13 Erweiterungen der Prädikatenlogik

In diesem Kapitel werden Erweiterungen der Prädikatenlogik vorgestellt, die es erlauben, einerseits bestimmte Sachverhalte einfacher, bzw. überhaupt auszudrücken und andererseits den Deduktionskalkül anzupassen.

13.1 Mehrsortige Logiken

Es gibt syntaktische Varianten von PL_1, die zwar nicht die prinzipielle Ausdrucksfähigkeit erhöhen, aber eine kompaktere Darstellung von Formeln erlauben ([Wal87,SS89,Wei95,Wei96]). Damit ist in den einzelnen Unterausdrücken mehr Information konzentriert, die dann von Kalkülen ausgenutzt werden kann, um sinnlose Aussagen zu erkennen und zu vermeiden. Die wichtigsten derartigen Varianten sind die *mehrsortigen Logiken*. Die Grundidee dabei ist, nicht einfach ein strukturloses Universum zu betrachten, sondern die Objekte in Klassen einzuteilen. Aussagen werden nur für Objekte bestimmter Klassen formuliert und sind für andere Objekte gar nicht definiert. Beispielsweise könnte man Aussagen über Menschen und über natürliche Zahlen in einer mehrsortigen Logik so formulieren, daß die Behauptung, Sokrates und Napoleon seien teilerfremd, nicht einfach wahr oder falsch, sondern sinnlos wäre.

Dazu stellen mehrsortige Logiken zusätzlich zu den genannten Datenobjekten noch eine Menge von *Sorten* zur Verfügung. Die Sorten können einfach eine unstrukturierte Menge bilden (flache Sortenstruktur) oder durch eine partielle Ordnung $\sqsubseteq$, die *Untersortenbeziehung,* geordnet sein (hierarchische Sortenstruktur). Man würde etwa die Menge der Sortensymbole $\{Integer, Menschen\}$ flach und die Menge $\{Integer, Real, Complex\}$ hierarchisch strukturieren. Die zulässige Struktur der Untersortenbeziehung, z.B. linear, baumförmig oder Halbverband hat Einfluß auf die möglichen Einsetzungen für Variablensymbole und beeinflußt daher die Konstruktion von Kalkülen für die jeweilige spezielle Logik.

Die Signatur der Logik wird nun folgendermaßen erweitert: jedem Konstanten- und Variablensymbol ist eine Sorte zugeordnet, Funktionssymbolen werden Argument/ Ergebnissorten- Beziehungen mit der entsprechenden Stelligkeit zugeordnet, Prädikatensymbolen die zulässigen Argumentsorten. Beispielsweise spezifiziert

$$Geburtsjahr : Mensch \to Integer$$

eine Funktion, die nur Argumente der Sorte $Mensch$ zuläßt und dann ein Ergebnis der Sorte $Integer$ liefert. Mit

$$\begin{aligned} + : (&Real \times Real \to Real \\ &Real \times Integer \to Real \\ &Integer \times Real \to Real \\ &Integer \times Integer \to Integer \end{aligned}$$

charakterisiert man mehrere mögliche Beziehungen der Argumentsorten zu den Ergebnissorten der Funktion $+$. $Teilerfremd : Integer \times Integer$ besagt, daß das Prädikatensymbol $Teilerfremd$ nur Terme vom Typ $Integer$ als Argument akzeptiert.

Die Sorte eines zusammengesetzten Terms wird nun aus der Sortendeklaration für das oberste Funktionssymbol und eventuell aus den Sorten der Unterterme berechnet. Ein korrekter Term $Geburtsjahr(\ldots)$ hat zum Beispiel immer die Sorte $Integer$, während die Sorte eines Terms $+(\ldots,\ldots)$ von den Sorten der Argumente abhängt. Ein Term wie $Geburtsjahr(+(\ldots,\ldots))$ ist nicht *sortenrecht* und daher verboten. Damit wird rein syntaktisch die Formulierung vieler unsinniger Aussagen verhindert.

Daß mehrsortige Logiken nicht wirklich ausdrucksstärker sind als unsortierte, zeigt sich darin, daß die gesamte Information über die Sorten in einstellige Prädikate kodiert werden kann: Die Untersortenbeziehungen $S \sqsubseteq P$ werden als Implikationen ausgedrückt: $\forall x : S(x) \Rightarrow P(x)$. Quantifizierungen über sortierte Variable $\forall x : S : \mathcal{F}$ werden in die Form $\forall x : S(x) \Rightarrow \mathcal{F}$ übersetzt, und die Argument/Ergebnissorten-Beziehungen der Funktionen $f : S_1 \times \ldots \times S_n \to S$ lassen sich durch Formeln $\forall x_1 \ldots x_n : S_1(x_1) \wedge \ldots \wedge S_n(x_n) \Rightarrow S(f(x_1,\ldots,x_n))$ beschreiben. Semantisch unsinnige Terme wie der oben erwähnte Term $Geburtsjahr(+(\ldots,\ldots))$ sind in dieser Form jedoch nicht ausgeschlossen, deshalb setzten sich mehrsortige Logiken in der Praxis immer mehr durch.

Beispiel 13.1. Ein Beispiel, welches zeigt, wie per Resolution in unsortierer Prädikatenlogik ziemlicher Unsinn abgeleitet werden kann:

Axiome:

Jeder Mensch, der Bananen ißt, lebt gesund. (Ob das stimmt oder nicht steht hier nicht zur Diskussion)
Judy ist ein Affe und ißt Bananen
Tom ist ein Mensch und ißt Bananen.

Kodierung in PL_1:

$$\forall x : \mathtt{Mensch}(x) \wedge \mathtt{isst}(x, \mathtt{Bananen}) \Rightarrow \mathtt{lebtgesund}(x)$$

$$\mathtt{Affe}(\mathtt{Judy})$$

$$\mathtt{isst}(\mathtt{Judy}, \mathtt{Bananen})$$

$$\mathtt{Mensch}(\mathtt{Tom})$$

$$\mathtt{isst}(\mathtt{Tom}, \mathtt{Bananen})$$

Klauselform:

$C1 : \neg\texttt{Mensch}(x), \neg\texttt{isst}(x, \texttt{Bananen}), \texttt{lebtgesund}(x)$

$C2 : \texttt{Affe}(\texttt{Judy})$

$C3 : \texttt{isst}(\texttt{Judy}, \texttt{Bananen})$

$C4 : \texttt{Mensch}(\texttt{Tom})$

$C5 : \texttt{isst}(\texttt{Tom}, \texttt{Bananen})$

Jetzt ist z.B. eine Resolution mit $C1, 2$ und $C3$ möglich. Die Resolvente ist: $\neg\texttt{Mensch}(\texttt{Judy}), \texttt{lebtgesund}(\texttt{Judy})$.

Um damit weiterzuarbeiten, müßte man einen Resolutionspartner für $\neg\texttt{Mensch}(\texttt{Judy})$ finden, d.h. eine Klausel mit einem Literal $\texttt{Mensch}(\texttt{Judy})$, was offensichtlich in der intendierten Bedeutung nicht möglich ist. D.h. die Resolvente ist nutzlos. Um diese Resolution zu verhindern, muß man die Variablen typisieren (mit Sorteninformation versehen), so daß eine Einsetzung $x \mapsto \texttt{Judy}$, wobei x die Sorte `Mensch` und `Judy` die Sorte `Affe` hat, verboten werden kann.

Jetzt zu den formalen Definitionen. Wir definieren eine einfache Variante von Sortenlogik erster Ordnung: OSPL (*Order Sorted Predicate Logic*).

13.2 Syntax von OSPL

Wir halten uns an das Schema wie es auch in Kapitel 4.1 zur Definition von PL_1 benutzt wurde. D.h. wir sagen zunächst mal, was die Signatur, d.h. die verwendeten Symbole sind. Im Unterschied zu unsortierten Signaturen nehmen wir die Variablen diesmal mit in die (sortierte) Signatur auf.

Definition 13.2. Signaturen mit Sorten
Eine sortierte Signatur S besteht aus den Komponenten $(V, F, P, S, \mathcal{S}, SD, FD, PD)$. V, F, P sind die Variablensymbole, Funktionssymbole und Prädikatensymbole, so wie in Definition 4.1. Die weiteren Komponenten sind:

- *S ist eine Menge von Sortensymbolen*
- *$\mathcal{S} : V \to S$ ist eine Funktion, die jedem Variablensymbol eine Sorte zuordnet. Meist schreibt man jedoch z.B. $x : A$ anstelle von $S(x) = A$ um zu sagen, daß die Variable x die Sorte A hat.*
- *SD ist eine Menge von Untersortendeklarationen der Form $R \sqsubseteq S$,*
- *FD ist eine Menge von Sortendeklarationen für Funktionen. Eine Sortendeklaration für Funktionen ist einfach ein Tupel $t : S$ welches besagt, daß alle Instanzen des Terms t von der Sorte S sind, wobei nur Terme der Form $f(x_1, \ldots, x_n)$ mit verschiedenen x_i zugelassen sind. Oft werden Deklarationen $f(x_1, \ldots, x_n) : S$ als $f : S_1 \times \ldots \times S_n \to S$ geschrieben, wobei S_i die Sorte der Variable x_i ist.*

– *PD ist eine Menge von Sortendeklarationen für Prädikatensymbole. Sie haben die Form $P : S_1 \times \ldots \times S_n$, wobei die S_i Sorten sind.*

Wir nehmen an, daß das Gleichheitsprädikat immer vorhanden ist und daß die Deklarationen $=: R \times S$ für alle Sorten R, S in PD sind, d.h. Gleichheit ist für alle Sorten definiert. Für eine Signatur Σ, sei $\sqsubseteq_\Sigma$ die partielle Ordnung auf S, die durch die reflexive und transitive Hülle der Untersortendeklarationen gegeben ist.

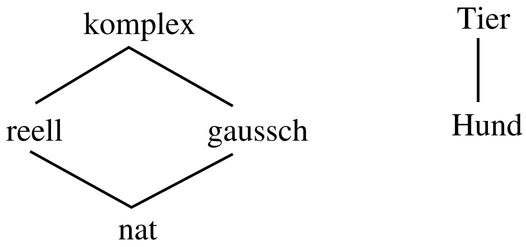
Beispiel 13.3. Beispiel für eine sortierte Signatur:

Variablensymbole:	$\{x, y, z, u, v, w \ldots\}$
Funktionssymbole:	$\{+, *, inv, o, Fido, Tweety, Vater\}$,
	$arity(+) = arity(*) = 2, arity(inv) = 1, arity(Fido) = arity(o) = 0$
	$arity(Tweety) = 0, arity(Vater) = 1$
Prädikatensymbole:	$\{<, gerade, brav, =\}$
Sortensymbole:	$\{komplex, reell, nat, gaussch, Hund, Tier\}$
Sortenfunktion:	$\mathcal{S}$: $\mathcal{S}(x) = komplex, \mathcal{S}(y) = reell, \mathcal{S}(z) = nat, \mathcal{S}(u) = Hund \ldots$
Untersorten:	$reell \sqsubseteq komplex, gaussch \sqsubseteq komplex, nat \sqsubseteq gaussch, nat \sqsubseteq reell, Hund \sqsubseteq Tier$
Funktionen:	$o : nat$
	$+ : komplex \times komplex \to komplex$
	$reell \times reell \to reell$
	$nat \times nat \to nat$
	$* : komplex \times komplex \to komplex$
	$reell \times reell \to reell$
	$nat \times nat \to nat$
	$inv : komplex \times komplex \to komplex, \ldots$
	$Fido : Hund; Tweety : Tier; Vater : Hund \to Hund \ldots$
Prädikate:	$gerade : nat; <: komplex \times komplex; brav : Hund \ldots$

Aus den Untersortendeklarationen ergibt sich z.B.:

$komplex \sqsubseteq_{\Sigma} komplex, \ldots$	(reflexive Hülle)
$reell \sqsubseteq_{\Sigma} komplex, \ldots$	(war sowieso schon der Fall)
$nat \sqsubseteq_{\Sigma} komplex, \ldots$	(transitive Hülle)

Meist stellt man das auch graphisch dar:



In der so definierten Signatur steckt Information, die man sonst explizit mit Formeln ausdrücken müßte. Z.B. ist die Deklaration $reell \sqsubseteq komplex$ ausdrückbar in unsortiertem PL_1 mit $\forall x : reell(x) \Rightarrow komplex(x)$.

Als nächstes müssen wir die Terme neu definieren. Dabei sollen Terme der Art $inv(Fido)$ oder $Vater(*(o, o))$ als "nicht wohlsortiert" ausgesondert werden. Das macht man, indem man positiv die Menge der wohlsortierten Terme definiert. Die Idee dafür ist, von den Deklarationen für Terme auszugehen, die sind allemal wohlsortiert, und daraus durch wohlsortierte Instantiierung weitere wohlsortierte Terme zu generieren.

Definition 13.4. Wohlsortierte Terme *Die Menge der wohlsortierten Terme $T_{\Sigma,A}$ der Sorte A in der Signatur $\Sigma = (V, F, P, S, \mathcal{S}, SD, FD, PD)$ wird durch folgende drei Regeln rekursiv konstruiert:*

- $x \in T_{\Sigma,A}$, *falls* $\mathcal{S}(x) \sqsubseteq_{\Sigma} A$.
- $f(x_1, \ldots x_n) \in T_{\Sigma,A}$, *falls mit* $x_i : S_i$ *und* $f : S_1 \times \ldots \times S_n \rightarrow A \in FD$ *und* $R \sqsubseteq_{\Sigma} A$
- $t[r/x] \in T_{\Sigma,A}$ *falls* $t \in T_{\Sigma,A}, r \in T_{\Sigma,R}$ *und* $x \in V, x : S$ *so daß* $R \sqsubseteq_{\Sigma} S$. [4]

Die Menge T_{Σ} aller wohlsortierten Terme über Σ (Σ-Terme) erhält man als Vereinigung:

$$T_{\Sigma} = \bigcup \{T_{\Sigma,A} \mid A \in S\}$$

Die Sorte $S(t)$ eines Terms t ist die Sorte A mit kleinster Menge (als Mengeninklusion) $T_{\Sigma,A}$, welches t enthält. Wenn diese Sorte nicht eindeutig ist, liefert $S(t)$ die Menge aller Sorten A, so daß $t \in T_{\Sigma,A}$.

Manchmal schreibt man auch $t : A$, um auszudrücken, daß A die Sorte von t ist.

[4] $t[r/x]$ bedeutet, r für x in t einsetzen.

Beispiel 13.5. In der Signatur von Beispiel 13.3 ist:

$$
\begin{aligned}
T_{\Sigma,Hund} &= \{Fido, Vater(Fido), Vater(Vater(Fido)), \ldots \\
&\quad u, Vater(u), Vater(Vater(u)), \ldots\} \\
T_{\Sigma,Tier} &= T_{\Sigma,Hund} \cup \{Tweety\} \\
T_{\Sigma,nat} &= \{o, +(o,o), *(o,o), +(o,*(o,o)), \ldots, \\
&\quad z, +(z,z), +(o,z), \ldots\} \\
T_{\Sigma,reel} &= T_{\Sigma,nat} \cup \{y, +(y,y), \ldots\} \\
S(Vater(Fido)) &= Hund
\end{aligned}
$$

Im nächsten Schritt werden die zulässigen Formeln definiert. Der Unterschied zu PL_1 ist im wesentlichen, daß die Sortendeklarationen für Prädikate benutzt werden, um nicht wohlsortierte Atome wie z.B. $brav(+(o,o))$ im obigen Beispiel auszuschließen.

Definition 13.6. Wohlsortierte Formeln
Ein Atom $P(t_1, \ldots, t_n)$ ist wohlsortiert, *falls $t_i \in T_{\Sigma,S_i}$ für $i = 1, \ldots, n$ und $P : S_1 \times \ldots \times S_n$ ist in Σ. Eine wohlsortierte Formel ist eine Formel, die mit wohlsortierten Atomen und den logischen Junktoren und Quantoren $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow, \forall, \exists$ wie üblich gebildet wird. Wir schreiben $\forall x : A : F$ und $\exists x : A : F$, um anzuzeigen, daß A die Sorte der Variablen x ist.*

Beispiel 13.7. In der Signatur von 13.3 sind die Formeln $<(o,o), \forall x : komplex :< (x,x)$ wohlsortiert, während $<(Fido, Tweety)$, $brav(o)$,$\forall x : komplex : brav(x)$ nicht wohlsortiert sind.

13.3 Semantik von OSPL

In Abschnitt 4.2 haben wir die Semantik von PL_1 über Σ-Algebren und Σ-Homomorphismen definiert. Mit den Erweiterungen von OSPL gegenüber PL_1 muß man auch die Definitionen von Σ-Algebren und Σ-Homomorphismen entsprechend anpassen. Insbesondere benötigen wir in der Semantik Gegenstücke zu den Sortensymbolen, Untersortendeklarationen und Deklarationen für Terme und Prädikatensymbolen.

Wir verzichten hier auf weitergehende Formalisierung und geben nur die Idee an: die Sortensymbole als Teilmengen der Trägermenge zu interpretieren, die Untersortenbeziehungen als Teilmengenbeziehungen und die Deklarationen im wesentlichen als Einschränkungen über Definitions- und Wertebereiche von Funktionen aufzufassen. Das heißt, daß Interpretationen und Modelle der Sortensignatur angepaßte Bedingungen erfüllen müssen. Eine wichtige Einschränkung ist, daß die Einsetzungen für Variablen deren Sorte entsprechen muß. D.h. wenn $x : A$, dann darf nur ein Element d aus dem Domain für x eingesetzt werden, das zur Menge D_A gehört. Dies betrifft zum Beispiel die Variablen die durch Quantoren gebunden sind. Die Definitionen der Relation der Erfüllbarkeit die gleiche wie im unsortierten Fall.

13.4 Relativierung

Das erste was sich Logiker fragen, wenn sie mit einer neuen Logik konfrontiert sind, ist, ob diese neue Logik wirklich ausdrucksstärker ist als schon vorhandene, d.h. ob man darin Sachverhalte ausdrücken kann, die man in anderen Logiken, insbesondere einfacher Prädikatenlogik, prinzipiell nicht ausdrücken kann.

Wenn der Verdacht besteht, daß das nicht der Fall ist, kann man diese Vermutung dadurch bestätigen, daß man eine korrekte und vollständige Methode angibt, Formeln dieser Logik in eine andere Logik zu übersetzen (zu *relativieren*). Korrekt und vollständig heißt in diesem Zusammenhang, daß eine erfüllbare Formel der Ausgangslogik in eine erfüllbare Formel der Ziellogik übersetzt wird (Korrektheit) und umgekehrt, wann immer die übersetzte Formel erfüllbar ist, ist auch die Ausgangsformel erfüllbar (Vollständigkeit). Solche Übersetzer bezeichnet man auch manchmal als konservative Transformationen.

Hat man einen Übersetzer dieser Art definiert, dann kann man im Prinzip auf einen Kalkül in der neuen Logik verzichten. Man übersetzt in die Ziellogik und beweist die übersetzte Formel. Dann weis man, daß die ursprüngliche Formel ebenfalls gültig ist. Ob diese Vorgehensweise zweckmäßig ist, hängt von praktischen Gesichtspunkten ab - wie effizient die Kalküle sind, ob Implementierungen von Beweisern zur Verfügung stehen, usw. Für OSPL geben wir jetzt eine Übersetzung in unsortierte Prädikatenlogik an. Sie zeigt einerseits daß OSPL nicht wirklich ausdrucksstärker ist als PL_1, und erlaubt andererseits die Kalküle zu vergleichen. (Für OSPL wird ein Kalkül noch nachgeliefert.) Die Übersetzung in PL_1 wird bewußt ausführlich betrachtet, weil der Mechanismus prototypisch für weitere Übersetzungsverfahren ist, die Formeln nichtklassischer Logiken in Prädikatenlogik übersetzen. In diesen Fällen benutzt man dann die Übersetzung, um mit Resolution auf den übersetzten Formeln zu operieren.

Definition 13.8. Übersetzer: $OSPL \to PL_1$)
Gegeben eine OSPL-Spezifikation $(\Sigma, \mathcal{F})$, *d.h. eine OSPL Signatur* Σ *zusammen mit einer Menge* $\mathcal{F}$ *von geschlossenen Formeln über* Σ, *dann besteht ein Übersetzer, der diese OSPL-Spezifikation in eine* PL_1 *Spezifikation* $(\Sigma', \mathcal{F}')$ *überträgt, aus drei Teilen:*

Ψ_{sig} *übersetzt die OSPL-Signatur in eine* PL_1*-Signatur,*

$\Psi_{sig,F}$ *generiert aus der OSPL-Signatur* PL_1*-Formeln und*

Ψ_{Σ} *übersetzt die Formeln.*

Sie sind folgendermaßen definiert:

Ψ_{sig}: *Sei $\Sigma = (V, \mathcal{F}, \mathcal{P}, S, \mathcal{S}, SD, FD, PD)$ eine OSPL Signatur. Dann ist $\Psi_{sig}(\Sigma) = (\mathcal{F}, \mathcal{P} \cup \{A' \mid A \in S\})$. Es ist $arity(A') = 1$. (d.h. aus jedem Sortensymbol wird ein einstelliges Prädikatensymbol gemacht.)*

$\Psi_{sig,F}$:

$$\Psi_{sig,F}(\Sigma) = \{\forall x : A(x) \Rightarrow B(x) \mid A \sqsubseteq B \in SD\}$$
$$\cup\{\forall x_1, \ldots x_n(A_1(x_1) \wedge \ldots \wedge A_n(x_n)) \Rightarrow A(f(x_1, \ldots x_n)) \mid f(x_1, \ldots x_n) : A \in FD\}$$

Die Deklarationen für Prädikatensymbole werden nicht übersetzt.

Ψ_Σ: *Die Übersetzung von Formeln wird rekursiv über die Formelstruktur definiert:*

Basisfall:

$$\Psi_\Sigma(L) = L \text{ falls } L \text{ ein Atom ist.}$$

Rekursionsfälle:

$$\Psi_\Sigma(\neg F) = \neg\Psi_\Sigma(F)$$
$$\Psi_\Sigma(F@G) = \Psi_\Sigma(F)@\Psi_\Sigma(G), @ \in \{\vee, \wedge, \Rightarrow, \Leftrightarrow\}$$
$$\Psi_\Sigma(\forall x : A : \ F) = \forall x : A(x) \Rightarrow \Psi_\Sigma(F)$$
$$\Psi_\Sigma(\exists x : A : \ F) = \exists x : A(x) \wedge \Psi_\Sigma(F)$$

Insgesamt ergibt sich also: $\Psi(\Sigma, \mathcal{F}) = (\Psi_{sig}(\Sigma), \Psi_{sig,F}(\Sigma) \cup \{\Psi_\Sigma(F) \mid F \in \mathcal{F}\})$.

Beispiel 13.9. Für eine Übersetzung von OSPL in PL_1
Wir nehmen im wesentlichen wieder die Signatur aus 13.3:

Variablensymbole:	$\{x, y, z, u, v, w \ldots\}$
Funktionssymbole:	$\{+, *, inv, o, Fido, Tweety, Vater\}$,
	$arity(+) = arity(*) = 2, arity(inv) = 1, arity(Fido) = arity(o) = 0$
	$arity(Tweety) = 0, arity(Vater) = 1$
Prädikatensymbole:	$\{<, gerade, brav, =\}$
Sortensymbole:	$\{komplex, reell, nat, gaussch, Hund, Tier\}$
Sortenfunktion:	$\mathcal{S}$: $\mathcal{S}(x) = komplex, \mathcal{S}(y) = reell, \mathcal{S}(z) = nat, \mathcal{S}(u) = Hund \ldots$
Untersorten:	$reell \sqsubseteq komplex, gaussch \sqsubseteq komplex, nat \sqsubseteq gaussch, nat \sqsubseteq reell, Hund \sqsubseteq Tier$
Funktionen:	$o : nat$
	$+ : komplex \times komplex \rightarrow komplex$
	$reell \times reell \rightarrow reell$
	$nat \times nat \rightarrow nat$
	$* : komplex \times komplex \rightarrow komplex$
	$reell \times reell \rightarrow reell$
	$nat \times nat \rightarrow nat$
	$inv : komplex \times komplex \rightarrow komplex, \ldots$
	$Fido : Hund; Tweety : Tier; Vater : Hund \rightarrow Hund \ldots$
Prädikate:	$gerade : nat; <: komplex \times komplex; brav : Hund \ldots$

Als rein formale Übung übersetzen wir folgende Formeln:

$$\mathcal{F} = \{\forall x : komplex \exists z : nat : \ (*(x, x) = z \wedge brav(Fido) \Rightarrow Alter(Fido) = z)\}$$

Es ergibt sich: $\Psi_{sig}(\Sigma) = (\{x, y, z, u, v, w \ldots\}, \{+, *, inv, o, Fido, Alter\}, \{<, gerade, brav, =, komplex, reell, nat, gaussch, Hund, Tier\})$.

Die Stelligkeiten übertragen sich: $arity(komplex) = arity(reell) = arity(gaussch) = arity(Hund) = arity(Tier) = 1$.

$$\begin{aligned}\Psi_{sig}(\Sigma) = \{&\forall x : reell(x) \Rightarrow komplex(x),\\
&\forall x : gaussch(x) \Rightarrow komplex(x),\\
&\forall x : nat(x) \Rightarrow gaussch(x),\\
&\forall x : nat(x) \Rightarrow reell(x),\\
&\forall x : Hund(x) \Rightarrow Tier(x),\\
&nat(o),\\
&\forall x, y : komplex(x) \wedge komplex(y) \Rightarrow komplex(+(x, y)),\\
&\forall x, y : reell(x) \wedge reell(y) \Rightarrow reell(+(x, y)),\\
&\forall x, y : nat(x) \wedge nat(y) \Rightarrow nat(+(x, y)),\\
&\forall x, y : komplex(x) \wedge komplex(y) \Rightarrow komplex(*(x, y)),\\
&\forall x, y : reell(x) \wedge reell(y) \Rightarrow reell(*(x, y)),\\
&\forall x, y : nat(x) \wedge nat(y) \Rightarrow nat(*(x, y)),\\
&\forall x, y : komplex(x) \Rightarrow komplex(*(x, inv(x))),\\
&Hund(Fido),\\
&\forall x : Hund(x) \Rightarrow nat(Alter(x))\}.\end{aligned}$$

Die Formel wird folgendermaßen übersetzt:

$$\begin{aligned}&\Psi_\Sigma(\forall x : komplex :\ \exists z : nat(*(x, x) = z \wedge brav(Fido) \Rightarrow Alter(Fido) = z))\\
&= \forall x : komplex(x) \Rightarrow \Psi_\Sigma(\exists z : nat(*(x, x) = z \wedge brav(Fido) \Rightarrow Alter(Fido) = z))\\
&= \forall x : komplex(x) \Rightarrow \exists z : nat(z) \wedge \Psi_\Sigma((*(x, x) = z \wedge brav(Fido) \Rightarrow Alter(Fido) = z))\\
&= \forall x komplex(x) \Rightarrow \exists z nat(z) \wedge (*(x, x) = z \wedge brav(Fido) \Rightarrow Alter(Fido) = z)\end{aligned}$$

In diesem Beispiel sind also aus einer OSPL-Formel 16 PL_1-Formeln geworden. Das deutet darauf hin, daß es besser sein könnte, mit der OSPL- Formel zu arbeiten.

Um die Korrektheit und Vollständigkeit der Übersetzung zu beweisen, müssen wir zeigen, daß jede erfüllbare Spezifikation in eine erfüllbare Spezifikation übersetzt wird (und umgekehrt).

Satz 13.10. (Korrektheit und Vollständigkeit der Übersetzung von OSPL in PL_1)
Sei $\mathcal{S} = (\Sigma, \mathcal{F})$ eine OSPL-Spezifikation und $\mathcal{S}' = (\Sigma', \mathcal{F}')$ die übersetzte PL_1 Spezifikation.
Dann ist $\mathcal{S}$ erfüllbar gdw. $\mathcal{S}'$ erfüllbar ist.

Den Beweis bietet keine echte Schwierigkeit. Er ist etwas länglich, da viele Fälle und Details beachtet werden müssen, und beide Richtungen gezeigt werden müssen.

Die Korrektheit sichert, daß wenn eine übersetzte Spezifikation widerlegt worden ist, auch die Originalformel unerfüllbar ist. D.h. Beweis durch Übersetzen und Widerlegen der negierten Behauptung ist ein korrektes Verfahren. Um nachzuweisen, daß die Methode auch immer funktioniert müssen wir noch die Vollständigkeit nachweisen, d.h. zeigen, daß Modelle für übersetzte Formeln auch immer die Existenz von Modellen für die Originalformeln sicherstellen.

Konsequenzen aus den Vollständigkeits- und Korrektheitsbeweisen für die Übersetzung in PL_1 sind:

- OSPL ist als Logik äquivalent zu PL_1. Daher kann man bestimmte Resultate wie Kompaktheit direkt übertragen.
- Formeln in OSPL sind i.a. erheblich kürzer und intuitiver als die entsprechenden Formeln in PL_1.

Daher kann man viele Sachverhalte in OSPL wesentlich besser ausdrücken. Außerdem ist zu erwarten, daß der Kalkül in OSPL effizienter ist.

13.5 Klauselnormalform für OSPL

Auf dem Weg zu einem Resolutionskalkül ist der erste Schritt die Herstellung der Klauselform. Bis auf eine Ausnahme funktioniert das wie in PL_1. Die Ausnahme ist die Skolemisierung.

Definition 13.11. Skolemisierung in OSPL
Eine OSPL Formel $\forall x_1 : A_1 \ldots \forall x_n : A_n \exists x : A : F$ wird folgendermaßen skolemisiert:

$$\forall x_1 : A_1 \ldots \forall x_n : A_n \exists x : AF \rightarrow \forall x_1 : A_1 \ldots \forall x_n : A_n F[x \rightarrow f(x_1, \ldots, x_n)]$$

und folgende Funktionsdeklaration wird der Signatur hinzugefügt:

$$f : A_1 \times \ldots \times A_n \rightarrow A$$

Der Beweis für die Korrektheit der Skolemisierung ist genau wie in PL_1 (Theorem 4.28). Die Korrektheit der Funktionsdeklaration ist offensichtlich. Da die Sorte des neuen Terms $f(x_1, \ldots, x_n)$ gleich der Sorte der Variablen x ist, garantiert die Funktionsdeklaration, daß die neue Formel $F[x \rightarrow f(x_1, \ldots, x_n)]$ auch wohlsortiert ist.

13.6 Unifikation

Zunächst wird ein allgemeiner Unifikationsalgorithmus für OSPL vorgestellt und dann werden Optimierungen für speziellere und für eingeschränkte Sortenstrukturen besprochen.

Wir geben einen Algorithmus an, der für vernünftige sortierte Signaturen korrekt und vollständig ist:

Definition 13.12. gutartige sortierte Signatur
Sei f_{sort} die Relation auf den Sorten, definiert als: $f_{sort}(S_1, \ldots, S_n) = S$ gdw. $S_1 \times \ldots \times S_n \to S$ eine Funktionsdeklaration ist. Folgendes sei erfüllt:

- *$\sqsubseteq$ ist eine partielle Ordnung auf den Sorten*
- *die kleinste Sorte eines Terms ist eindeutig bestimmt.*
- *f_{sort} ist eine (partielle) Funktion.*
- *Wenn f_{sort} auf $(S_1, \ldots, S_n)$ definiert ist und $(S'_1, \ldots, S'_n) \sqsubseteq (S_1, \ldots, S_n)$, dann ist ist f_{sort} auch auf $(S'_1, \ldots, S'_n)$ definiert.*
- *f_{sort} ist monoton auf den Sorten.*

Dann ist die sortierte Signatur gutartig.

Definition 13.13. Unifikationsalgorithmus U_S für gutartige Signaturen:

Eingabe: *zwei Terme oder Atome s_0 und t_0:*

Ausgabe: *"nicht unifizierbar" oder eine Aufzählung der allgemeinsten Unifikatoren:*

Zustände *auf denen der Algorithmus operiert: Eine Menge Γ von Gleichungen.*

Initialzustand: $\Gamma_0 = \{s_0 \stackrel{?}{=} t_0\}$.

Unifikationsregeln:

$$\frac{f(s_1,\ldots,s_n) \stackrel{?}{=} f(t_1,\ldots,t_n), \Gamma}{s_1 \stackrel{?}{=} t_1,\ldots,s_n \stackrel{?}{=} t_n, \Gamma} \qquad \textit{(Dekomposition)}$$

$$\frac{x \stackrel{?}{=} f(s_1,\ldots,s_n), x \stackrel{?}{=} f(t_1,\ldots,t_n), \Gamma}{x \stackrel{?}{=} f(t_1,\ldots,t_n), s_1 \stackrel{?}{=} t_1,\ldots,s_n \stackrel{?}{=} t_n, \Gamma} \qquad \textit{(Dekomposition)}$$

$$\frac{x \stackrel{?}{=} x, \Gamma}{\Gamma} \qquad \textit{(Tautologie)}$$

$$\frac{x \stackrel{?}{=} y, \Gamma}{x \stackrel{?}{=} y, \{x \mapsto y\}\Gamma} \quad \textit{wenn } S(y) \sqsubseteq S(x) \qquad \textit{(Ersetzung)}$$

$$\frac{t \stackrel{?}{=} x, \Gamma}{x \stackrel{?}{=} t, \Gamma} \quad t \notin V \qquad \textit{(Orientierung)}$$

$$\frac{x \stackrel{?}{=} y, \Gamma}{y \stackrel{?}{=} x, \Gamma} \quad \textit{falls } S(x) \sqsubseteq S(y) \qquad \textit{(Variablenorientierung)}$$

$$\frac{x \stackrel{?}{=} y, \Gamma}{x \stackrel{?}{=} z, y \stackrel{?}{=} z, \Gamma} \quad z \textit{ neu mit } S(z) \sqsubseteq S(x), S(z) \sqsubseteq S(y) \qquad \textit{(Variablenabschwächung)}$$

$$\frac{x \stackrel{?}{=} f(t_1,\ldots,t_n), \Gamma}{x \stackrel{?}{=} f(x_1,\ldots,x_n), \Gamma} \quad x_i \textit{ neu mit } S(x_i) = S(t_i) \qquad \textit{(Auffalten)}$$

$$\frac{x \stackrel{?}{=} f(x_1,\ldots,x_n), \Gamma}{x \stackrel{?}{=} f(y_1,\ldots,y_n), x_1 \stackrel{?}{=} y_1, \ldots x_n \stackrel{?}{=} y_n, \Gamma} \qquad \textit{(Termabschwächung)}$$

$$\textit{falls } S(f(x_1,\ldots,x_n)) \not\sqsubseteq S(x).\ y_i \textit{ neu mit } S(y_i) \sqsubseteq S(t_i)$$

Abbruchbedingungen:

$$\frac{f(\ldots) \stackrel{?}{=} g(\ldots), \Gamma}{Fail} \quad \textit{wenn } f \neq g \qquad \textit{(Clash)}$$

$$\frac{x_1 \stackrel{?}{=} f(.,x_2,.),\ldots,x_n \stackrel{?}{=} f(.,x_1,.)}{Fail} \quad \textit{d.h. Zyklus gefunden:} \qquad \textit{(occurs check)}$$

$(x \stackrel{?}{=} y) \in \Gamma$ — *Variablenabschwächung nicht anwendbar*

$(x \stackrel{?}{=} f(x_1,\ldots,x_n)) \in \Gamma$ — *Termabschwächung nicht anwendbar*

Die Verzweigungsregeln sind: Termabschwächung und Variablenabschwächung. Die Steuerung ist nicht-deterministisch. Start ist mit Γ_0. Am günstigsten ist es, mit hoher Priorität die Simplifikationsregeln zu verwenden: Dekomposition, Tautologie, Ersetzung, Orientierung, Auffalten. Die Suche wird nicht-deterministisch *in Breitensuche durchgeführt. Wenn eine Gleichungen $\{x_1 \stackrel{?}{=} t_1, \ldots, x_k \stackrel{?}{=} t_k\}$ erzeugt wird, auf die keine Regel mehr anwendbar ist und keine Abbruchbedingung zutrifft, wird die Substitution $\{x_i \mapsto t_i\ \}$ als Unifikator ausgegeben.*

Bemerkung 13.14. Die Anwendungsregel kannoptimiert werden indem man die Beschränkung auf wohlsortierte Substitutionen aufgibt. Die Vorgehensweise ist dann, zunächst Unifikation ohne Beachtung der Sorten durchzuführen, und danach erst die Sorten anhzupassen. Es können dabei zwischendurch nicht wohlsortierte Terme auftreten. Das wird wieder ausgeglichen, da eine Gleichung $x \stackrel{?}{=} t$, die für die Anwendungsregel benutzt wird, nur dann gelöst ist, wenn die Sorte von t kleiner oder gleich der Sorte von x ist. Das kann durch die Abschwächungsregeln erreicht werden.

Beispiel 13.15. Wir geben zunächst Beispiele, die die einzelnen Regeln illustrieren.

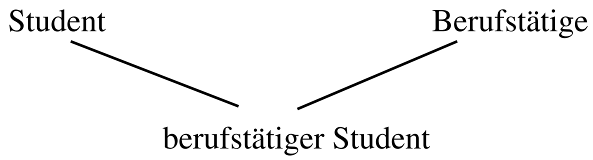
1. Sorten: $N \sqsubseteq R$

$$x : N \stackrel{?}{=} y : R$$

$$\to y : R \stackrel{?}{=} x : N \text{ (Variablenorientierung)}$$

$$\to \{y \mapsto x\}$$

2. Sortenstruktur:



$$x : \text{Student} \stackrel{?}{=} y : \text{Berufstätig}$$

$$\to x \stackrel{?}{=} z, y \stackrel{?}{=} z : \text{berufstätiger Student (Variablenabschwächung)}$$

$$\to \{x \mapsto z, y \mapsto z\}$$

4. Sorten: $nat \sqsubseteq reell$
Funktionsdeklarationen:

$$plus : reell \times rell \to reell$$

$$nat \times nat \to nat$$

$$x : nat \stackrel{?}{=} plus(u : reell, v : reell)$$

$$\rightarrow x : nat \stackrel{?}{=} plus(u_1, v_1), u \stackrel{?}{=} u_1 : nat, v \stackrel{?}{=} v_1 : nat \text{ (Termabschwächung)}$$

Bei den hier betrachteten gutartigen Sortenstrukturen haben alle Unifikationsprobleme nur endlich viele allgemeinste Unifikatoren.

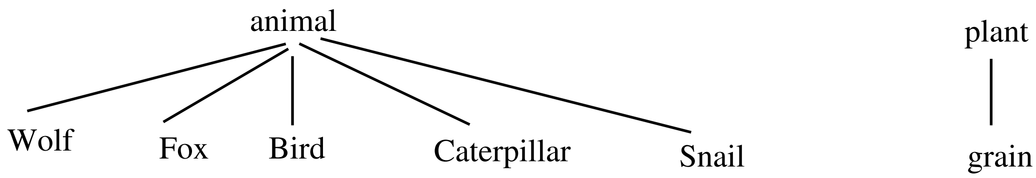
Jetzt, wo wir wissen wie die Unifikation funktioniert können wir auch Resolutionsbeweise führen. Der Unterschied in der Performanz wird verdeutlichen an einem Beispiel, das man einmal unsortiert und einmal sortiert rechnet. Es handelt sich um das als *Schuberts Steamroller* bekannte Beispiel.

Beispiel 13.16. **Schuberts Steamroller**
Zunächst die unsortierte Variante:

```
set(ur_res).
assign(max_mem,1500).
assign(max_seconds,1800).
set(free_all_mem).

formula_list(axioms).
(all x (Wolf(x) -> animal(x))).
(all x (Fox(x) -> animal(x))).
(all x (Bird(x) -> animal(x))).
(all x (Caterpillar(x) -> animal(x))).
(all x (Snail(x) -> animal(x))).
(all x (Grain(x) -> plant(x))).
(all x (animal(x) -> ((all y (plant(y) -> eats(x,y))) |
(all z ((animal(z) & Smaller(z,x) & (exists u (plant(u) & eats(z,u))))
                -> eats(x,z)))))).
(all x all y ((Caterpillar(x) & Bird(y)) -> Smaller(x,y))).
(all x all y ((Snail(x) & Bird(y)) -> Smaller(x,y))).
(all x all y ((Bird(x) & Fox(y)) -> Smaller(x,y))).
(all x all y ((Fox(x) & Wolf(y)) -> Smaller(x,y))).
(all x all y ((Bird(x) & Caterpillar(y)) -> eats(x,y))).
(all x (Caterpillar(x) -> (exists y (plant(y) & eats(x,y))))).
(all x (Snail(x) -> (exists y (plant(y) & eats(x,y))))).
(all x all y ((Wolf(x) & Fox(y)) -> -eats(x,y))).
(all x all y ((Wolf(x) & Grain(y)) -> -eats(x,y))).
(all x all y ((Bird(x) & Snail(y)) -> -eats(x,y))).
-(exists x exists y (animal(x) & animal(y) & eats(x,y) &
                    (all z (Grain(z) -> eats(y,z))))).
end_of_list.
```

Der Beweis dazu benötigt 70 Resolutions+ Faktorisierungs-schritte
Eine Version mit Sorten hat folgende Sortenstruktur:



Mit einem Trick wurde der OTTER Beweiser dazu gebracht auch mit Sorten umzugehen. Der Beweis benötigt nur noch 18 Schritte ist und ist sehr viel schneller.

13.7 Optimierungen der Unifikation in OSPL

Die Unifikation in nicht gutartigen Sortenstrukturen ist unentscheidbar und infinitär. Aber auch bei gutartigen kann die Komplexität hoch sein und die Unifikation umständlich. Weitere Einschränkungen, die zu besseren Algorithmen führen, sind:

Man läßt nur Halbverbände als partielle Ordnung der Sortenhierarchie zu. Halbverbände bedeutet in unserem Zusammenhang, daß es für je zwei Sorten maximal eine größte gemeinsame Untersorte gibt. D.h. zwei Sorten haben entweder einen glb oder keine gemeinsame Untersorte.

In diesem Fall kann man die Variablenabschwächung eindeutig durchführen.

$$x : R \stackrel{?}{=} y : S \rightarrow x \stackrel{?}{=} z : T, y \stackrel{?}{=} z : T$$

wobei $T = glb(R, S)$ ist. Wenn der glb nicht existiert, kann man abbrechen.

Die Unifikation kann unitär (d.h. mgu ist eindeutig) werden, wenn in der Signatur gilt, daß zusätzlich die Termabschwächung immer auf eindeutige Weise ausgeführt werden kann

14 Im WWW verfügbare Deduktionssysteme

Einige Emailadressen von Beweissystemen: bzw. weitere Verweise:

- Otter: http://www-unix.mcs.anl.gov/AR/otter/
- EQP: http://www-unix.mcs.anl.gov/AR/EQP/
- SPASS: http://www.mpi-sb.mpg.de/units/ag2/projects/SPASS/
- TPTP: http://www.cs.jcu.edu.au/ tptp/ http://wwwjessen.informatik.tu-muenchen.de/ tptp/