Constraintsystem

Definition 1 Ein *Constraintsystem* ist ein Tripel $(\Sigma, CT, \mathcal{C})$. Dabei sind

- Σ eine Signatur, wobei
  - $true \in \Sigma$ und $false \in \Sigma$,
- die *Constrainttheorie* CT eine nichtleere, konsistente Theorie über Σ, und
- die *erlaubten Constraints* $\mathcal{C}$ eine Menge von Formeln über Σ, wobei
  - $true \in \mathcal{C}$ und $false \in \mathcal{C}$
  - $\mathcal{C}$ abgeschlossen ist unter Konjunktion und Variablenumbenennung (Variantenbildung)

Stand Juni 1999

Anmerkung

Im Höhefeld-Smolka-Schema wird in einem Constraintsystem angegeben, um welche Constraintsymbole es sich handelt, wie die Constraintsymbole definiert sind, und welche Constraints erlaubt sind.

Die Definition der erlaubten Constraints $\mathcal{C}$ stellt sicher, daß wir mit Konjunktionen von Constraintatomen bzw. Constraints arbeiten können, d. h., daß wir Constraints als Ziele in den Rümpfen von Klauseln verwenden können. Gleichzeitig schränkt $\mathcal{C}$ die Formeln so ein, daß eine effiziente Implementierung des zugehörigen Constraintlösers möglich ist.

Beispiel Constraintsystem E

Das Constraintsystem E (engl. Herbrand system) stellt Constraints für die Gleichheit von Termen, d.h. endlichen Bäumen (engl. finite trees), zur Verfügung.

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

Constrainttheorie von E:
Clark'sche Gleichheitstheorie CET.

Erlaubte Constraints von E:

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

wobei s und t Terme über der Signatur von E sind.

Anmerkung

Die Theorie von E ist entscheidbar.

Im folgenden gehen wir davon aus, daß die Signatur Σ auch das Symbol $\doteq/2$ enthält. Dementsprechend fügt man die zugehörige Constrainttheorie CET zu CT und die Gleichheit zwischen bestimmten Termen als erlaubtes Constraint zu $\mathcal{C}$ hinzu.

Theorem 1 *Seien t, s zwei Terme. Dann gilt:*

$$t, s \quad \text{unifizierbar}$$

gdw.

$$\exists X(X \doteq t \wedge X \doteq s) \quad \text{erfüllbar}$$

gdw.

$$H_\Sigma \models \exists X(X \doteq t \wedge X \doteq s)$$

$\lhd$

Clarksche Gleichheitstheorie CET

Definition 2 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.

Erwünschte Eigenschaften

Definition 3 Eine Theorie $\mathcal{T}$ heißt *vollständig* (engl. complete), falls für jede abgeschlossene Formel φ gilt: $\mathcal{T} \models \varphi$ oder $\mathcal{T} \models \neg\varphi$. $\lhd$

Definition 4 Ein Constraintsystem $(\Sigma, CT, \mathcal{C})$ heißt *erfüllbarkeitsvollständig* genau dann, wenn entweder $CT \models \exists C$ oder $CT \models \neg\exists C$ für alle Constraints $C \in \mathcal{C}$ gilt. $\lhd$

Anmerkung

Als Constrainttheorie CT hat man gerne eine *vollständige Theorie*. Es reicht aber aus, wenn CT eine schwächere Vollständigkeitsbedingung erfüllt, da wir es in unseren Kalkülen nur mit erlaubten Constraints (existentiell abgeschlossene Konjunktionen von atomaren Constraints) zu tun haben.

Beispiel Vollständigkeit

Die Gleichheitstheorie CET des Constraintsystems E ist vollständig. Wenn die Signatur von E nur noch endlich viele Funktionssymbole (davon mindestens eine Konstante) enthielte, dann wäre CET nicht mehr vollständig, aber erfüllbarkeitsvollständig. [Maher93].

Gegeben seien die Funktionssymbole a und f. Betrachte
$CET \models \exists X, Y \ (X \doteq a \vee X \doteq f(Y))$

Anmerkung

Endliche Disjunktion aller möglichen Variablenbelegungen gleich true.

Unabhängigkeit

Definition 5 Ein Constraintsystem $(\Sigma, CT, \mathcal{C})$ heißt *unabhängig (engl. independent of negated constraints)* genau dann, wenn für alle Constraints $C, C_1, \ldots, C_n \in \mathcal{C}$ gilt:

$$CT \models \exists C \wedge \neg C_1 \wedge \ldots \wedge \neg C_n \text{ gdw } CT \models \exists C \wedge \neg C_i$$

$$\text{für alle } i = 1, \ldots, n.$$

$\lhd$

Anmerkung

Eine weitere wichtige Eigenschaft von Constraintsystemen ist die *Unabhängigkeit*. Sie erlaubt es, allgemeine Formeln über Constraints einfacher zu behandeln (siehe auch Abschnitt Constrainterweiterungen).

Die Unabhängigkeit besagt also, daß in einem unabhängigen Constraintsystem eine Disjunktion genau dann gültig ist, wenn zumindest eins der Disjunkte gültig ist bzw. eine Konjunktion mit negierten Constraints genau dann erfüllbar, wenn jedes negierte Constraint für sich mit allen positiven Constraints erfüllbar ist.

Beispiel Unabhängigkeit

Das Constraintsystem E ist unabhängig.

E mit nur endlich vielen Funktionssymbolen ist nicht unabhängig:

$CET \models \exists X, Y\ (X \doteq f(Y) \wedge \neg Y \doteq a)$ und
$CET \models \exists X, Y, Z\ (X \doteq f(Y) \wedge \neg Y \doteq f(Z))$, aber
$CET \not\models \exists X, Y, Z\ (X \doteq f(Y) \wedge \neg Y \doteq a \wedge \neg Y \doteq f(Z))$

Keine Anmerkung

Beispiel Unabhängigkeit

Lineare Gleichungssysteme über den rationalen oder reellen Zahlen bilden ein unabhängiges Constraintsystem.

Ein Constraintsystem, das eine totale Ordnungsrelation $\leq$ für Zahlen zu Verfügung stellt, ist nicht unabhängig.

$CT \models \exists X, Y\ \neg(X \leq Y)$ und
$CT \models \exists X, Y\ \neg(Y \leq X)$ aber
$CT \not\models \exists X, Y\ (\neg(X \leq Y) \wedge \neg(Y \leq X))$.

Anmerkung

$CT \models \exists X, Y\ \neg(X < Y)$,
$CT \models \exists X, Y\ \neg(X \doteq Y)$ und
$CT \models \exists X, Y\ \neg(Y < X)$ aber
$CT \not\models \exists X, Y\ (\neg(X < Y) \wedge \neg(X \doteq Y) \wedge \neg(Y < X))$.

Aufgaben eines Constraintlösers

- Konsistenzprüfungen
- Erzeugen von Konsequenzen
- Vereinfachen von Constraints
- Berechnen von allgemeinen Lösungen

Berechnungsdienste (engl. reasoning services)

Keine Anmerkung

Berechnungsdienste I

- Konsistenztest (Erfüllbarkeitstest) (engl. satisfiability, consistency):
  $CT \models \exists C$
  z.B. $X > 3$, $X > Y$, $X > X$.

- Simplifikation (Vereinfachung): Wenn
  $CT \models \forall(C \leftrightarrow D)$ und D „einfacher“ ist als C, dann ersetze C durch D.
  z.B. $2 * X = 6$, $X^2 = X$, $X^2 < X$.

- Determination: Wenn
  $CT \models \forall(C \rightarrow X = t)$, d.h. eine Variable nur noch einen einzigen Wert annehmen kann, dann füge Gleichheit hinzu.
  z.B. $X \leq 2 \wedge 2 \leq X$.

Anmerkung

Ein *Constraintlöser* implementiert die Constraintheorie CT und liefert ein deterministisches Entscheidungsverfahren für ein Constraintsystem. Ein Constraintlöser sammelt die aus dem Programmablauf resultierenden Constraints, prüft ihre Erfüllbarkeit und vereinfacht und löst die Constraints.

Simplifikation:
Meistens wird D eine *Normalform* von C sein.
Wenn ein Constraintlöser ein neu ankommendes Constraint zusammen mit den bisherigen Constraints vereinfachen kann, ohne die Vereinfachung der bisherigen Constraints zu wiederholen, sprechen wir davon, daß der Constraintlöser inkrementell ist. *Inkrementalität* wichtig für effiziente CLP-Sprachen. Eine Normalform erleichtert es, inkrementell zu sein.

Determination:
Damit koexistierende Constraintlöser Informationen über gemeinsame Variable austauschen können. Auch nützlich für die informative Darstellung einer Antwort. Alle determinierten Variablen sollen explizit durch syntaktische Gleichheiten $\doteq$ ihren Wert erhalten. Lösen heißt Normalform erzeugen, die nur aus Gleichheiten besteht.

Berechnungsdienste II

- Projektion:
  Existenziell quantifizierte Variablen eliminieren.
  $CT \models \forall(\exists X C \leftrightarrow D)$
  z.B. $\exists Y\ (X \leq Y \wedge Y \leq Z)$,
  $\exists Y\ (X \doteq f(Y) \wedge Y \doteq a)$,
  $\exists Y\ (X \doteq f(Y))$ vs. $\exists X\ (X \doteq f(Y))$.

- Folgerungstest (engl. entailment):
  $CT \models \forall(C \rightarrow D)$ und
  $CT \models \forall(C \rightarrow \neg D)$
  z.B. $X > Y \rightarrow X \geq Y$, $X = Y \rightarrow X \leq Y$,
  $X \geq Y \rightarrow X > Y$.

- Negation: Implizit durch eigene Constraintsymbole (z. B. $=$ und $\neq$, $<$ und $\geq$), seltener durch expliziten Operator ($\neg$).

Anmerkung

Projektion:
Für schöne Antworten und kleine Normalformen. Aber je nach Constraintsystem und Algorithmus nicht immer (effizient) möglich.

Folgerungstest:
Er ist erfolgreich ist, wenn $CT \models \forall(C \rightarrow D)$ gilt, er scheitert, wenn $CT \models \forall(C \rightarrow \neg D)$ gilt, ansonsten wird der Test *verzögert*, bis er entschieden werden kann.
Für die Implementierung bedingter Anweisungen und nebenläufiger Constraint-Programmiersprachen.

Negation:
Die Negation ist i. A. nur effizient behandelbar, wenn das Constraintsystem unabhängig ist, weil dann jedes negierte Constraint für sich entschieden werden kann.

Implikation:
$C \rightarrow D$ wenn $\neg(C \wedge \neg D)$ d.h. $loese(C \wedge \neg D)$.

Alle Berechnungsdienste auf Basis des Vereinfachens mit Normalformen implementierbar.

Constraintlöser mit allen Berechnungsdiensten oft zu komplex oder ineffizient oder unmöglich.

Erwünschte Eigenschaften

- *korrekt*: Wenn $löse(C) = D$, dann $CT \models C \leftrightarrow D$.
- *vollständig*: Wenn $CT \models \neg C$, dann $löse(C) = false$.
- *monoton*: Wenn $löse(C) = false$, dann $löse(C \wedge D) = false$.
- *unabhängig von Variablennamen*: Wenn C Variante von D, dann $löse(C) = löse(D)$.
- *respektiert Kongruenz*: $löse(C \wedge D) = löse(D \wedge C)$ etc.
- *kanonisch*: Wenn $C \leftrightarrow D$, dann $löse(C) = löse(D)$.
- *inkrementell*: Aufwand f. $löse(löse(C) \wedge D)$ nicht grösser als Aufwand für $löse(C \wedge D)$.

Anmerkung

Constraintlöser implementiert (Normalisierungs-)Funktion *löse* von Constraints zu (einfacheren) Constraints.

Berechnungsdienste:

Gleichheit:
trivial, wenn kanonisch.

Implikation:
wenn kanonisch: $C \rightarrow D$ wenn $C \leftrightarrow (C \wedge D)$ d.h.
$löse(C \wedge D) = löse(C)$.