Constraintsystem Terme T

Das Constraintsystem T enthält ein Gleichheits-Constraints für Terme, und zwar für endliche Bäume und unendliche Bäume, die nur endlich viele Unterbäume enthalten (engl. rational trees). T ist wie E, aber Theorie ohne *Azyklizität*.

Signatur von T:
Unendlich viele Funktionssymbole (mind. eine Konstante)
Constraintsymbole $\doteq$, *true* und *false*.

Constrainttheorie von T:
Clark'sche Gleichheitstheorie CET ohne *Azyklizität*.

Erlaubte Constraints von T:

$$C \quad ::= \quad s \doteq t \quad | \quad true \quad | \quad false \quad | \quad C \wedge C$$

s und t sind Terme über der Signatur von T.

Stand 1999

Anmerkung

Die Theorie von T ist entscheidbar.

Clarksche Gleichheitstheorie CET

Definition 1 Sei Σ eine Signatur mit unendlich vielen Funktionssymbole, davon mindestens eine Konstante. Die *CET* ist die Menge aller Sätze nach folgenden Schemata:

Reflexivität:
$\forall(true \rightarrow x \doteq x)$
Symmetrie:
$\forall(x \doteq y \rightarrow y \doteq x)$
Transitivität:
$\forall(x \doteq y \wedge y \doteq z \rightarrow x \doteq z)$
Verträglichkeit:
$\forall(x_1 \doteq y_1 \wedge \ldots \wedge x_n \doteq y_n \rightarrow f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n))$
Zerlegung:
$\forall(f(x_1, \ldots, x_n) \doteq f(y_1, \ldots, y_n) \rightarrow x_1 \doteq y_1 \wedge \ldots \wedge x_n \doteq y_n)$
Widerspruch:
$\forall(f(x_1, \ldots, x_n) \doteq g(y_1, \ldots, y_m) \rightarrow false)$ falls $f \neq g$ oder $n \neq m$
Azyklizität:
$\forall(x \doteq t \rightarrow false)$ falls t keine Variable ist und x in t vorkommt

Anmerkung

Die ersten drei Sätze sagen aus, daß $\doteq$ eine Äquivalenzrelation ist.

n bzw. m können 0 sein. Falls $n = 0$ wird die entstehende leere Konjunktion bei *Zerlegung* und *Verträglichkeit* als *true* gelesen. Die beiden Sätze sind dann aber redundant.

Constraintsystem Terme T

Das Constraintsystem T ist wie E, aber mit einer um *Azyklizität* reduzierten Theorie.

Definition 2 Eine Konjunktion von erlaubten Constraints von T heißt *gelöst (oder in Normalform)*, wenn sie entweder *false* ist oder die Form $x_1 \doteq t_1 \wedge \ldots \wedge x_n \doteq t_n$ hat, wobei $n \geq 0$ und die Variablen $x_1, \ldots, x_n$ paarweise verschieden sind und jede Variable x_i ungleich t_j ist, wenn $i \leq j$.

Anmerkung

Ein Constraintlöser für E kann mit (quasi-)linearer Zeitkomplexität in der Größe der beteiligten Terme implementiert werden, obwohl in der Praxis oft ein einfacherer, aber quadratischer Algorithmus vorgezogen wird, weil man annimmt, daß seine durchschnittliche Laufzeit besser ist.

Die meisten Prolog-Systeme verzichten aus Effizienzgründen bei der Unifikation auf einen Test Azyklität (engl. occur-check), der durch das letzte Axiomenschema *Azyklizität* gefordert ist. Dadurch gewinnt man an Effizienz, allerdings können Terminierungsprobleme auftreten.

Im Gegensatz zu der Normalform von CET ist es wegen des fehlenden Tests auf Azyklizität nicht mehr notwendig, das Vorkommen der Variablen $x_1, \ldots, x_n$ in jenen Termen $t_1, \ldots, t_n$ zu verbieten, die keine Variablen sind.

Terme T

```
reflexivitaet @ XT ≐ XT <=> true.
orientierung  @ XT ≐ X  <=> X @< XT |
                X ≐ XT.
zerlegung     @ T1 ≐ T2 <=>
                nonvar(T1),nonvar(T2) |
                gleicher_funktor(T1,T2),
                T1=..[F|L1], T2=..[F|L2],
                gleiche_args(L1,L2).
konfrontation @ X ≐ T1, X ≐ T2 <=>
                T1 @=< T2 |
                X ≐ T1, T1 ≐ T2.

    gleiche_args([],[]) <=> true.
    gleiche_args([X|L1],[Y|L2]) <=>
                X ≐ Y,
                gleiche_args(L1,L2).
```

Anmerkung

Ordnungsrelation (als Prädikate `@<` und `@=<`) auf Termen gibt, bei der ungebundene Variablen immer kleiner als alle anderen Terme sind. Diese Ordnung ist notwendig, um Terminierungsprobleme zu vermeiden.

Die CHR-Regeln sind so gewählt, daß sie Gleichheitsconstraints zur Normalform vereinfachen können. Im wesentlichen wurde dabei versucht, die Implikationen in der Theorie zu logischen Äquivalenzen zu verstärken, damit sich daraus Simplifikationsregeln gewinnen lassen.

Symmetrie läßt sich in beide Richtungen anwenden. Die zugehörige Regel `orientierung` hat zusätzlich einen Wächter, der Terminierung sicherstellt.

Verträglichkeit, *Zerlegung* und *Widerspruch* wurden in der Regel `zerlegung` zusammengefaßt: Das Prädikat `gleicher_funktor/2` ist genau dann erfolgreich, wenn die Terme `T1` und `T2` das gleiche Funktionssymbol haben. Das Hilfsprädikat `=..` zerlegt einen Term `f(t1,...,tn)` in eine Liste mit seinem Funktionssymbol und seinen Argumenten `[f,t1,...,tn]`.

Die Regel `konfrontation` ist eine Folgerung aus den Axiomenschemata *Reflexivität* und *Transitivität*.

$$\begin{array}{ll}
 & \texttt{h(Y,f(a),g(X,a))} \doteq \texttt{h(f(U),Y,g(h(Y),U))} \\
\mapsto_{zerlegung} & \texttt{gleiche_args([Y,f(a),g(X,a)],[f(U),Y,g(h(Y),U)])} \\
\mapsto^{*} & \texttt{Y} \doteq \texttt{f(U), f(a)} \doteq \texttt{Y , g(X,a)} \doteq \texttt{g(h(Y),U)} \\
\mapsto_{orient} & \texttt{Y} \doteq \texttt{f(U), Y} \doteq \texttt{f(a) , g(X,a)} \doteq \texttt{g(h(Y),U)} \\
\mapsto^{*}_{zerlegung} & \texttt{Y} \doteq \texttt{f(U), Y} \doteq \texttt{f(a) , X} \doteq \texttt{h(Y), a} \doteq \texttt{U} \\
\mapsto_{orient} & \texttt{Y} \doteq \texttt{f(U), Y} \doteq \texttt{f(a) , X} \doteq \texttt{h(Y), U} \doteq \texttt{a} \\
\mapsto_{konfrontation} & \texttt{Y} \doteq \texttt{f(U), f(U)} \doteq \texttt{f(a) , X} \doteq \texttt{h(Y), U} \doteq \texttt{a} \\
\mapsto^{*}_{zerlegung} & \texttt{Y} \doteq \texttt{f(U), U} \doteq \texttt{a , X} \doteq \texttt{h(Y), U} \doteq \texttt{a} \\
\mapsto_{konfrontation} & \texttt{Y} \doteq \texttt{f(U), U} \doteq \texttt{a , X} \doteq \texttt{h(Y), a} \doteq \texttt{a} \\
\mapsto_{reflexivitaet} & \texttt{Y} \doteq \texttt{f(U), U} \doteq \texttt{a , X} \doteq \texttt{h(Y)}
\end{array}$$

Anmerkung

Das folgende Beispiel zeigt, wie Gleichheitsconstraints zur Normalform vereinfacht werden, wobei wir die Behandlung des Constraints `gleiche_args/2` der Einfachheit halber übergehen:

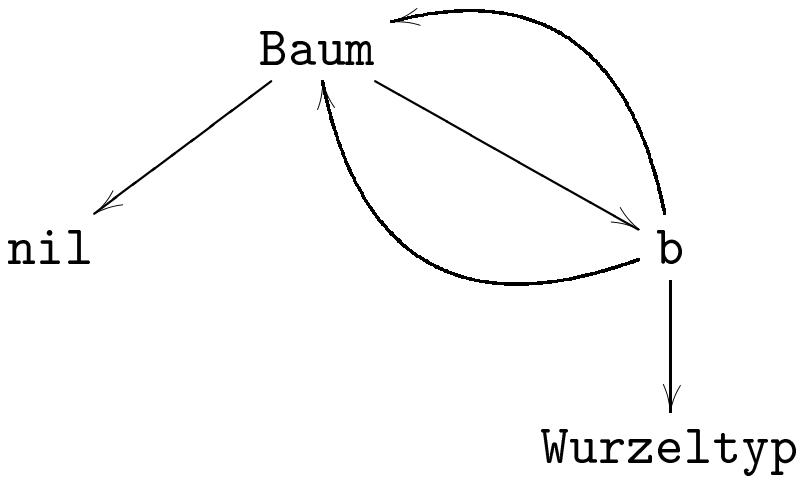
`X` ist also gleich `h(f(a))`.

Anwendungsbeispiel: Typüberprüfung

Wir definieren den Typ des binären Baumes in BNF-artiger Syntax durch einen zyklischen Term, der durch folgendes Constraint definiert wird:

```
baum(Baum,Wurzeltyp) :-
      Baum ≐ nil | b(Baum,Wurzeltyp,Baum).
```

"Ein Baum ist entweder die Konstante `nil` oder das dreistellige Funktionssymbol `b` angewendet auf einen Baum, einen Term vom Typ `Wurzeltyp` und einen Baum"



Anmerkung

A. Colmerauer hat mit der LP-Sprache Prolog II um 1982 erkannt, daß durch den Verzicht auf einen Test auf Azyklität unendliche, zyklische Terme[1] (engl. rational trees) beschrieben werden können und das dafür die Constrainttheorie RT vollständig ist. Zum Beispiel definiert die Gleichheit `X` $\doteq$ `f(X)` einen zyklischen Term. Mit zyklischen Termen lassen sich elegant Grammatiken für Parser und Zustandsübergänge für Automaten realisieren. Eine Anwendung zyklischer Terme zur Darstellung und Überprüfung von Typen zeigt das folgende Beispiel.

[1] Nicht jeder unendliche Term ist auch zyklisch. Zyklische Terme sind unendliche Terme mit einer endlichen Anzahl von Untertermen.

Anwendungsbeispiel: Typüberprüfung

```
pruefe(Term,Typ1 | Typ2):- pruefe(Term,Typ1).
pruefe(Term,Typ1 | Typ2):- pruefe(Term,Typ2).
pruefe(Term,Typ):- gleicher_funktor(Term,Typ),
                   Term=..[F|Args],Typ=..[F|Typen],
                   pruefe_args(Args,Typen).

    pruefe_args([],[]).
    pruefe_args([A|Args],[Typ|Typen]) :-
                   pruefe(A,Typ),
                   pruefe_args(Args,Typen).
```

`baum(ZiffernBaum, (0 | 1 | ... | 9))`
definiert einen binären Baum über Ziffern.

Typüberprüfung eines Baumes mit dem Ziel
`baum(ZiffernBaum, (0 | 1 | ... | 9)),`
`pruefe(b(nil,3,b(nil,0,nil)),ZiffernBaum).`

Anmerkung

Wenn wir einen endlichen Term daraufhin überprüfen wollen, ob es sich um einen binären Baum handelt, genügt es, den aus obigem Constraint entstandenen Typterm `Baum` entlangzugehen. Dies bewerkstelligt das folgende Prädikat `pruefe(Term,Typ)`: