS=robin.ladiges@haw-hamburg.de, CN=Robin Christopher Ladiges
	Datum/Zeit	Sat Jun 26 23:57:42 CEST 2010
	Austeller-Zertifikat	CN=CAcert Class 3 Root, OU=http://www.CAcert.org, O=CAcert Inc.
	Serien-Nr.	44727
	Methode	urn:adobe.com:Adobe.PPKLite:adbe.pkcs7.sha1 (Adobe Signatur)