INSTITUT FÜR INFORMATIK
Lehr- und Forschungseinheit für
Programmier- und Modellierungssprachen

Wagmüllerstraße 23 D–80538 München

Ludwig
Maximilians-
Universität
München

Ein Prolog-Werkzeugkasten zur Manipulation logischer Formeln

Samsamoddin Rajaei

Fortgeschrittenenpraktikum

Beginn der Arbeit:	1. August 1994
Abgabe der Arbeit:	27. Oktober 1994
Betreuer:	Dipl.-Inform. Slim Abdennadher
	Dr. Norbert Eisinger
	Dipl.-Inform. Tim Geisler

Inhaltsverzeichnis

1 Einleitung

Die Logikprogrammierung, die die Resolution als eine Kalkülregel verwendet, setzt voraus, daß die Formeln in einer bestimmten Normalform, nämlich in Klauselnormalform vorliegen. Eine Umformung bzw. Änderung einer logischen Formel zu einer Formel in Klauselnormalform, kann in aufeinanderfolgenden Schritten erfolgen, die nach und nach die nötigen Änderungen in der Formel vornehmen. Ziel dieser Arbeit ist, PROLOG Prädikate zur Verfügung zu stellen, die die logischen Formeln unabhängig von ihrer konkreten Syntax manipulieren und verschiedene Normalformen als Zwischenstufen zur Klauselnormalform herleiten. Das soll allerdings nicht heißen, daß diese (Zwischen-)Normalformen nicht auch für andere Zwecke verwendet werden können.

2 Beschreibung der Normalformen

2.1 Variablendisjunkte Formel

Ein Ausdruck ψ ist Variablendisjunkt, wenn alle seine durch Quantoren gebundene Variablen verschieden sind.
Wenn $\psi = Qx\psi_0$ eine Formel mit $Q \in \{\forall, \exists\}$ ist und y eine Variable, die nicht in ψ_0 vorkommt, dann ist $\psi = Qx\psi_0[x/y]$.
Durch systematisches Ausführen derartiger Variablenumbenennungen, wobei immer neue, also nicht schon benutzte Variablen verwendet werden, kann man zu jeder Formel eine äquivalente Formel in variablendisjunkter Form angeben. [4]

2.2 Elementar-Junktor-Form

Ein Ausdruck ist in Elementar-Junktor-Form, wenn er ausschließlich mit Hilfe der Verknüpfungen $\wedge$, $\vee$ und $\neg$ gebildet ist. Um die anderen Verknüpfungen (d.h. $\Longrightarrow$, $\Longleftarrow$, $\Longleftrightarrow$, und $\not\Longleftrightarrow$) aufzulösen, werden die Äquivalenzen der folgenden Tabelle als Transformationsregeln verwendet.

$\phi \Longrightarrow \psi$	$\rightleftharpoons$	$\neg\phi \vee \psi$
$\phi \Longleftarrow \psi$	$\rightleftharpoons$	$\phi \vee \neg\psi$
$\phi \Longleftrightarrow \psi$	$\rightleftharpoons$	$(\neg\phi \vee \psi) \wedge (\phi \vee \neg\psi)$ oder $(\phi \wedge \psi) \vee \neg(\phi \vee \psi)$
$\phi \not\Longleftrightarrow \psi$	$\rightleftharpoons$	$\neg(\phi \Longleftrightarrow \psi)$

Für die Auflösung von $\Longleftrightarrow$ gibt es zwei Möglichkeiten. Wenn nachher die konjunktive Normalform der Formel hergeleitet werden soll, ist eine Auflösung von $\phi \Longleftrightarrow \psi$ in

$$(\phi \wedge \psi) \vee \neg(\phi \vee \psi)$$

von Nachteil. Sie liefert nämlich die folgende Formel

$$(\phi \vee \neg\phi) \wedge (\phi \vee \neg\psi) \wedge (\psi \vee \neg\phi) \wedge (\psi \vee \neg\psi),$$

die zwei Tautologien enthält, die nicht immer sofort erkennbar und eliminierbar sind. Eine Auflösung auf andere Weise ist also günstiger. Wenn aber die ganze Äquivalenz negiert ist, ist eine Auflösung in $(\phi \wedge \psi) \vee \neg(\phi \vee \psi)$ günstiger, weil es (nach Distribution von Negation und Konjunktion)

$$(\neg\phi \vee \neg\psi) \wedge (\phi \vee \psi)$$

liefert, wobei eine Auflösung der Formel auf zweite Weise wieder Tautologien zurückliefern würde.
Wenn eine Äquivalenz sich also im Bereich einer ungeraden Anzahl von (expliziten oder impliziten) Negationen befindet, soll sie besser auf die zweite Weise aufgelöst werden, anderenfalls auf die erste Weise. (Siehe [1].)

2.3 Negationsform

Ein Ausdruck ist in Negationsform, wenn er in Elementar-Junktor-Form ist und außerdem alle seine Negationen unmittelbar vor atomaren Formeln stehen.
Wenn eine Formel sich schon in Elementar-Junktor-Form befindet, ist die Erstellung der Negationsform einfach. Dazu sind die folgenden Transformationen so lange auf eine Formel anzuwenden, bis die Negationen sich ausschließlich vor den atomaren Formeln befinden.

$\neg\neg\phi$	$\rightleftharpoons$	ϕ
$\neg(\phi \vee \psi)$	$\rightleftharpoons$	$\neg\phi \wedge \neg\psi$
$\neg(\phi \wedge \psi)$	$\rightleftharpoons$	$\neg\phi \vee \neg\psi$
$\neg\forall_x\phi$	$\rightleftharpoons$	$\exists_x\neg\phi$
$\neg\exists_x\phi$	$\rightleftharpoons$	$\forall_x\neg\phi$

2.4 Konjunktive Normalform

Ein Ausdruck ist in konjunktiver Normalform, wenn er quantorenfrei und eine Konjunktion von Disjunktionen aus atomaren und negierten atomaren Formeln ist.
Wenn eine Formel sich in Negationsform befindet und **keine Quantoren enthält**, kann man mit wiederholter Anwendung des Distributivgesetzes ihre konjunktive Normalform herleiten. Das kann man mit Induktion nach Aufbau der Formel zeigen:
Induktionsanfang: p ist ein Literal, dann ist p in konjunktiver Normalform.
Wenn ϕ und ψ in konjunktiver Normalform sind, ist auch $\phi \wedge \psi$ in konjunktiver Normalform.
Wenn ϕ und ψ in konjunktiver Normalform sind, dann kann man die Formel $\phi \vee \psi$ auf folgende Weise in konjunktive Normalform bringen. Die Formel ist in folgender Form:

$$\bigwedge_{i=1}^{n_i} \bigvee_{j=1}^{n_j} \phi_{ij} \vee \bigwedge_{k=1}^{n_k} \bigvee_{l=1}^{n_l} \psi_{kl}$$

mit $\phi = \bigwedge_{i=1}^{n_i} \bigvee_{j=1}^{n_j} \phi_{ij}$ und $\psi = \bigwedge_{k=1}^{n_k} \bigvee_{l=1}^{n_l} \psi_{kl}$. Eine Anwendung des (verallgemeinerten) Distributivgesetzes auf diese Formel liefert

$$\bigwedge_{k=1}^{n_k} \Big[(\bigwedge_{i=1}^{n_i} \bigvee_{j=1}^{n_j} \phi_{ij}) \vee \bigvee_{l=1}^{n_l} \psi_{kl} \Big].$$

Eine nochmalige Verwendung des Distributivgesetzes liefert

$$\bigwedge_{k=1}^{n_k} \bigwedge_{i=1}^{n_i} (\bigvee_{j=1}^{n_j} \phi_{ij} \vee \bigvee_{l=1}^{n_l} \psi_{kl}),$$

die offensichtlich in konjunktiver Normalform ist [3].
Dabei ist zu beachten, daß eventuell Disjunktionen der Form $\phi \vee p \vee \neg p$ mit atomarem p vorkommen können. Hier kann man den ganzen Ausdruck mit TRUE ersetzen, da diese eine Tautologie ist. Dann tritt wahrscheinlich der Fall $\psi \wedge TRUE$ auf, wo

man TRUE ohne weiteres weglassen kann. Ein Vorkommen der gleichen Konjunktionsteilformeln der Form $\bigwedge_{i=1}^{j} \psi$ mit $j > 1$ ist auch möglich, wobei man durchaus alle ψ bis auf eine weglassen kann, da $\bigwedge_{i=1}^{j} \psi$ und ψ die gleichen Wahrheitswerte haben. Eine Konjunktion der Form $\phi \wedge p \wedge \neg p$ kann dementsprechend gleich FALSE gesetzt werden[1].

2.5 Pränexe Normalform

Unter einem Ausdruck in pränexer Normalform versteht man einen Ausdruck ψ der Form

$$Q_1 x_1 \cdots Q_n x_n \psi_0,$$

wobei $Q_i \in \{\forall, \exists\}$ für $1 \leq i \leq n$ sowie ψ_0 quantorenfreie Formel sind. $Q_1 x_1 \cdots Q_n x_n$ ist der *Präfix* von ψ, und ψ_0 ist die *Matrix* von ψ.
Um die pränexe Normalform einer Formel herzuleiten, werden die Äquivalenzen der folgenden Tabelle[2] in die Richtung "$\leftharpoondown$" als Transformationen benutzt:

$\forall_x \phi \rightleftharpoons \phi$ falls x nicht frei in ϕ	$\exists_x \phi \rightleftharpoons \phi$ falls x nicht frei in ϕ
$\forall_x \neg\phi \rightleftharpoons \neg\exists_x \phi$	$\exists_x \neg\phi \rightleftharpoons \neg\forall_x \phi$
$\forall_x (\phi \wedge \psi) \rightleftharpoons (\forall_x \phi) \wedge (\forall_x \psi)$	$\exists_x (\phi \wedge \psi) \rightleftharpoons (\exists_x \phi) \wedge \psi$ falls x nicht frei in ψ $\exists_x (\phi \wedge \psi) \rightleftharpoons \phi \wedge (\exists_x \psi)$ falls x nicht frei in ϕ
$\forall_x (\phi \vee \psi) \rightleftharpoons (\forall_x \phi) \vee \psi$ falls x nicht frei in ψ $\forall_x (\phi \vee \psi) \rightleftharpoons \phi \vee (\forall_x \psi)$ falls x nicht frei in ϕ	$\exists_x (\phi \vee \psi) \rightleftharpoons (\exists_x \phi) \vee (\exists_x \psi)$
$\forall_x (\phi \Longrightarrow \psi) \rightleftharpoons (\exists_x \phi) \Longrightarrow \psi)$ falls x nicht frei in ψ $\forall_x (\phi \Longrightarrow \psi) \rightleftharpoons \phi \Longrightarrow (\forall_x \phi)$ falls x nicht frei in ϕ	$\exists_x (\phi \Longrightarrow \psi) \rightleftharpoons (\forall_x \phi) \Longrightarrow (\exists_x \psi)$
$\forall_x \forall_y \phi \rightleftharpoons \forall_y \forall_x \phi$	$\exists_x \exists_y \phi \rightleftharpoons \exists_y \exists_x \phi$

Man kann durch eine Umbenennung der Variabeln erreichen, daß solche Transformation immer möglich ist. Als Beispiel dient die Formel $\Psi = (\forall_x \psi) \vee \phi$. Die Variable x komme in ϕ frei vor. Weiter sei y eine neue Variable, die nicht in ψ und nicht in ϕ vorkommt. Dann ist $\Psi = (\forall_y \psi[x/y]) \vee \phi$, die die Voraussetzungen für eine Transformation erfüllt, und man bekommt $\Psi = \forall_y (\psi[x/y] \vee \phi)$.
Bei der Erstellung der pränexen Normalform läßt sich die Auflösung der Äquivalenz- und Antivalenzrelationen, wenn diese Quantoren enthalten, nicht vermeiden.

2.6 Antipränexe Normalform

Ein Ausdruck ist in antipränexer Normalform, wenn alle seine Quantoren so weit wie möglich innen in Teilformeln stehen. Um die antipränexe Normalform einer Formel herzuleiten, werden die gleichen Äquivalenzen wie bei der Herleitung von der pränexen Normalform (in Richtung "$\rightharpoonup$") verwendet.
Hier ist eine Bewegung der Quantoren in die Richtung von atomaren Formeln nicht immer möglich, da die Bedingung des Freiseins der gebundenen Variablen in einer

[1] Bei dieser Implementierung wurden diese Ersetzung von Termen durch Wahrheitswerte nicht durchgeführt

[2] Die Tabelle ist von N. Eisinger.

Teilformel nicht immer gewährleistet ist und auch nicht durch eine Variablenumbenennung erreicht werden kann. Dies ist zum Beispiel bei der Formel: $\forall_x(p(x) \vee q(x))$ der Fall.

2.7 Skolemsche Normalform

Ein Ausdruck ist in existenzquantorfreier Form (oder in skolemscher Normalform), wenn er keine echten Existenzquantoren enthält. Mit echten Existenzquantoren sind Existenzquantoren im Bereich einer geraden Anzahl von Negationen und Allquantoren im Bereich einer ungeraden Anzahl von Negationen gemeint.
Um eine skolemsche Normalform aus einer Formel zu konstruieren, müssen also die Existenzquantoren eliminiert werden. Dafür werden alle durch Exsistenzquantoren gebundenen Variablen durch (eventuell 0-stellige) Funktionen ersetzt. Falls der Existenzquantor sich in der Subformel eines (oder mehrerer) Allquantoren befindet (damit sind Allquantoren im Bereich einer geraden Anzahl von Negationen und Existenzquantoren im Bereich einer ungeraden Anzahl von Negationen gemeint), werden seine Variablen durch Funktionen ersetzt, die alle vorangehenden allgebundenen Variablen als Argumente enthalten. Sonst werden diese Variablen durch Konstanten ersetzt.
Es muß natürlich berücksichtigt werden, daß die Symbole für neue Funktionen und Konstanten nicht schon in der Formel vorhanden sind.

2.8 Negationsnormalform

Ein Ausdruck ist in Negationsnormalform, wenn er in pränexer Normalform ist, keine Existenzquantoren enthält, und seine Matrix in Negationsform ist.
Um eine Negationsnormalform aus einer Formel zu konstruieren, wird diese Formel schrittweise umgeformt.

2.9 Klauselnormalform

Ein Ausdruck ist in clausal_normal_form, wenn er in prenäxer Normalform ist, keine Existenzquantoren enthält. und seine Matrix in konjunktiver Normalform ist.
Um eine Klauselnormalform aus einer Formel zu konstruieren, wird diese Formel schrittweise umgeformt.

3 Verfahren

Um die implementierten Prädikate zu benutzen, sollen sie an konkrete Anforderungen der Benutzer angepaßt werden. Die Prädikate müssen vor allem die von Benutzer verwendete Syntax kennen. Weiterhin kann die Unterscheidung von Konstanten und Variablen und die Definition der atomaren Formeln den Anwendungen des Benutzers von der hier benutzten abweichen. In diesem Fall müssen die entsprechende Prädikate neu definiert werden.

3.1 Eine Syntax definieren

Jede Syntax hat einen Namen, der als erstes festgelegt werden soll. Das geschieht mit Hilfe des Prädikates `syntax/1`. Ein Beispiel wäre:

`syntax(tex`[3] `) .`

[3] Die Behandlung von Formeln der Syntax `tex` bedarf einige Manipulationen, auf die in der Datei `syntaxen.pl` hingewiesen ist

Als nächstes sollen die logischen Verknupfüngen in dieser Syntax mit Hilfe des folgenden Prädikates bekanntgegeben werden:

```
formula(?Syntax, ?Operator,?Components,?Formula).
```

Zum Beispiel soll "$\Longrightarrow$" als ein als Infixoperator verwendetes Symbol für die Implikation eingeführt werden.
Definition von "$\Longrightarrow$" als ein binärer Operator:

```
op(500,xfy,⟹).
```

Definition der `implication` als eine Verknüpfung der Syntax `tex`:

```
formula(tex, implication , [ A,B ], A ⟹ B).
```

3.2 Eine neue Verknüpfung definieren

Als unäre Verknüpfung ist zur Zeit nur die Negation definiert. Als binäre Verknüpfung sind Konjunktion, Disjunktion, Implikation, Rückimplikation, Äquivalenz und Antivalenz definiert. Die Allquantor und Existenzquantor sind die einzigen Quantoren. Wenn eine neue Verknüpfung definiert wird, kann man in der Datei formula.pl bestimmen, zu welcher Klasse sie gehört.
Eine Verknüpfung `nor` kann zum Beispiel folgendermaßen als eine binäre Verknüpfung definiert werden:

```
binary_connected(S,Formula):-
          ⋮
      ; formula(_,nor,_,Formula).
```

Wenn eine Verknüpfung als unär oder binär definiert wird, wird diese automatisch auch als Verknüpfung (`connected`) anerkannt.
Wie eine neue (von den oben genannten verschidene) Verknüpfung bei der Erstellung von Normalformen behandelt werden soll, muß der Benutzer selbst bestimmen. Das einfachste ist, daß man die neu definierte auf schon definierte Verknüpfungen zurückführt.

3.3 Definition der atomaren Formel ändern

Ein Atom ist allgemein als eine Verknüpfung definiert, die nicht eine der anderen logischen Verknüpfungen ist. Es kann allerdings ein spezielles Format dafür angegeben werden und zwar auf folgende Weise:

```
formula(Syntax, atom, [Functor,Arguments], Formula):-
   ! , format_of_atomic_formula(Syntax,Functor,Arguments,Formula).
```

Dabei ist `Syntax` der Name der zugehörigen Syntax. Das Prädikat

```
format_of_atomic_formula(Syntax,Functor,Arguments,Formula)
```

wird ebenfalls durch den Benutzer definiert und legt das Format der atomaren Formel und ihrer Argumente fest. Wichtig ist, daß diese Klausel vor den Klauseln steht, die die atomare Formel für andere Syntaxen definieren.

Folgendes Beispiel definiert die atomare Formel als ein Prädikat, das durchs Voranstellen des Buchstaben 'p' am Anfang des Namens der atomaren Formel in anderen Syntaxen konstruiert ist:

```
format_of_atomic_formula(Syntax,Functor,Arguments,Formula):-
        var(Formula),nonvar(Functor),nonvar(Arguments),!,
        name(Functor,FunctorAsIntegerList),
        name(NewFunctor,[112|FunctorAsIntegerList]), % ASCII 112 = p
        Formula=..[NewFunctor,Arguments].

format_of_atomic_formula(Syntax,Functor,Arguments,Formula):-
        Formula=..[NewFunctor|Arguments],
        name(NewFunctor,[_|FunctorAsIntegerList]),
        name(Functor,FunctorAsIntegerList).
```

Das Prädikat `formula/4` kann nun für die Konstruktion von Formeln, für die Untersuchung ihrer Art oder die Bestimmung ihrer Komponenten benutzt werden. Beispiele:

```
?- formula(tex,Operator,Components,p(x) ==> q(x)).
Operator = implication
Components = [p(x), q(x)] ;
No

?- formula(tex,backimplication,[p(x), q(x)],Formula).
Formula =p(x) <== q(x) ;
No
```

Nachdem die Klauseln über Syntax und Verknüpfungen zur Datei syntax.pl hinzugefügt worden sind, können alle anderen Prädikate auf Formeln der neu definierten Syntax angewendet werden.

3.4 Definition der Konstanten und Variablen ändern

Um Formeln der mathematischen Logik bearbeiten zu können, sollen Konstanten und Variablen vorhanden sein. Konstanten sind zur Zeit als PROLOG Konstanten definiert, die mit einem der folgenden Zeichen anfangen: a, b, c, d, e, und f. Variablen sind alle anderen Zeichenketten, die im PROLOG als Konstante erlaubt sind. Die dazu gehörigen Prädikate `is_constant` und `is_variable` befinden sich in der Datei formula.pl und können entsprechend der Bedürfnisse geändert werden.

4 Beschreibung der Prädikate

Die Arbeit teilt sich in mehrere Dateien auf. Die erste zwei sind die, die die Syntaxen und Formeln definieren. Andere Programme laden diese Dateien automatisch. Im folgenden werden alle Dateien mit ihren Inhalten aufgelistet:

Datei	**Inhalt**
syntaxen.pl	Definition von Syntaxen und ihre Verknüpfungen
formula.pl	Definitionen von Variablen, Konstanten, Formelklassen und andere Prädikate, die bei der Erstellung von Normalformen als Hilfsprädikate benutzt werden
disjoiner.pl	Prüfung auf Variablendisjunkt und Umformung in variablen diejunkte Form.
basicer.pl	Prüfung auf Elementar-Junktor-Form und Umformung in Elementar-Junktor-Form
negator.pl	Prüfung auf Negationsform und Umformung in Negationsform
conjnormalizer.pl	Prüfung auf konjunktive Normalform und Umformung in konjunktive Normalform
prenexer.pl	Prüfung auf pränexe Normalform und Umformung in pränexe Normalform
antiprenexer.pl	Prüfung auf antipränexe Normalform und Umformung in antipränexe Normalform
skolemizer.pl	Prüfung auf skolemsche Normalform und Erstellung eine skolemsche Normalform
normalizer.pl	Prüfung auf Negations- bzw. Klauselnormalform und Erstellung einer Negations- bzw. Klauselnormalform Formel
converter.pl	Prädikat zur Konvertierung von einer Syntax in eine andere

In den folgenden Abschnitten werden alle dem Benutzer zur Verfügung gestellten Prädikate in Form eines Referenzhandbuches beschrieben.

4.1 Syntax und Formeln

syntax(?Syntax).

Argument `Syntax` ist ein PROLOG-Konstante.

Beschreibung Wenn `Syntax` eine definierte Syntax ist liefert `syntax(Syntax).` `Yes`. Sonst liefert es `No`.

Beispiel `syntax(tex).`

Datei `syntaxen.pl`.

Datei `syntaxen.pl`.

formula(?Syntax, ?Operator,?Components,?Formula).

Argumente `Formula` ist eine Formel in der Syntax `Syntax`, `Operator` ist die Art der Verknüpfung und `Components` ist eine Liste der Komponenten der `Formula`.

Beschreibung Unäre Verknüpfungen haben nur einen Komponenten. Binäre Verknüpfungen haben zwei Komponenten. Diese Verknüpfungen können auch infixe Operatoren sein. In dem Fall müssen diese Operatoren vorher mit Hilfe des eingebauten Prädikates `op/3` definiert worden sein.
Eine Quantifier besteht aus zwei Komponenten: a. Variablen, die er bindet und b. die Formel, die sie quantifiziert.
Eine atomare Formel hat ebenfalls zwei Komponenten. Dies ist zum Ersten ihr Funktor und zum Zweiten eine Liste ihrer Argumente, welche auch leer sein kann. Mit Hilfe dieses Prädikates kann eine Verknüpfung festgelegt werden, und es können ihre Komponente und Operatorart erfragt werden.

Beispiel

```
?- formula(dt,conjunction,Components,und(p,q)).
Components =[p,q]
Yes
```

Fehler Keine.

Datei `syntaxen.pl`.

4.2 Konstanten und Variablen

Folgende Prädikate dienen zur Überprüfung eines Symbols auf Konstante oder Variable:

is_constant(+Symbol)
is_variable(+Symbol)

Argument `Symbol` ist eine PROLOG-Konstante.

Beschreibung `is_constant` ist erfolgreich (bzw. shlägt fehl) falls `Symbol` eine Konstante ist (bzw. keine Konstante ist). `is_variable` ist erfolgreich (bzw. shlägt fehl) falls `Symbol` eine Variable ist (bzw. keine Variable ist).

Fehler Keine.

Datei `formula.pl`.

4.3 Formel-Klassen

Für eine definierte Syntax können die folgenden Prädikate zur Erkennung der Art der Formel benutzt werden:
is_atomic(?Syntax,+Formula)
is_literal(?Syntax,+Formula)
unary_connected(?Syntax,+Formula)
binary_connected(?Syntax,+Formula)
connected(?Syntax,+Formula)
quantified(?Syntax,+Formula)

Argumente `Syntax` ist eine definierte Syntax und `Formula` ist eine Formel in der Syntax `Syntax`.

Beschreibung `is_atomic` und `is_literal` überprüfen, ob eine Formel atomar bzw. ein Literal ist. Die restlichen Prädikate überprüfen, ob die äußerste Verknüpfung einer Formel (wenn alle Verknüpfungen in der Formel als prefix betrachtet werden) eine unäre (`unary_connected`), binäre (`binary_connected`), unäre oder binäre Veknüpfung (`connected`), oder ein Quantor (`quantified`) ist.

Fehler Keine.

Datei `formula.pl`.

4.4 Normalformen

4.4.1 Variablendisjunkte Formel

variable_disjoint(?Syntax, +Formula)

Beschreibung `Formula` ist eine variablendisjunkte Formel in der Syntax `syntax`.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `disjoiner.pl`.

variable_disjoint(?Syntax, ?Formula, ?VariableDisjointFormula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax`. `VariableDisjointFormula` ist eine zu `Formula` äquivalente Formel in variablendisjunkter Form. Die Variablen, die mehrfach in der Formel vorkommen, werden umbenannt. Die neuen Variablensymbole bestehen aus den alten und einer hinzugefügten Zahl.

Beispiel
`?- variable_disjoint(tex,`$(\forall_x p(x)) \wedge (\exists_x q(x))$`,F).`
`F = `$(\forall_x p(x)) \wedge (\exists_{x1} q(x1))$` ;`
`No`

Fehler `Formula` und `VariableDisjointFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `disjoiner.pl`.

4.4.2 Elementar-Junktor-Form

basic_connective_form(?Syntax, +Formula)

Beschreibung `Formula` ist eine Formel in der Syntax `syntax` und in Elementar-Junktor-Form.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `conjnormalizer.pl`.

basic_connective_form(?Syntax, ?Formula, ?BasicFormula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax`. `BasicFormula` ist eine zu `Formula` äquivalente Formel in der Elementar-Junktor-Form.

Beispiele

`?- basic_connective_form(tex,`$p(x) \Longleftrightarrow q(x)$`,F).`
`F = `$(p(x) \vee \neg q(x)) \wedge (\neg p(x) \vee q(x))$` ;`
`No`

`?- basic_connective_form(tex,`$\neg(p(x) \Longleftrightarrow q(x))$`,F).`
`F =`$\neg(p(x) \wedge q(x) \vee \neg(p(x) \vee q(x)))$` ;`
`No`

Fehler `Formula` und `BasicFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `conjnormalizer.pl`.

4.4.3 Negationsform

negation_form(?Syntax, +Formula)

Beschreibung `Formula` ist eine Formel in der Syntax `syntax` und in Negationsform.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `negator.pl`.

negation_form(?Syntax, ?Formula, ?NegationFormula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax`. `NegationFormula` ist eine zu `Formula` äquivalente Formel in Negationsform.

Beispiel `?- negation_form( tex,`$\neg(p(x) \Longleftrightarrow q(x))$`,F).`
`F = `$(\neg p(x) \vee \neg q(x)) \wedge (p(x) \vee q(x))$`;`
`No`

Fehler `Formula` und `NegationFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `negator.pl`.

4.4.4 Konjunktive Normalform

conjunctive_normal_form(?Syntax,+Formula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax` und in konjunktiver Normalform.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `conjnormalizer.pl`.

conjunctive_normal_form(?Syntax,?Formula,?ConjNormalFormula)

Beschreibung `Formula` ist eine quantorenfreie Formel in Syntax `Syntax`. `ConjNormalFormula` ist eine zu `Formula` äquivalente Formel in knojunktiver Normalform und in derselben Syntax.

Beispiele `?- conjunctive_normal_form(tex,`$p(x) \Longrightarrow (q(x) \wedge r(x))$`,F).`
`F = `$(\neg p(x) \vee q(x)) \wedge (\neg p(x) \vee r(x))$` ;`
`No`

`?- conjunctive_normal_form(tex,`$p(x) \Longrightarrow (p(x) \wedge r(x))$`,F).`
`F =`$(\neg p(x) \vee p(x)) \wedge (\neg p(x) \vee r(x))$`;`
`No`

Fehler `Formula` und `ConjNormalFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `conjnormalizer.pl`.

4.4.5 Pränexe Normalform

prenex_form(?Syntax, +Formula)

Beschreibung `Formula` ist eine Formel in der Syntax `syntax` und in pränexer Normalform.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `prenexer.pl`.

prenex_form(?Syntax, ?Formula, ?PrenexFormula)

Beschreibung `Formula` ist eine Formel in der Syntax `syntax`. `PrenexFormula` ist eine zu `Formula` äquivalente Formel in pränexer Normalform.

Beispiel
`?- prenex_form( tex,`$\forall_{x,y}(\exists_y p(x,y)) \vee q(y)$`, F).`
`F = `$\forall_{x,y}\exists_{y1}(p(x,y1) \vee q(y))$` ;`
`No`

`?- prenex_form(tex,`$(\forall_{x,y} f(x,y,z)) \wedge (\forall_{y,z} q(x,y,z))$`,F).`
`F =`$\forall_{x1,y,z1} f(x1,y,z) \wedge q(x,y,z1)$` ;`
`No`

Fehler `Formula` und `PrenexFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `prenexer.pl`.

prenex_form(?Syntax, ?Formula, ?Prefix, ?Matrix)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax` und in pränexer Normalform. `Prefix` ist der Prefix der `Formula` in Form von einer Liste von Paaren, deren erstes Element ein Quantor ist und deren zweites Element eine Liste von Variablen ist, die dieser Quantor bindet. `Matirx` ist die Matrix der `Formula`.

Beispiele
`?- prenex_form(tex,`$\forall_x\exists_{y,z} p(x,y,z)$`,Prefix,Matrix).`
`Prefix =`[[`universal_quantifier`,$[x]$], [`existential_`
`quantifier` , $[y,z]$]]
`Matrix =`$p(x,y)$` ;`
`No`

`?-`
`prenex_form(tex,Formula,`[[`universal_quantifier`,$[x]$]]`,`$q(x)$`).`
`Formula =`$\forall_x q(x)$` ;`
`No`

Fehler `Formula` und `Matrix` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `prenexer.pl`.

prenex_form(?Syntax, ?Formula, ?PrenexFormula, ?Prefix, ?Matrix)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax`. `PrenexFormula` ist eine zu `Formula` äquivalente Formel in pränexer Normalform. `Prefix` ist der Prefix der Formel `PrenexFormula` (s.o.) und `Matirx` ist ihre Matrix.

Fehler `Formula`, `PrenexFormula` und `Matrix` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `prenexer.pl`.

4.4.6 Antipränexe Normalform

anti_prenex_form(?Syntax, +Formula)

Beschreibung `Formula` ist eine Formel in Syntax `Syntax` und in antipränexer Normalform. Wenn von den durch eine Quantor gebundenen Variablen einige (oder mehrere) nicht in der Subformel vorkommen, ist diese Formel nicht in antipränexer Normalform.

Beispiele `?- anti_prenex_form(tex,`$\forall_{x,y} p(x)$`).`
`No`

`?- anti_prenex_form(tex,`$\forall_x p(x)$`).`
`Yes`

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei antiprenexer.pl

anti_prenex_form(?Syntax, ?Formula, ?AntiPrenexFormula)

Beschreibung `Formula` ist eine Formel in Syntax `Syntax`. `AntiPrenexFormula` ist eine zu `Formula` äquivalente Formel in antipränexer Normalform und in der selben Syntax. Falls gebundene Variablen nicht im Beireich des Quantors vorkommen, werden sie eliminiert.

Beispiel `?- anti_prenex_form(tex,`$\forall_x \exists_{y,z}(p(x) \wedge q(y))$`,F).`
`F = `$\forall_x p(x) \wedge \exists_y q(y)$` ;`
`No`

Fehler `Formula` und `VariableDisjointFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei antiprenexer.pl

4.4.7 Skolemsche Normalform

existential_free_form(?Syntax, ?Formula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax` und in skolemscher Normalform.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `skolemizer.pl`

existential_free_form(?Syntax, ?Formula, ?SkolemizedFormula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax`. SkolemizedFormula ist eine transformierte Formel in skolemscher Normalform.
Die Konstantensymbole werden durch Voranstellung des Buchstaben 'a' und Hinzufügen einer natürlichen Zahl an der ursprünglichen Variable gebildet. Die Funktionssymbole werden auf gleiche Weise, nur mit 'f' statt 'a', gebildet. Es wird berücksichtigt, daß die Symbole alle neu sind.
Bei der Bildung der skolemschen Normalform wird versucht, alle Verknüpfungen so lange möglich beibehalten. Eine Auflösung von Äquivalenz bzw. Antivalenz Relationen läßt sich nicht immer vermeiden.

Beispiele

`?- existential_free_form(tex,` $\forall_x(\exists_y p(x,y)) \wedge \exists_z q(z)$`,F).`
`F =`$\forall_x p(x, fy1(x)) \wedge q(az1)$ `;`
`No`

`?- existential_free_form(tex,`$\forall_x \neg(\forall_y p(x) \Longrightarrow q(y))$`,F).`
`F =` $\forall_x \neg(p(x) \Longrightarrow q(fy1(x)))$ `;`
`No`

Fehler `Formula` und `SkolemizedFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `skolemizer.pl`

4.4.8 Negationsnormalform

negation_normal_form(?Syntax, +Formula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax` und in Negationsnormalform.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `normalizer.pl`

negation_normal_form(?Syntax, ?Formula, ?NormalizedFormula)

Beschreibung `Formula` ist eine Formel in Syntax `Syntax`. `NormalizedFormula` ist eine transformierte Formel in Negationsnormalform.

Beispiel
`?- negation_normal_form(tex,` $\forall_x(\neg p(x) \Longrightarrow q(x) \wedge r(x))$`,F).`
`F =` $\forall_x(p(x) \vee q(x) \wedge r(x))$ `;`
`No`

Fehler `Formula` und `NormalizedFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `normalizer.pl`

4.4.9 Klauselnormalform

clausal_normal_form(?Syntax, +Formula)

Beschreibung `Formula` ist eine Formel in Syntax `Syntax` und in Klauselnormalform.

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `normalizer.pl`

clausal_normal_form(?Syntax, ?Formula, ?ClausalNormalFormula)

Beschreibung `Formula` ist eine Formel in der Syntax `Syntax`. `ClausalNormalFormula` ist eine transformierte Formel in Klauselnormalform.

Beispiel
`?- clausal_normal_form(tex,` $\forall_x(\neg p(x) \Longrightarrow q(x) \wedge r(x))$`,F).`
`F =` $\forall_x((p(x) \vee q(x)) \wedge (p(x) \vee r(x)))$ `;`
`No`

Fehler `Formula` und `ClausalNormalFormula` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `normalizer.pl`

clauses_list(?Syntax,+Formula,-ListOfClauses)

Beschreibung	`Formula` ist eine beliebige Formel in der Syntax `Syntax`. `ListOf-Clauses` ist eine Liste von Klauseln, die durch Konjunktion die Matrix der Klausenormalform der Formel bilden. Aus dieser Liste ist jede einzelne Klausel leicht extrahierbar.

Beispiel	`?- clauses_list(tex,` $\neg p(x) \Longleftrightarrow q(x)$`,L).`
	`L =` $[p(x) \vee q(x), \neg p(x) \vee \neg q(x)]$ `;`
	`No`

Fehler	`Formula` ist kein grundinstanziierter PROLOG-Term.
Datei	`normalizer.pl`

Als letztes Beispiel zur Normalformen wird die von Norbert Eisinger und Hans Jünrgen Ohlbach in [1] angegebene ε-Definiton des Grenzwertes in Klauselnormalform umgeformt:

`?- clauses_list(tex,`$\forall_\varepsilon \varepsilon > 0 \Longrightarrow (\exists_\delta (\delta > 0) \wedge (\forall_{x,y} x - y < \delta \Longrightarrow (g(x) - g(y) < \varepsilon)))$`,ListOfClauses).`
`ListOfClauses =`$[\neg(\varepsilon > 0) \vee (f\delta 1(\varepsilon) > 0), \neg(\varepsilon > 0) \vee \neg(x - y < f\delta 1(\varepsilon)) \vee (g(x) - g(y) < \varepsilon)]$ `;`
`No`

4.5 Syntaxkonvertierung

convert(?Syntax1, ?Formula1, ?Syntax2, ?Formula2)

Argumente	`Formula1` ist eine Formel in der Syntax `Syntax1` und `Formula2` ist eine äquivalente Formel in der Syntax `Syntax2`.
Beschreibung	Konvertiert eine beliebige Formel von einer Syntax in eine andere.

Beispiel	`?- convert(tex,`$\forall_y p(x) \Longrightarrow \neg q(x) \wedge r(x)$`, dt, F).`
	`F =` $fueralle([y], impliziert(p(x), und(nicht(q(x)), r(x))))$ `;`
	`No`
	(In der Syntax `dt` sind alle Verknüpfungen als prefix Operatoren definiert.)

Fehler	`Formula1` und `Formula2` sind beide nicht grundinstanziierte PROLOG-Terme.
Datei	`converter.pl`

4.6 Hilfsmittel

Als Hilfsmittel wurden folgende Prädikate implementiert.

complements(?Syntax,?Literal1,?Literal2)

Argumente `Syntax` ist eine beliebige Syntax. `Literal1` und `Literal2` sind zwei komplementäre Literale in der Syntax `Syntax`.

Beschreibung `Literal1` und `Literal2` sind komplementäre Literalen in der Syntax `Syntax`.

Beispiel `?- complements(tex,`$p(x)$`,F).`
`F = `$\neg p(x)$` ;`
`No`

Fehler `Literal1` und `Literal2` sind beide nicht grundinstanziierte PROLOG-Terme.

Datei `formula.pl`

free_vars_of(?Syntax,+Formula,-FreeVars)

Argumente `Syntax` ist eine beliebige Syntax. `Formula` ist eine beliebige Formel in der Syntax `Syntax`. `FreeVars` ist eine Liste von PROLOG-Konstanten.

Beschreibung `FreeVars` ist eine Liste aller in `Formula` vorkommenden freien Variablen.

Beispiel `?- free_vars_of(tex,`$p(x) \wedge (\forall_z q(y,z))$`,L).`
`L =[x,y] ;`
`No`

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `formula.pl`

function_symbols_of(?Syntax,+Formula,-Symbols)

Argumente `Syntax` ist eine beliebige Syntax. `Formula` ist eine beliebige Formel in der Syntax `Syntax`. `Symbols` ist eine Liste von PROLOG-Konstanten.

Beschreibung `Symbols` ist eine Liste aller in `Formula` vorkommenden Funktionssymbole.

Beispiel `?- function_symbols_of(tex,`$p(a) \wedge (\forall_z q(f(b,y),g(c)))$`,L).`
`L =[a,f,b,g,c] ;`
`No`

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `formula.pl`

eliminate_quantifier(?Syntax,+Formula,-NewFormula)

Argumente `Syntax` ist eine beliebige Syntax. `Formula` und `NewFormula` sind zwei Formeln in der Syntax `Syntax`.

Beschreibung Wenn es in `Formula` Quantoren gibt, deren Variablen in der Subformel nicht vorkommen, dann werden diese Quantoren eliminiert. Falls nur ein Teil dieser Variablen nicht in der Subformel vorkommt, dann wird dieser Teil eliminiert. `NewFormula` ist die so entstehende Formel.

Beispiel

`?- eliminate_quantifier(tex,`$\forall_x \exists_{y,z} p(z)$`, F).`
`F = `$\exists_z p(z)$` ;`
`No`

Fehler `Formula` ist kein grundinstanziierter PROLOG-Term.

Datei `formula.pl`

new_symbol(+ForbiddenSymbols,+OldSymbol,-NewSymbol) und **new_symbols(+ForbiddenSymbols,+OldSymbols,-NewSymbols)**

Argumente `ForbiddenSymbols`, `OldSymbols` und `NewSymbols` sind **Mengen** von beliebigen PROLOG-Konstanten, `OldSymbol` und `NewSymbol` sind zwei PROLOG-Konstanten.

Beschreibung Bei `new_symbol/3` wird das neue Symbol `NewSymbol` gebildet, das kein Element von `ForbiddenSymbols` ist. Dieses Symbol wird gebildet indem eine naturliche Zahl an `OldSymbol` gehängt wird.
Bei `new_symbols/3` wird dementsprechend für jedes Element von `OldSymbols` ein neues Symbol gebildet. `NewSymbols` ist die Menge der neu genbildeten Symbole.

Beispiele

`?- new_symbol(`$[a, b, b1, c], b$`,X).`
`X= `$b2$` ;`
`No`

`?- new_symbols(`$[a, b, b1, ac], [b, ac]$`,X).`
`X= `$[b2, ac1]$` ;`
`No`

Fehler `ForbiddenSymbols` oder `OldSymbols` sin keine Listen.

Datei `formula.pl`

5 Schluß

Man könnte diese Arbeit erweitern, indem man die Definition der Prädikate auf Formelmengen erweitert und anhand von Formelmengen ganze Logik-Programme erstellt.
Die Implementierung der Prädikate war mit Rekursion auf Grund von Aufbau von logischen Formeln leicht möglich. Das schwierigste Problem war dabei Einbau von Mechanismen zur Kontrole von jeweils gebrauchten Symbole, wo immer weitere Symbole (sei es Variablensymbole oder Funktionssymbole) in die Formel eingefügt werden mussten. So eine Kontrolle setzt voraus, daß die Teile einer Formel in einer Reihenfolge bearbeitet werden, was sonst für logische Formeln nicht nötig ist. Die Berarbeitung der Daten in einer Reihenfolge ist außerdem von prozeduraler Natur, was mit einer deklarativen Programiersprache nur über Umwege möglich ist.

Literatur

[1] Norbert Eisinger and Hans Jürgen Ohlbach. *Deduction Systems Based on Resolutuion.* Handbook of logic in artificial intelligence and logic programming. Clarandon press, Oxford, 1993.

[2] H.-D. Ebbinghaus, J. Flum, W. Thomas. *Einführung in matematische Logik.* Wissenschaftsverlag, Mannheim, 1992.

[3] H. Hermes. *Einführung in die mathematische Logik.* B.G. Teubner Stuttgart, 1972.

[4] Uwe Schöning. *Logik für Informatiker.* Wissenschaftsverlag, Manheim, 1989.

Anhang: Listing