SS2010
BAI2-LBP Gruppe 1 Team 07
Lösung zu Aufgabe 2

R. C. Ladiges, D. Fast

28. April 2010

Inhaltsverzeichnis

2 Aufgabe 2

2.1 Sich mit dem Programmpaket vertraut machen

2.1.1 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.

2.1.2 Entwurf

Ansehen aller Prädikate der vier Module und falls möglich in den fünf 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.

2.1.3 Lösung

`test/1` in `main.pl`:
Prüft, ob eine Formel eine Tautologie, ein Widerspruch oder erfüllbar ist und gibt das Ergebnis auf der Konsole aus.

`test_tt/1` in `main.pl`:
Gibt die Wahrheitstabelle, einer eingegebenen Formel, bei allen möglichen Belegungen der Aussagensymbole, auf der Konsole aus. Zur Realisierung der Ausgabe der Wahrheitstabelle sind Hilfsprädikate für `test_tt` definiert:

- `ttHeader/2` zur Darstellung der Kopfzeile der Wahrheitstabelle.
- `ttBorder/2` zur Ausgabe der horizontalen Linie.
- `wline/1` als Hilfe für `ttBorder` um eine bestimmte Anzahl Bindestriche auszugeben.
- `ttLine/2` zur Ausgabe einzelner Zeilen der Wahrheitstabelle.

`wff/1` in `formulae.pl`:
Prüft ob die Eingabe eine korrekte Formel ist.

`aussagensymbole_L/2` in `formulae.pl`:
Liefert eine Liste aller Aussagensymbole einer Formel, ohne doppeltes Vorkommen derer.

`tautologie/1` in `modelCheckerAL.pl`:
Prädikat, dass Wahr ist, wenn die eingegebene Formel eine Tautologie[1] ist.

`widerspruch/1` in `modelCheckerAL.pl`:
Prädikat, dass Wahr ist, wenn die eingegebene Formel ein Widerspruch[2] ist (unerfüllbar).

[1] Tautologie:Eine Formel, die bei allen möglichen Belegungen der Aussagensymbole Wahr ist.
[2] Widerspruch: Eine Formel, die bei allen möglichen Belegungen der Aussagensymbole Falsch ist.

`erfuellbar/1` in `modelCheckerAL.pl`:
Ist Wahr, wenn die eingegebene Formel erfüllbar[3] ist.

`falsifizierbar/1` in `modelCheckerAL.pl`:
Ist Wahr, wenn die eingegebene Formel falsifizierbar[4] ist.

`kontingent/1` in `modelCheckerAL.pl`:
Prüft ob eine Formel sowohl erfüllbar als auch falsifizierbar ist.

`asBelegung/2` in `modelCheckerAL.pl`:
Erzeugt alle möglichen Belegung einer eingehenden Liste von Aussagensymbolen als getrennte Ergebnisse.

`evaluateAsF/3` in `modelCheckerAL.pl`:
Dieses Prädikat ist Wahr, wenn eine eingegebene Formel bei angegebener Belegung zum übergebenen Wahrheitswert evaluiert.

`aussagensymbol/1` in `examples.pl`:
Dient zur Deklaration der in Formeln vorkommenden Aussagensymbolen.

`example_F/2` in `examples.pl`:
Wird für die Zuordnung von Beispielformeln zu Bezeichnern benutzt. Dadurch kann die Formel leichter an mehrere (nur an `test` und `test_tt`) Prädikate übergeben werden, in dem sie über den Bezeichner adressiert wird.

2.2 Anwendung des Paketes für bestimmte Fragestellungen

2.2.1 Aufgabenstellung

Ergänzen Sie Aussagensymbole und Beispielformeln (Datei: examples.pl). (**Anwendung** des Paketes für bestimmte Fragestellungen)

2.2.2 Entwurf

Ergänzung des Prädikates `examples_F` in `examples.pl` um eigene Fakten, welche als zweites Argument Formeln haben. Diese sind selbst ausgedachte simple Formeln, sowie weitere komplexere Formeln aus der Vorlesung und/oder Literatur.

2.2.3 Lösung

Wir haben folgende Beispiele geschrieben

```
example_F(lf(1), a v b).
example_F(lf(2), a & b).
example_F(lf(3), ~a).
example_F(lf(4), a <> b).
% LfI S.38 A2.7a - Unerfüllbar? ja
example_F(lf(5), (~(a&b))&(c>b)&(((~a)&b)>(~c))&c ).
```

[3] Erfüllbar: Eine Formel, die bei mindestens einer möglichen Belegung der Aussagensymbole Wahr ist.
[4] Falsifizierbar: Eine Formel, die bei mindestens einer möglichen Belegung der Aussagensymbole Falsch ist.

```
% LfI S.39 A2.9a - Erfüllbar? nein
example_F(lf(6), ((b>a)v((~c)&d))&(b v a)&(~((c>d)v a)) ).
% LfI S.39 A2.9b - Tautologie? ja
example_F(lf(7), (a>b)v(a&(~(b v c)))v(a&c) ).
```

welche wir mit den Prädikaten `test` und `test_tt` aus `main.pl` testen können. Die Beispiele 5, 6 und 7 sind aus dem Buch „Logik für Informatiker“ von M. Kreuzer und S. Kühling erschienen bei „Pearson Studium“ im März 2006.

Betrachten wir Beispiel 6. Hier soll geprüft werden ob die Formel erfüllbar ist. `?-test(lf(6))` sagt uns, dass die Formel ein Widerspruch ist, also nicht erfüllbar sein kann. Um das Ergebnis zu verifizieren führen wir den Beweis per Hand:

Die Formel

$$F = ((M \Rightarrow A) \vee (\neg T \wedge H)) \wedge (M \vee A) \wedge (\neg((T \Rightarrow H) \vee A))$$

soll auf Erfüllbarkeit geprüft werden.

Dafür formen wir die Formel F zuerst in die KNF um und vereinfachen dann die Formel:

$F \equiv \neg A \wedge \neg H \wedge T \wedge (A \vee M) \wedge (A \vee \neg M \vee (H \wedge \neg T))$
$\equiv \neg A \wedge \neg H \wedge T \wedge M \wedge (\neg M \vee (H \wedge \neg T))$
$\equiv \neg A \wedge \neg H \wedge T \wedge M \wedge (\neg M \vee Falsch)$
$\equiv \neg A \wedge \neg H \wedge T \wedge M \wedge \neg M$
$\equiv \neg A \wedge \neg H \wedge T \wedge Falsch$
$\equiv Falsch$

Also ergibt die Formel bei jeder möglichen Belegung Falsch und ist damit ein Widerspruch.

2.3 Veränderung der Syntax und der Semantik

2.3.1 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, modelCheckerAL.pl, examples.pl)

2.3.2 Entwurf

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

In `formulae.pl` ergänzen wir das Prädikat `wff(Formel)` um einen weiteren Fall für das Auftreten des **`xor`** Operators. Was, bis auf den Operator, syntaktisch gleich mit den anderen Junktoren (mit Stelligkeit 2) ist.

In `modelCheckerAL.pl` ergänzen wir das Prädikat `evaluateAsF` 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`

In `examples.pl` fügen wir noch, wie in [2.2], einige Beispiele zur Kontravalenz hinzu.

2.3.3 Lösung

Zuerst erweitern wir das Prädikat `wff` in `formulae.pl` mit

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

damit eine Formel in der der Operator **xor** enthalten ist auch als richtige Formel erkannt werden kann.

Danach erweitern wir das Prädikat `evaluateAsF` in `modelCheckerAL.pl` mit

```
% Kontravalenz
evaluateAsF(Formula1 xor Formula2,AsBelegung, true):-
   evaluateAsF(Formula1, AsBelegung, true),
   evaluateAsF(Formula2, AsBelegung, false),
   !.
evaluateAsF(Formula1 xor Formula2,AsBelegung, true):-
   evaluateAsF(Formula1, AsBelegung, false),
   evaluateAsF(Formula2, AsBelegung, true),
   !.
```

damit bekannt ist, wann die Formel bei gegebener Belegung der Variablen Wahr ist. Dafür wird die Formel in zwei Teilformeln (`Formula1` und `Formula2`) aufgeteilt und mit der selben Belegung Rekursiv aufgerufen, um zu sehen ob sie so evaluieren wie es für die Kontravalenz erforderlich ist um Wahr zu sein.

In `examples.pl` fügen wir

```
example_F(lf(8),  a xor b ).
example_F(lf(9),  a&(a xor b) ).
example_F(lf(10), a xor b xor c ).
```

ein, was Beispiele für die folgenden simplen Formeln sind:

(1) $a \dot{\vee} b$

(2) $a \wedge (a \dot{\vee} b)$

(3) $(a \dot{\vee} b) \dot{\vee} c$

Mit dem Prädikat `test_tt` können wir uns zu diesen Beispielen die Wertetabellen ausgeben lassen und sehen, ob unsere Kontravalenz funktioniert wie erwartet.

b	a	(1)	(2)	c	(3)
1	1	0	0	1	1
1	0	1	0	1	0
0	1	1	1	1	0
0	0	0	0	1	1
1	1	0	0	0	0
1	0	1	0	0	1
0	1	1	1	0	1
0	0	0	0	0	0

2.4 Sammeln syntaktischer Informationen

2.4.1 Aufgabenstellung

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

2.4.2 Entwurf

Zählen der Junktoren:
Mit einem modifiziertem `wff` Prädikat, welches eine Ausgabe besitzt gehen wir die Formel rekursiv durch.
Literale sind der Rekursionsabbruch. Ein Aufruf des Prädikats mit einem Literal gibt 0 zurück.
Wenn wir einen Junktor mit der Stelligkeit 1 (Negation $\neg$) haben, addieren wir 1 auf den rekursiven Aufruf der Formel(ohne Junktor) und geben die Summe aus.
Bei Junktoren der Stelligkeit 2 (Konjunktion $\wedge$, Disjunktion $\vee$, Kontravalenz $\veebar$, Subjunktion $\rightarrow$ und Äquivalenz $\leftrightarrow$) geben wir die Summer der rekursiven Aufrufe der Formeln und 1 aus.

Liste aller Teilformeln:
Wie beim Zählen der Junktoren verwenden wir ein modifiziertes `wff` Prädikat mit Ausgabe.
Bei einem Literal wird nur es selbst in einer einelementigen Liste ausgegeben.
Bei einem Junktor wird die Formel selbst, zusammen mit der/den Ausgabe(n) des/der rekursiven Aufrufe(s), in die Ausgabeliste getan.

2.4.3 Quelltext

```
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, 0):- aussagensymbol(F), !.
njunktoren(F, _):- write('Ungueltige Formel:'), writeln(F), !, fail.
```

```
teilformeln(Var, _):- var(Var), !, fail.
teilformeln(F, [F|R]):- (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, [~ F|R]):- !, teilformeln(F,R).
teilformeln(F, [F]):- aussagensymbol(F), !.
teilformeln(F, _):- write('Ungueltige Formel:'), writeln(F), !, fail.
```

2.4.4 Bemerkung

Wir haben die kopierten Regeln aus `wff` etwas verändert, so dass wir nicht für jeden Junktor der Stelligkeit 2 eine eigene syntaktisch gleiche Regel brauchen. Dazu haben wir eine Disjunktion von verschiedenen „Junktorformeln“ die unsere Eingangsformel unifizieren kann.

2.5 Sammeln semantischer Informationen

2.5.1 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 (`aequiv(Formel1,Formel2)`, gilt, wenn beide Formeln bei der gleichen Belegung den gleichen Wahrheitsgehalt haben) und Prädikate zur Bestimmung der Folgerung (Implikation) von Formeln aus Formeln (`impl(Formel1,Formel2)`, gilt nicht, wenn `Formel1` wahr und `Formel2` bei der gleichen Belegung falsch ist, sonst gilt sie immer).

2.5.2 Entwurf

Wir definieren uns die beiden Prädikate welche wahr werden, wenn die gegebenen Formeln mit dem jeweiligen Junktor verknüpft eine Tautologie ergibt. Dazu benutzen wir das gegebene `tautologie(F)` Prädikat.

2.5.3 Lösung

```
% ?- aequiv( (a&b), ~(~a v ~b) ).  -> true
% ?- aequiv( (a>b)&(b>c), (a>c) ). -> false.
aequiv(FA, FB) :- tautologie(FA <> FB).

% ?- impl( (a&b), ~(~a v ~b) ).  -> true
% ?- impl( (a>b)&(b>c), (a>c) ). -> true.
impl(FA, FB) :- tautologie(FA > FB).
```

Die beiden Prädikate haben den gleichen Aufbau. Die Formeln FA und FB werden mit dem jeweiligen, zum Prädikat passendem und vom Paket zur Verfügung gestellten, Junktor verknüpft. Danach wird überprüft ob die verknüpfte Formel eine Tautologie ist.

Dass dies zur Lösung führt liegt daran, dass zwei Formeln äquivalent sind, wenn bei jeder möglichen Belegung die Verknüpfung per Äquivalenzjunktor Wahr ist. Selbiges gilt für die Implikation.

2.6 Erweiterung der Verarbeitungsmöglichkeiten des Pakets

2.6.1 Aufgabenstellung

Ergänzen Sie Prädikate zur Bestimmung der **Erfüllbarkeit von Formelmengen** (repräsentiert als Listen) und zur **Äquivalenz** und **Folgerung** von Formeln aus Formellisten (**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.

2.6.2 Entwurf

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

Mit dem Prädikat `erfuellbar` testen wir ob die neue Formel erfüllbar ist.

Ob die zusammengebaute Formel mit der übergebenen Formel äquivalent ist überprüfen wir mit dem Prädikat `aequiv` aus [2.5].

Dass, aus der zusammengebauten Formel, die übergebene Formel folgt, fragen wir über das `impl` Prädikat aus [2.5] ab.

2.6.3 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:

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

Informationen zur Signatur

	**Unterzeichner**	EMAILADDRESS=robin.ladiges@haw-hamburg.de, CN=Robin Christopher Ladiges
	Datum/Zeit	Sun Jun 27 00:03:43 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)