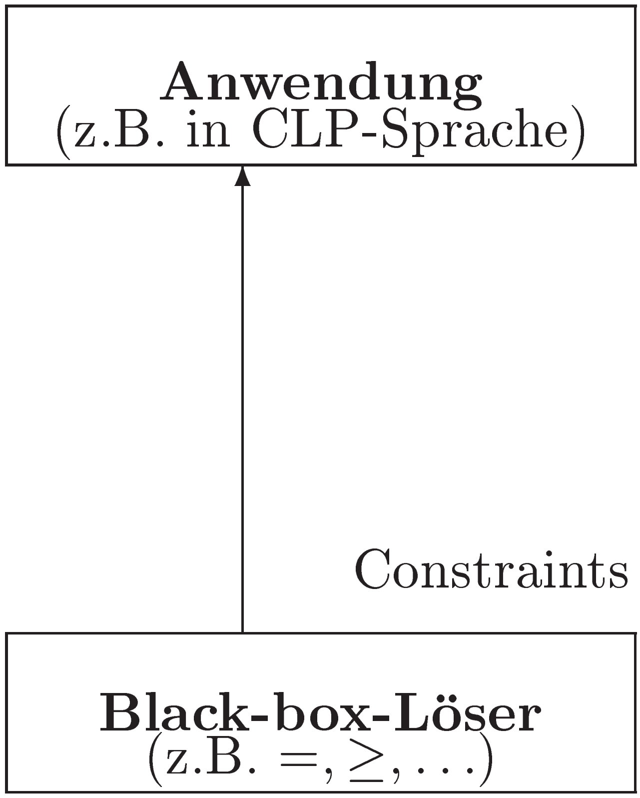
Architektur von Constraint-Programmen

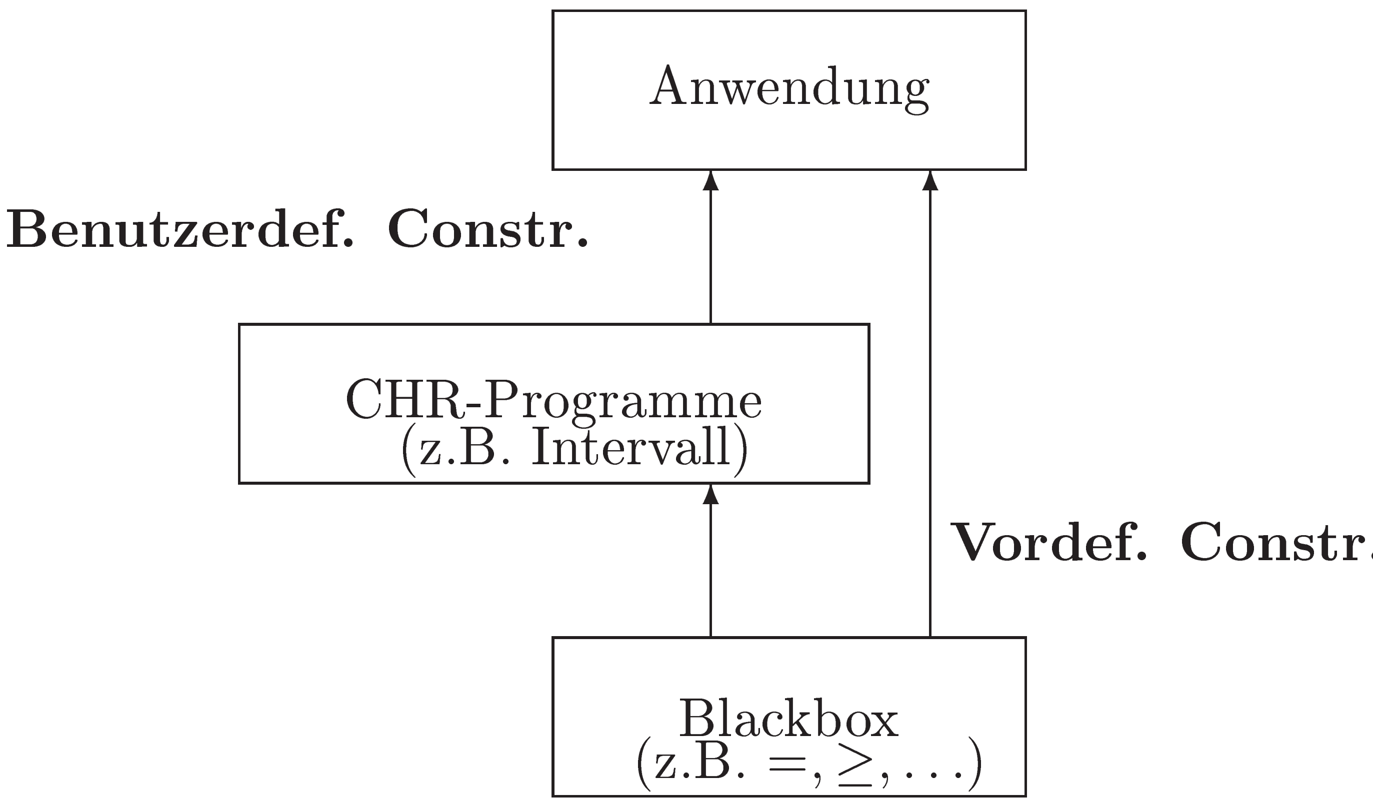


Stand 1999

Constraint Handling Rules

Höhere Sprache zur
Spezifikation/Implementierung von
Constraintlösern.

In CHR implementierten Constraintlöser sind offen, deklarativ, modifizierbar, komponierbar.



Constraint Handling Rules

Beispiel 1 Wir definieren mit CHR die Ordnungsrelation `X` $\leq$ `Y` als Constraint.

`reflexivitaet`	`@`	`X`$\leq$`X`	$\Leftrightarrow$	`true`.
`antisymmetrie`	`@`	`X`$\leq$`Y` $\land$ `Y`$\leq$`X`	$\Leftrightarrow$	`X`$=$`Y`.
`transitivitaet`	`@`	`X`$\leq$`Y` $\land$ `Y`$\leq$`Z`	$\Rightarrow$	`X`$\leq$`Z`.
Name		Kopf		Rumpf

`A`$\leq$`B` $\land$ `C`$\leq$`A` $\land$ `B`$\leq$`C`	(Transitivität)
`A`$\leq$`B` $\land$ `C`$\leq$`A` $\land$ `B`$\leq$`C` $\land$ `C`$\leq$`B`	(Antisymmetrie)
`A`$\leq$`B` $\land$ `C`$\leq$`A` $\land$ `B`$=$`C`	(Blackbox-Löser)
`A`$\leq$`B` $\land$ `B`$\leq$`A` $\land$ `B`$=$`C`	(Antisymmetrie)
`A`$=$`B` $\land$ `B`$=$`C`	

Anmerkung

CHR-Programme bestehen aus Regeln, die beschreiben, wie Constraints sich zu neuen Constraints vereinfachen und wie Constraints andere Constraints propagieren. Die Regeln zur Vereinfachung können als Erweiterung von CCL-Klauseln um Köpfe mit Konjunktionen (d. h. mehreren Atomen) aufgefaßt werden. Zur Anwendung kommt der „don't care“ Nichtdeterminismus. Fortgesetzte Vereinfachung und Propagierung von Constraints führt schließlich zur Lösung der Constraints, wie das folgende Beispiel illustriert.

Die Anwendung der Transitivitäts-Regel auf die Constraints $\mathtt{C} \leq \mathtt{A} \wedge \mathtt{A} \leq \mathtt{B}$ des Zieles propagiert das neue Constraint $\mathtt{C} \leq \mathtt{B}$. Dieses Constraint zusammen mit $\mathtt{B} \leq \mathtt{C}$ kann durch die Antisymmetrie-Regel zu $\mathtt{B} \doteq \mathtt{C}$ vereinfacht werden. Nachdem `B` und `C` nun gleich sind, kann die Antisymmetrie-Regel auch $\mathtt{A} \leq \mathtt{B} \wedge \mathtt{C} \leq \mathtt{A}$ zu $\mathtt{A} \doteq \mathtt{B}$ vereinfachen. Die Antwort ist also “`A`, `B` und `C` haben den gleichen Wert”.

CHR-Kalkül

Definition 1 Ein *CHR-Programm* ist eine endliche Menge von CHR-Regeln. Es gibt zwei Arten von CHR-Regeln:

Simplifikation $E \Leftrightarrow C \mid G$
Propagierung $E \Rightarrow C \mid G$

wobei der Kopf E eine Konjunktion von CHR-Constraints ist, der Wächter C eine Konjunktion von vordefinierten Constraints, und der Rumpf G ein beliebiges Ziel.

Bemerkung:

Es gibt eine dritte Art von CHR-Regeln:

Simpagation $E_1 \backslash E_2 \Rightarrow C \mid G$

Anmerkung

Im Gegensatz zur Signatur im CLP-Kalkül ist die Menge $\mathcal{P}$ der Prädikatensymbole in der Signatur im CHR-Kalkül in drei disjunkte Mengen unterteilt: $\mathcal{P}_p$ enthält die Prädikatensymbole der Basissprache, $\mathcal{P}_c$ enthält die in der Basissprache vordefinierten (engl. built-in) Constraintsymbole und $\mathcal{P}_e$ enthält die durch CHR definierten Constraintssymbole. Wir nennen durch CHR definierte Constraints abkürzend CHR-Constraints.

Die Definitionen für Atom, (atomare) Constraints, Zielatome und Ziele für CLP-Sprachen und CCLP-Sprachen gelten weiterhin, wobei darin die Constraints beliebig (d. h. mit Constraintsymbolen aus $\mathcal{P}_c \cup \mathcal{P}_e$) sein können.

CHR-Syntax

Atom:	A, B	$::=$	$p(t_1, \ldots, t_n)\ n \geq 0$
Vordef. Constraints:	C, D	$::=$	$c(t_1, \ldots, t_n) \mid C \wedge D\ n \geq 0$
CHR-Constraints:	E, F	$::=$	$\top \mid e(t_1, \ldots, t_n) \mid E \wedge F\ n \geq 0$
Ziele:	G, H	$::=$	$\top \mid \bot \mid A \mid C \mid E \mid G \wedge H$
Regeln:	R	$::=$	$E \Leftrightarrow C \mid G \quad \mid \quad E \Rightarrow C \mid G$
Programm:	P, Q	$::=$	$\{R_1, \ldots, R_m\}\ m \geq 0$

Kongruenz

Kommutativität:	$G_1 \wedge G_2$	$\equiv$	$G_2 \wedge G_1$
Assoziativität:	$G_1 \wedge (G_2 \wedge G_3)$	$\equiv$	$(G_1 \wedge G_2) \wedge G_3$
Identität:	$G \wedge \top$	$\equiv$	G
Absorption:	$G \wedge \bot$	$\equiv$	$\bot$

CHR-Kalkül

Definition 2 Ein *Zustand* ist ein Tripel $\langle G, E, C \rangle$, wobei G ein Ziel ist, E ein CHR-Constraint und C ein vordefiniertes Constraint. G wird Zielspeicher genannt, E CHR-Speicher und C (vordefinierter) Constraintspeicher. Der leere CHR-Speicher wird durch $\top$ dargestellt.

Anmerkung

Die Zustände werden um eine Komponente, die die CHR-Constraints aufnimmt, erweitert:

CHR-Kalkül

Definition 3 Ein *Anfangszustand* ist ein Zustand von der Form $\langle G, \top, true \rangle$. Ein Zustand heißt *erfolgreicher Endzustand* , wenn er von der Form $\langle \top, E, C \rangle$ ist, keine Reduktion mehr auf den Zustand anwendbar ist, und C ungleich *false* ist. Ein Zustand heißt *erfolgloser Endzustand* (gescheitert), wenn er die Form $\langle G, E, false \rangle$ hat. Die Definitionen für die Arten von Ableitungen und von Zielen aus den bisherigen Kalkülen gelten weiterhin.

Definition 4 Eine Ableitung, die mit $\langle G, \top, true \rangle$ beginnt, heißt *Ableitung für G.*

Anmerkung

Die Arten der Zustände sind analog zu den bisherigen definiert. Wir lassen dabei in erfolgreichen Endzuständen CHR-Constraints genauso wie vordefinierte Constraints zu:

CHR Reduktionen

Einfügen

$$\langle F \wedge G, E, C\rangle \mapsto_{Einfügen} \langle G, F \wedge E, C\rangle$$

Simplifizieren

$$\frac{\begin{array}{c}(F \Leftrightarrow D \mid H) \text{ aus } P\\ CT \models C \rightarrow \exists \bar{x}((F \doteq F') \wedge D)\end{array}}{\langle G, F' \wedge E, C\rangle \mapsto_{Simplifizieren} \langle G \wedge H, E, (F \doteq F') \wedge C \wedge D\rangle}$$

Propagieren

$$\frac{\begin{array}{c}(F \Rightarrow D \mid H) \text{ aus } P\\ CT \models C \rightarrow \exists \bar{x}((F \doteq F') \wedge D)\end{array}}{\langle G, F' \wedge E, C\rangle \mapsto_{Propagieren} \langle G \wedge H, F' \wedge E, (F \doteq F') \wedge C \wedge D\rangle}$$

Basissprache

$$\frac{\langle G, C\rangle \mapsto_{Name} \langle H, D\rangle \text{ ist eine Reduktion in der Basissprache}}{\langle G, E, C\rangle \mapsto_{Name} \langle H, E, D\rangle}$$

Anmerkung

Einfügen fügt ein im Ziel gefundenes CHR-Constraint in den CHR-Speicher ein. Dort kann es durch *Simplifizieren* und *Propagieren* vereinfacht und ergänzt werden. Die Reduktion *Simplifizieren* ist ähnlich der Reduktion *Entfalten* von CCLP-Sprachen. Der wesentliche Unterschied liegt darin, daß nicht nur ein Atom, sondern eine beliebige Konjunktion F' von CHR-Constraints durch eine Regel entfaltet werden kann. Die Reduktion *Propagieren* unterscheidet sich von der Reduktion *Simplifizieren* dadurch, daß die CHR-Constraints F' im CHR-Speicher verbleiben.

Im kombinierten Zustandsübergangssystem werden die Reduktionen der Basissprache, *Basissprache*, so auf Zustände mit CHR-Speicher übertragen, daß sie den CHR-Speicher mitführen, sonst aber ignorieren.

CHR-Kalkül

Beispiel 2

$$\{\texttt{X} \leq \texttt{Y} \Leftrightarrow \texttt{X=Y} \mid \mathit{true}, \quad \texttt{\% Reflexivitaet}$$
$$\texttt{X} \leq \texttt{Y} \wedge \texttt{Y} \leq \texttt{X} \Leftrightarrow \texttt{X=Y}, \quad \texttt{\% Antisymmetrie}$$
$$\texttt{X} \leq \texttt{Y} \wedge \texttt{Y} \leq \texttt{Z} \Rightarrow \texttt{X} \leq \texttt{Z} \quad \texttt{\% Transitivitaet}$$
$$\}$$

$$
\begin{array}{ll}
 & \langle \texttt{A} \leq \texttt{B} \wedge \texttt{C} \leq \texttt{A} \wedge \texttt{B} \leq \texttt{C}, \top, \mathit{true} \rangle \\
\mapsto^{3}_{Einf\ddot{u}gen} & \langle \top, \texttt{A} \leq \texttt{B} \wedge \texttt{C} \leq \texttt{A} \wedge \texttt{B} \leq \texttt{C}, \mathit{true} \rangle \\
\mapsto_{Propagieren} & \langle \texttt{C} \leq \texttt{B}, \texttt{A} \leq \texttt{B} \wedge \texttt{C} \leq \texttt{A} \wedge \texttt{B} \leq \texttt{C}, \mathit{true} \rangle \\
\mapsto_{Einf\ddot{u}gen} & \langle \top, \texttt{A} \leq \texttt{B} \wedge \texttt{C} \leq \texttt{A} \wedge \texttt{B} \leq \texttt{C} \wedge \texttt{C} \leq \texttt{B}, \mathit{true} \rangle \\
\mapsto_{Simplifizieren} & \langle \texttt{B} = \texttt{C}, \texttt{A} \leq \texttt{B} \wedge \texttt{C} \leq \texttt{A}, \mathit{true} \rangle \\
\mapsto_{Vereinfachen} & \langle \top, \texttt{A} \leq \texttt{B} \wedge \texttt{C} \leq \texttt{A}, \texttt{B} = \texttt{C} \rangle \\
\mapsto_{Simplifizieren} & \langle \texttt{A} = \texttt{B}, \top, \texttt{B} = \texttt{C} \rangle \\
\mapsto_{Vereinfachen} & \langle \top, \top, \texttt{A} = \texttt{B} \wedge \texttt{B} = \texttt{C} \rangle
\end{array}
$$

CHR-Kalkül

Beispiel 3 Wir definieren `min` als CHR-Constraint, wobei $\doteq$, als auch $\leq$ und $<$ vordefinierte Constraints sind:

$$
\begin{array}{l}
\{\mathtt{min(X,Y,Z)} \Leftrightarrow \mathtt{X} \leq \mathtt{Y} \mid \mathtt{X} \doteq \mathtt{Z}, \\
\;\mathtt{min(X,Y,Z)} \Leftrightarrow \mathtt{Y} \leq \mathtt{X} \mid \mathtt{Y} \doteq \mathtt{Z}, \\
\\
\;\mathtt{min(X,Y,Z)} \Leftrightarrow \mathtt{Z} < \mathtt{X} \mid \mathtt{Y} \doteq \mathtt{Z}, \\
\;\mathtt{min(X,Y,Z)} \Leftrightarrow \mathtt{Z} < \mathtt{Y} \mid \mathtt{X} \doteq \mathtt{Z}, \\
\\
\;\mathtt{min(X,Y,Z)} \Rightarrow \mathtt{Z} \leq \mathtt{X} \wedge \mathtt{Z} \leq \mathtt{Y} \\
\}
\end{array}
$$

Ziele:

`min(1,2,C)`

`min(A,A,B)`

$\mathtt{min(A,B,C)} \wedge \mathtt{A} \leq \mathtt{B}$

`min(A,2,2)`

`min(A,2,3)`

Anmerkung

Im Unterschied zur Formulierung des Prädikates `min` im CCLP-Kalkül sind wir nun an einer möglichst vollständigen Abdeckung aller Vereinfachungsmöglichkeiten für das Constraint `min` interessiert. Die ersten beiden Simplifikationsregeln entsprechen der Formulierung im CCLP-Kalkül. Die zusätzlichen zwei Simplifikationsregeln können `min` vereinfachen, wenn das Größenverhältnis zwischen der Variable `Z` und einer der Variablen `X` bzw. `Y` bekannt ist. Die letzte Regel sagt: Das Minimum zweier Zahlen muß kleiner oder gleich den beiden Zahlen sein. Auf weitere, mehrköpfige Regeln haben wir hier verzichtet.

Das Ziel `min(1,2,C)` reduziert wie in CCLP gewohnt zu $\mathtt{C} \doteq \mathtt{1}$. Dem Ziel `min(A,2,2)` fügt die Propagierungsregel das Constraint $\mathtt{2} \leq \mathtt{A} \land \mathtt{2} \leq \mathtt{2}$ hinzu. $\mathtt{2} \leq \mathtt{A}$ macht die zweite Simplifikationsregel auf `min(A,2,2)` anwendbar und die Antwort ist $\mathtt{2} \leq \mathtt{A}$. Das Scheitern von `min(A,2,3)` ist ebenso möglich.

Die drei Ziele können also mit geeigneten CHR-Regeln ohne die Notwendigkeit zur Suche (wie im CLP-Kalkül) oder zu einer Constraintserweiterung vollständig behandelt werden.

Vergleich: CHR vs. Prolog

CHR	Prolog
Spracherweiterung	Allg. Sprache
Relational	Relational
Simplifikationsregeln	Klauseln
Propagierungsregeln	-
Mehrköpfige Regeln	-
committed-choice	backtracking
Matching	Unifikation
Wächter	-

Committed-choice bedingt:

- Matching anstatt Unifikation (sonst unvollständig).
- Wächter zur Regelauswahl (da kein backtracking).

Monotonieeigenschaft für CHR-Sprachen

Die Monotonieeigenschaft gilt für die Reduktionen *Einfügen*, *Simplifizieren* und *Propagieren* von CHR:
Wenn $\langle G, E, C\rangle \mapsto \langle G', E', C'\rangle$ und
$CT \models \exists(C \wedge D)$, dann
$\langle G \wedge H, E \wedge F, C \wedge D\rangle \mapsto$
$\langle G' \wedge H, E' \wedge F, C' \wedge D\rangle$.

Anmerkung

Wenn eine Reduktion für CHR auf einen Zustand angewendet werden kann, dann kann diese Reduktion auch auf jeden erweiterten Zustand mit zusätzlichem Ziel und zusätzlichen Constraints angewendet werden, solange der Speicher für vordefinierte Constraints konsistent bleibt, und der Folgezustand unterscheidet sich nur durch das zusätzliche Ziel und die zusätzlichen Constraints.

Deklarative Semantik

Die logische Leseweise der beiden Arten von CHR-Regeln ist:

Simplifikation	$\forall \bar{x}\ (C \rightarrow (E \leftrightarrow \exists \bar{y}\ G))$
Propagierung	$\forall \bar{x}\ (C \rightarrow (E \rightarrow \exists \bar{y}\ G))$

wobei $\bar{x}$ die in E und C vorkommenden Variablen sind und $\bar{y}$ die Variablen, die nur in G vorkommen.

Die logische Leseweise $\mathcal{P}$ eines CHR-Programms P ist die Konjunktion der logischen Leseweise der im Programm enthaltenen CHR-Regeln.

Anmerkung

Die deklarative Semantik von CHR-Programmen beruht nicht auf der Clark'schen Vervollständigung wie in den bisher behandelten Sprachfamilien, sondern jeder CHR-Regel wird für sich genommen eine logische Formel zugeordnet. Um die Darstellung der deklarativen Semantik zu erleichtern und unabhängig von der Basissprache zu machen, gehen wir in diesem Abschnitt aus, daß CHR-Programme bzw. Ziele nur Constraints und keine Atome beinhalten. Wir können dies o.E.d.A. tun, weil die in CHR-Programmen verwendeten Prädikate der Basissprache zur Laufzeit deterministisch sein müssen und daher als vordefinierte Constraints aufgefaßt werden können.

Der Wächter als Vorbedingung einer Regel impliziert dabei einen logischen Zusammenhang zwischen Kopf und Rumpf der Regel: Im Fall einer Simplifikationsregel ist der Zusammenhang eine logischen Äquivalenz zwischen Kopf und Rumpf, im Fall einer Propagierungsregel eine Implikation.

Logische Bedeutung eines Zustandes

Definition 5 Jedem Zustand $\langle H, E, C\rangle$, der in einer Ableitung für G vorkommt, wird eine Formel $\exists\bar{x}\ H \wedge E \wedge C$ als *logische Beschreibung* zugeordnet, wobei $\bar{x}$ die Variablen sind, die in $\langle H, E, C\rangle$ aber nicht in G vorkommen. Die in G vorkommenden Variablen bleiben frei.

Ein *herleitbares Constraint eines Zieles G* ist ist die logische Beschreibung eines Zustandes, der in einer Ableitung beginnend mit $\langle G, \top, true\rangle$ vorkommt.

Ein *Antwortconstraint (kurz: eine Antwort) eines Zieles G* ist die logische Beschreibung eines Endzustandes einer Ableitung für G.

Zusammenhang zwischen Deklarativer und Operationaler Semantik

Korrektheit

Lemma 1 Sei P ein CHR-Programm und G ein Ziel. Wenn H ein herleitbares Constraint von G ist, dann gilt

$$\mathcal{P}, CT \models \forall\ (G \leftrightarrow H).$$

Theorem 1 (Korrektheit) Sei P ein CHR-Programm und G ein Ziel. Wenn G eine Ableitung mit einem Antwortconstraint H hat, dann gilt

$$\mathcal{P}, CT \models \forall\ (G \leftrightarrow H).$$

Anmerkung

Vollständigkeit und Korrektheit des CHR-Kalküls sind unabhängig von der Art der Ableitung, eine Unterscheidung in erfolgreiche und erfolglose Ableitungen wie in den bisherigen Kalkülen ist nicht notwendig. Statt mit Implikationen zwischen Ziel und Antwortconstraint haben wir es mit logischen Äquivalenzen zu tun. Die Resultate basieren vor allem darauf, daß Reduktionen für CHR die Äquivalenz von den Zuständen erhalten:

Zusammenhang zwischen Deklarativer und Operationaler Semantik

Vollständigkeit

Theorem 2 (Vollständigkeit) Sei P ein CHR-Programm und G ein Ziel, das mindestens eine endliche Ableitung hat. Wenn $\mathcal{P}, CT \models \forall\, (G \leftrightarrow H)$ gilt, dann hat G eine Ableitung mit Antwortconstraint H', wobei

$$\mathcal{P}, CT \models \forall\, (H \leftrightarrow H').$$

Anmerkung

Im CHR-Kalkül betrachten wir wie im CCLP-Kalkül bei der Vollständigkeit eine einzige Ableitung und ihre Antwort:

Vollständigkeit: Beispiel (I)

Das Vollständigkeitstheorem gilt nicht, wenn ein Ziel keine endlichen Ableitungen hat:

Beispiel 4 Sei P das CHR-Programm:

$\{\texttt{p} \;\Leftrightarrow\; \texttt{p}\}$

Gegeben sei G das Ziel $\texttt{p}$. Es gilt $\mathcal{P}, CT \models \texttt{p} \leftrightarrow \texttt{p}$. Aber G hat nur eine unendliche Ableitung.

Vollständigkeit: Beispiel (II)

Die Vollständigkeitsaussage ist für gescheiterte Ableitungen unbefriedigend:

Grund: Antwortconstraint nicht explizit inkonsistent:

Wenn $\mathcal{P}, CT \models \neg\exists G$ gilt, dann hat G eine Ableitung mit Antwortconstraint H', wobei

$$\mathcal{P}, CT \models \forall\ (\mathit{false} \leftrightarrow H').$$

Trivialer Fall: Das Antwortconstraint H' ist identisch zu G

Beispiel 5 Sei P das folgende CHR-Programm

$\{$`p` $\Leftrightarrow$ `q`,
 `p` $\Leftrightarrow$ *false* $\}$

Es gilt $\mathcal{P}, CT \models \neg$`q`. Aber `q` hat keine gescheiterte Ableitung, sondern eine erfolgreiche mit Antwortconstraint `q`.

Verbesserte Vollständigkeitsergebnisse

Konfluente Programme garantieren, daß zu einem Ziel immer das (syntaktisch) gleiche Ergebnis hergeleitet wird, egal welche Regeln verwendet wird.

Konfluenz ist eine Verallgemeinerung und Erweiterung des Determinismusbegriffs aus CCLP-Sprachen

Theorem 3 Korrektheit und Vollständigkeit von erfolglosen Ableitungen Seien P ein *konfluentes* CHR-Programm und G ein Ziel, das mindestens eine Antwort hat, die nur vordefinierte Constraints enthält.
$\mathcal{P}, CT \models \neg\exists G$ gilt genau dann, wenn jede endliche Ableitung für G scheitert.

Konfluenz

Beispiel 6 Sei P das CHR–Programm für das Prädikat `mwerfen` aus dem CCLP-Kapitel:

$$\{\texttt{mwerfen(Seite)} \Leftrightarrow \mathit{true} \mid \texttt{Seite} \doteq \texttt{kopf},$$
$$\texttt{mwerfen(Seite)} \Leftrightarrow \mathit{true} \mid \texttt{Seite} \doteq \texttt{zahl}\}$$

Vom Ausgangszustand

$$\langle \top, \texttt{mwerfen}(\texttt{Seite}), \mathit{true} \rangle$$

erreicht man je nach angewandter Regel die Zustände (das kritische Paar)

$\langle \top, \top, \texttt{Seite}\doteq\texttt{kopf} \rangle$ und $\langle \top, \top, \texttt{Seite}\doteq\texttt{zahl} \rangle$

Diese beiden Zustände sind nicht zusammenführbar: Das Programm ist also nicht konfluent.

Anmerkung

Der Ausgangszustand wird durch das *Überlappen* zweier Regeln gebildet: Er hat im CHR-Constraintsspeicher die Köpfe der beiden Regeln, wobei ein oder mehrere Kopfconstraints miteinander gleichgesetzt wurden, und im vordefinierten Constraintspeicher die Wächter der beiden Regeln.

Konfluenz

Beispiel 7 Sei P das CHR–Programm:

$\{\mathtt{min(X,Y,Z)}\ \Leftrightarrow\ \mathtt{X}\ \leq\ \mathtt{Y}\ |\ \mathtt{X}\ \doteq\ \mathtt{Z},$
$\ \mathtt{min(X,Y,Z)}\ \Leftrightarrow\ \mathtt{Y}\ \leq\ \mathtt{X}\ |\ \mathtt{Y}\ \doteq\ \mathtt{Z}\}$

Die beiden Regeln haben den Ausgangszustand

$$\langle \top, \mathtt{min}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}), X \leq Y \wedge Y \leq X \rangle$$

und das kritische Paar

$\langle X \doteq Z, \top, X \leq Y \wedge Y \leq X \rangle$ und
$\langle Y \doteq Z, \top, X \leq Y \wedge Y \leq X \rangle$

Beide Zustände des kritischen Paares reduzieren zu einem syntaktisch gleichen Zustand

$$\langle \top, \top, X \doteq Y \wedge Y \doteq Z \rangle$$

Das Programm ist also konfluent.