

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

R. C. Ladiges, D. Fast

10. Juni 2010

Inhaltsverzeichnis

4 Aufgabe 4	3
4.1 Sich mit dem Programmpaket vertraut machen	3
4.1.1 Aufgabenstellung	3
4.1.2 Entwurf	3
4.2 Anwendung des Paketes für bestimmte Fragestellungen	3
4.2.1 Aufgabenstellung	3
4.2.2 Entwurf	3
4.3 Veränderung der Syntax und der Semantik	3
4.3.1 Aufgabenstellung	3
4.3.2 Entwurf	4
4.4 Sammeln syntaktischer Informationen	4
4.4.1 Aufgabenstellung	4
4.4.2 Entwurf	4
4.5 Sammeln semantischer Informationen	4
4.5.1 Aufgabenstellung	4
4.5.2 Entwurf	5
4.6 Erweiterung der Verarbeitungsmöglichkeiten des Pakets	5
4.6.1 Aufgabenstellung	5
4.6.2 Entwurf	5
4.7 Vergleich zur Aussagenlogik	5
4.7.1 Aufgabenstellung	5
4.7.2 Entwurf	5

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.

4.2 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`).

4.3 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 $\hat{A}(A)=1$ und $\hat{A}(B)=0$

2. Fall: Wenn $\hat{A}(A)=0$ und $\hat{A}(B)=1$

für die Formel: $A \text{ xor } B$

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.

4.4 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 Rekursionsaufruf auf die Formel des Quantors findet eventuelle weitere Quantorteilformeln.

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.

4.6 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.

4.7 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 23:59:04 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)