Feature Terme *FT*

Das Constraintsystem Feature Terme *FT* erlaubt die deklarative Verwendung von Verbunden (engl. records). Selektoren (Attribute) werden als *Features* bezeichnet, der Typ-Name des Verbunds als *Sorte*. Da *FT* Verbunde modelliert, ist es sehr einfach.

Wie Herbrand Terme sind Feature Terme als Bäume (engl. trees) repräsentierbar. Und zwar als endliche Bäume oder unendliche Bäume, die nur endlich viele Unterbäume enthalten. Diese Bäume sind durch endliche gerichtete, u.U. zyklische, Graphen darstellbar.

Stand 1999

Keine Anmerkung

Constraintsystem
Feature Terme FT

Signatur
Keine Funktionssymbole
Endlich viele unäre und binäre
Constraintsymbole (Sorten und Features),
sowie $\doteq$, *true* und *false*.

Constrainttheorie
(a) $\doteq$ ist eine "Aquivalenzrelation
(b) $X\ f\ Y \wedge X\ f\ Z \Rightarrow Y \doteq Z$
(c) $a(X) \wedge b(Y) \Rightarrow false$ wenn $a \neq b$

Erlaubte Constraints

$$C \quad ::= \quad s(X) \ \mid \ X\ f\ Y \ \mid \ X \doteq Y \ \mid$$
$$true \ \mid \ false \ \mid \ C \wedge C$$

X und Y sind Variablen, s ist ein
Sortensymbol, f ein Featuresymbol in
Infix-Schreibweise.

Anmerkung

Im einfachsten Fall kommt FT ohne Konstanten aus. Denn $X \doteq k$ kann durch die einelementige Sorte k als $k(X)$ geschrieben werden.

Man könnte eigentlich auch auf Sorten verzichten - mit einem vordefinierten Feature *sort*:
$s(X) \Leftrightarrow X\,sort\ s$.

$X\ f\ Y$ kann in konkreter Syntax z.B. als `X.f = Y` geschrieben werden.

Lösungs-Normalform

Lösungs-Normalform (engl. solved form) für FT-Constraints:
(1) Kein atomares Constraint kommt zweimal vor
(2) Kommt $X \doteq Y$ in einem Constraint vor, so kommt Y nirgendwo sonst vor.
(3) Die Constrainttheorie CT wird respektiert.

In der Lösungs-Normalform wird jedes inkonsistente FT-Constraint durch $false$ repräsentiert.

Keine Anmerkung

Beispiel Normalform

$person(X) \wedge X\ name\ Y \wedge leo(Y) \wedge X\ wohnt\ Z \wedge$
$münchen(Z) \wedge X\ name\ V \wedge sepp(V)$
Aus CT folgt $Z \doteq V$,
wegen (2) ersetzt man V durch Z,
wegen (1) entfernt man eins der beiden
$X\ name\ Y$,
In CT wegen $leo(Y)$ und $sepp(Y)$ ergibt sich:
$false$.

Anmerkung

$person(X) \wedge X\ name\ Y \wedge leo(Y) \wedge X\ wohnt\ Z \wedge münchen(Z)$
kann in konkreter Syntax z.B. auch geschrieben werden als: `X:person(name=leo,wohnt=münchen)`

Constrainlöser

$\doteq$ wird durch `eq` dargestellt, $s(X)$ durch `X:s`, $X\ f\ Y$ durch `X.f.Z`.
Der Constraintlöser verwendet die eingebaute syntaktische Gleichheit (Unifikation) mit `=` für (a) in CT.

```
equal @ X eq Y <=> X=Y.
feature @ X.F.Y \ X.F.Z <=> Y=Z.
sort @ X:S1 \ X:S2 <=> S1=S2.
```

`equal`: Implementiert Variablen-Elimination laut (2) der Normalform (NF).
`feature`: Implementiert Feature Decomposition laut (b) von CT und (1) von NF.
`sort`: Implementiert Sort Intersection laut (c) von CT und (1) von NF.

Anmerkung

Dieser Constraintlöser funktioniert auch, wenn Variablen durch Konstanten gebunden werden.

Erweiterung um Negation

Wir erweitern FT um die erlaubten Constraints:
$\neg s(X)$
$\neg X\ f$ (abkürzend für $\forall Y \neg(X\ f\ Y)$)

Die Constrainttheorie wird erweitert um:
$s(X) \wedge \neg s(X) \Rightarrow false$
$X\ f\ Y \wedge \neg X\ f \Rightarrow false$

Der Constraintlöser wird erweitert um:

```
X:S, not X:S <=> false
X.F.Z, not X.F <=> false
```

Keine Anmerkung

Erweiterung um Aritäts-Constraint

Man möchte Herbrand Terme darstellen.

Idee: $X \doteq g(Y, Z)$ darstellen durch
$g(X) \wedge X\ 1\ Y \wedge X\ 2\ Z$.

Aber: Zusätzliche Features sind möglich, z.B.
$X\ 3\ V$.

Daher: Erlaubte Features explizit durch sog.
Aritäts-Constraint $X\{f_1, \ldots, f_n\}$ definieren,
z.B. $X\{1, 2\}$.

Auch nützlich, um Definition eines Feature
Terms explizit abzuschliessen, z.B.
$X\{name, wohnt\}$.

Keine Anmerkung

Aritäts-Constraint, Fortsetzung

Constrainttheorie wird erweitert um:
$X\ F \wedge X\ G \Rightarrow false$ wenn $F \neq G$
$X\ F \wedge X\ f\ Y \Rightarrow false$ wenn $f \notin F$

Constraintlöser wird erweitert um:

```
X in Fs \ X in G <=> Fs=G.
X in Fs, X.F.Y ==> member(F,Fs).
```

Anmerkung

Andere Erweiterung:
Order Sorted Feature Terms (OSF) von Ait-Kaci et. al. 1993:
Mit Sorten-Hierarchie.

Anwendungsbeispiel: Linguistik

Feature Terme werden traditionell zur Wissensrepräsentation in der Sprachwissenschaft eingesetzt, d.h. um den Inhalt von Sätzen zu beschrieben.

$$X = \exists Y \left[\begin{array}{l} \text{Person} \\ \text{office} : Y \\ \text{home} : Y \\ \text{name} : \left[\begin{array}{l} \text{first} : Leo \\ \text{last} : Smith \end{array}\right] \end{array}\right]$$

Keine Anmerkung

Terminological Reasoning Description Logics

Reichhaltiger Formalismus zur Wissensrepräsentation.
Begann mit Brachmanns KL-ONE System und seinen Nachfolgern in den 80er Jahren.
Theoretische Aufarbeitung durch Nebel u.a. in den 90er Jahren.

Es gibt *Konzepte* (unäre Constraints) und *Rollen* (binäre Constraints). Darstellung durch Graphen: Konzepte sind Knoten, Rollen sind Kanten.
Im Unterschied zu Feature Termen: relationale Rollen statt funktionaler Features, impliziert definierte Konzept-Hierarchie (Taxonomien) statt einfacher Sorten. Konzept-Hierarchie ermöglicht Mehrfachvererbung. Darstellung im Graphen durch spezielle Kanten.

Anmerkung

Konzepte sind immer verträglich, Feature-Sorten schließen sich immer aus.

Es gibt inzwischen unzählige Varianten, Implementierungen und theoretische Untersuchungen z.B. über die Mächtigkeit der Varianten.

Wissensrepräsentation

Man unterscheidet zwei Arten von Wissen:

Terminologisches Wissen (T-Box)
Allgemeines Wissen des Anwendungsbereiches.
Konzepte werden implizit durch ihre Rollen-Beziehungen und andere Konzepte definiert.
Intensionale Darstellung (vgl. Datenbankschema).

Assertionales Wissen (A-Box
Konkretes Wissen, tatsächlich existierende *Individuen* (Objekte).
Explizite Angabe von Konzept- und Rollen-Beziehungen zwischen Individuen.
Exensionale Darstellung (vgl. Datenbankextension).

Im Constraintsystem: T-Box durch Regeln, A-Box durch Anfragen repräsentiert.

Keine Anmerkung

Constraintsystem Description Logic *DL*

Signatur
Unendliche viele Konstanten für Individuen-, Konzept- und Rollennamen.
Funktionssymbole *and, or, not, every, some, is* bilden Konzeptterme mit Konzepten und Rollen.
Constraintsymbole: *isa* und :

Erlaubte Constraints

$$C \quad ::= \quad I : S \quad | \quad (I, J) : r \quad | \quad c \; isa \; S \quad |$$
$$true \quad | \quad false \quad | \quad C \wedge C$$

I, J sind Variablen bzw. Individuen-Konstanten, c Konzeptnamen, r Rollennamen, S Konzeptterme.
$I : S$ heißt *membership*, $(I, J) : r$ *role filler*, zusammen bilden sie die A-Box. Die Konzeptdefinitionen $c \; isa \; S$ bilden die T-Box.

Keine Anmerkung

Man beachte: T-Box ist variablenfrei und individuenfrei.

$cisaS$ im einfachsten Fall zyklenfrei, d.h. C ist weder direkt noch indirekt durch sich selbst definiert. Mit Zyklen sind erweiterte Algorithmen notwendig, um Terminierung sicherzustellen. Bsp. Zyklus:
`male isa not female.`
`female isa not male.`

Constrainttheorie *DL*

$I : not\ S \Leftrightarrow \neg(I : S)$
$I : S_1\ and\ S_2 \Leftrightarrow I : S_1 \wedge I : S_2$
$I : S_1\ or\ S_2 \Leftrightarrow I : S_1 \vee I : S_2$
$I : some\ R\ is\ S \Leftrightarrow \exists J((I, J) : R \wedge J : S)$
$I : every\ R\ is\ S \Leftrightarrow ((I, J) : R \Rightarrow J : S)$
$C\ isa\ S \Leftrightarrow (I : C \Leftrightarrow I : S)$
sowie die T-Box (Konzept-Definitionen) des jeweiligen Anwendungsbereichs.

Anmerkung

$I : every \ldots$ gilt auch, wenn kein $(I, J) : R$-Constraint vorhanden ist.

Daher Erweiterung denkbar, die sicherstellt, dass mindestens ein Rollen-Constraint existiert:
$I : all \ R \ is \ S \Leftrightarrow I : every \ R \ is \ S \wedge I : some \ R \ is \ S$

DL ist auch um funktionale Rollen (Features, Attribute, Selektoren) erweiterbar. D.h. DL kann FT darstellen.

DL kann um transitive Rollen-Beziehungen erweitert werden:
$grandfatherisafatheroffather$ bedeutet
$(I, J) : grandfather \Leftrightarrow (I, K) : father \wedge (K, J) : father$

Constraintlöser *DL*

```
I:S1 and S2 <=> I:S1, I:S2.
I:S1 or S2 <=> I:S1 ; I:S2.
I:some R is S <=> (I,J):R, J:S.
I:every R is S, (I,J):R ==> J:S.

I:not S, I:S <=> false.
```
Clash Rule
```
I:not not S <=> I:S.
I:not (S1 and S2) <=> I:not S1 or not S2.
I:not (S1 or S2) <=> I:not S1 and not S2.
I:not (some R is S) <=> I:every R is not S.
I:not (every R is S) <=> I:some R is not S.

I:C <=> I:S.
I:not C <=> I:not S.
```
jeweils für jede Konzept-Definition *C isa S*.

Anmerkung

`I:S`-Constraints werden zerlegt.

Nach kompletter Anwendung der Regeln gilt: Antwort entweder false oder die Antowrt besteht aus:
- `I:C` bzw. `I:not C`, `C` primitives Konzept
- `I:every...`
- Rollen-Constraints `(I,J):R`

Variante: Falls T-Box/Konzept-Definitionen ebenfalls in Anfrage:

```
C isa S \ I:C <=> I:S.
C isa S \ I:not C <=> I:not S.
```

Beispiel

```
male isa not female.
parent isa human and some child is human.
mother isa parent and female.
proud_parent isa parent and every child is phd.
```

Nicht definierte Konzepte nennt man *primitiv*,
hier `female` und `human`.
Man beachte:
Implizite Definition der Konzept-Hierarchie
(Taxonomie), hier `parent<human`,
`mother<parent`, `mother<female`,
`proud_parent<parent`.

Anmerkung

Ohne erste Definition ist `X:female, X:male` konsistent.

Weitere Regel: `human isa female or male.`, dann ist nur mehr `female` primitives Konzept.

Beispiel, Fortsetzung

- Anfrage:
`sue:proud_parent, (sue,joe):child,`
`joe:not phd.`
- Entfalten der Konzept-Definition `proud_parent`
`sue:parent and every child is phd,...`
- Entf. Konzept-Definition, Zerlegung von `and`
`sue:human, sue:some child is human,`
`sue:every child is phd,...`
- Vereinfachen von *exists-in restriction* `some`
`sue:human, (sue,X):child, X:human, sue:every child is phd,...`
- Propagierung mit *value restriction* `every`
`X:phd, joe:phd, sue:human, (sue,X):child, X:human, sue:every child is phd, (sue,joe):child, joe:not phd.`
- *Clash* zwischen `joe:phd` und `joe:not phd`
`false`

Anmerkung

`X` kann `joe` sein, muß aber nicht

Anwendungen von *DL*

Z.B. zur Konfiguration technischer Systeme (z.B. PC, Peripherie) und Prozesse. A. Wolf von GMD FIRST in Berlin hat 1996 auf der Leipziger Innovationsmesse ein auf CHR basierendes *DL* Konfigurations-System names ConTeS, mit ausführbarer Spezifikationssprache TRLC, vorgestellt.

Z.B. zur Kategorisierung von Web-Seiten-Inhalten.

Z.B. zur Darstellung medizinischer Terminologie (z.B. Krankheiten).

Anmerkung

ENDE