	Nebenläufige Constraintlogikprogrammierung
1981	Clark und Gregory, Parlog, Relational Language for Parallel Programming
1983	Shapiro, Concurrent Prolog, dann FCP (Flat Conurrent Prolog)
1985	Ueda, GHC (Guarded Horn Clauses)
1986	Clark, Parlog (Parallel Prolog)
1987	Maher, ALPS-Sprachfamilie
1989	Saraswat, CC-Sprachfamilie (Concurrent Constraints)
1990	Fifth Generation Computing Project, KL1 (Kernel Language One)
1990	Haridi, AKL (Andorra Kernel Language)
1991	Hermengildo, CIAO
1992	Smolka, OZ (integriert Funktionen, Objekte und Constraints)

Anmerkung

Nebenläufige Programmiersprachen erlauben die Beschreibung von Programmen, die nebeneinander ausgeführt werden und miteinander interagieren können, sogenannten Prozessen. Prozesse können entweder lokale Aktionen ausführen oder miteinander kommunizieren und sich synchronisieren, indem sie Nachrichten senden und empfangen. Die miteinander kommunizierenden Prozesse bilden ein Prozess-Netzwerk, das sich dynamisch ändern kann. Für die Nebenläufigkeit spielt es keine Rolle, ob Prozesse physikalisch parallel oder im Wechsel durchgeführt werden.

In nebenläufigen Constraintlogikprogrammiersprachen sind die Prozesse durch Prädikate implementiert, die miteinander durch Abfragen und Einfügen von Constraints in einen Constraintspeicher interagieren. Das Abfragen ist mittels Folgerungstest realisiert.

Prozesse sind typisch in verteilten Systemen. Prozesse können auch bewußt nichtterminierend sein. Man denke etwa an ein Betriebssystem, das im Normalfall ständig läuft; oder an Überwachungsprogramme, die kontinuierlich ankommende Meßdaten verarbeiten und regelmäßig ein Zwischenergebnis liefern sollen. Es muß also möglich sein, bereits während des Rechenvorganges, während der Prozeß läuft, eindeutige Zwischenergebnisse an andere Prozesse zu übermitteln.

Nebenläufige CLP

Definition 1 Ein *NCL-Programm* ist eine endliche Menge von NCL-Klauseln. Eine NCL-Klausel (auch: bedingte Klausel) ist von der Form

$$A \leftarrow C \mid G$$

wobei der Kopf A ein Atom ist, C eine Konjunktion von Constraints, genannt *Wächter* (auch: Vorbedingung, engl. guard), und der Rumpf G ein Ziel.

Anmerkung

Wir werden uns im folgenden auf NCLP-Sprachen beschränken, die nur „don't care“ Nichtdeterminismus erlauben. Die Syntax und Semantik von NCLP-Sprachen orientiert sich wie bei den bisher behandelten Sprachen an der Logik, während viele andere nebenläufige Programmiersprachen auf Prozessalgebra basieren. Die Syntax von NCLP-Sprachen unterscheidet sich von der Syntax der CLP-Sprachen dadurch, daß Hornklauseln syntaktisch um Vorbedingungen erweitert werden.

Man schreibt für $H \leftarrow \mathit{true} \mid B$ auch abkürzend $H \leftarrow B$. In manchen NCLP-Sprachen werden auch Wächter mit beliebigen Zielen zugelassen, man nennt diese englischsprachig "deep guards" im Unterschied zu den "flat guards", wo Wächter auf Constraints beschränkt sind.

NCLP Syntax

Atome:	A, B	$::=$	$p(t_1, \ldots, t_n) \quad n \geq 0$
Constraints:	C, D	$::=$	$c(t_1, \ldots, t_n) \mid C \wedge D \; n \geq 0$
Ziele:	G, H	$::=$	$\top \mid A \mid C \mid G \wedge H$
NCL-Klauseln:	K	$::=$	$A \leftarrow C \vert G$
Programme:	P, Q	$::=$	$\{K_1, \ldots, K_m\} \; m \geq 0$

NCLP 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:	$G \wedge \bot$	$\equiv$	$\bot$

NCLP Reduktionsregeln

Entfalten

$$\frac{\begin{array}{c}(B \leftarrow D \mid H) \text{ aus } P \\ CT \models C \rightarrow \exists \bar{x}((B \doteq A) \wedge D)\end{array}}{\langle A \wedge G, C \rangle \mapsto_{Entfalten} \langle H \wedge G, (B \doteq A) \wedge D \wedge C \rangle}$$

Vereinfachen:

$$\frac{CT \models (C \wedge D_1) \leftrightarrow D_2}{\langle C \wedge G, D_1 \rangle \mapsto_{Vereinfachen} \langle G, D_2 \rangle}$$

Anmerkung

Der Unterschied liegt in den Reduktionen: In der Reduktionsbedingung für *Entfalten* wird nun der Wächter berücksichtigt und die Reduktion kann nicht mehr rückgängig gemacht werden. Die Reduktion *Scheitern* gibt es nicht mehr.

Ein Atom A eines Zustandes kann mit einer Klausel $B \leftarrow D \mid G$ genau dann mit *Entfalten* reduziert werden, wenn aus dem aktuellen Constraintsspeicher die Formel $\exists \bar{x}\ (B \doteq A \wedge D)$ logisch folgt, wobei $\bar{x}$ die in B vorkommenden Variablen sind. Wird die Reduktion *Entfalten* ausgeführt, spricht man davon, daß die Klausel *feuert.*

Der obige Folgerungstest verallgemeinert den Begriff des Tests aus herkömmlichen Programmiersprachen. Operational entspricht der Folgerungstest für die Gleichheit $\exists \bar{x}\ (B \doteq A)$ einer Überprüfung, ob das ausgewählte Ziel eine Instanz des Klauselkopfes ist, mit einer nachfolgenden Unifikation (engl. one-way unification, matching).

Unsere vom CLP-Kalkül ausgehende operationale Semantik ist für die genaue Beschreibung nebenläufiger Berechnungen nicht ganz ausreichend, denn sie beschreibt nicht, wie mehrere Atome gleichzeitig entfaltet werden können.

Zustände, Ableitungen und Ziele

Die Definitionen der Arten von Zuständen, Ableitungen und Zielen sind diesselben wie im CLP-Kalkül mit einem einzigen Unterschied:

Definition 2 Ein Zustand $\langle G, C \rangle$ mit G ungleich $\top$ und C ungleich *false*, auf den keine Reduktion mehr anwendbar ist, heißt *verklemmt* (engl. deadlocked).

Anmerkung

Es gibt die Reduktion *Scheitern* nicht mehr, d. h. Atome, für die keine passende Klausel gefunden wurde, werden verzögert in der Hoffnung, daß *Entfalten* später möglich ist. Daraus ergibt sich auch eine neue Art von Endzustand, die daher kommt, daß kein Atom mehr reduziert werden kann.

Verklemmte Zustände will man nach Möglichkeit vermeiden. Sie werden meist als Programmfehler angesehen, denn Prozesse sollen ja entweder ständig weiterrechnen oder ordnungsgemäß terminieren.

Beispiel min in NCLP

min als NCL-Prädikat:

$$\{\texttt{min(X,Y,Z)} \leftarrow \texttt{X} \leq \texttt{Y} \mid \texttt{X} \doteq \texttt{Z},$$
$$\texttt{min(X,Y,Z)} \leftarrow \texttt{Y} \leq \texttt{X} \mid \texttt{Y} \doteq \texttt{Z}\ \}$$

$$\begin{array}{ll} & \langle \texttt{min}(1,2,\texttt{C}), true\rangle \\ \mapsto_{Entfalten} & <\texttt{X}\doteq\texttt{Z},\ 1\doteq\texttt{X}\wedge 2\doteq\texttt{Y}\wedge\texttt{C}\doteq\texttt{Z}\wedge\texttt{X}\leq\texttt{Y}> \\ \mapsto_{Vereinfachen} & <\top,\ \texttt{C}\doteq 1> \end{array}$$

Erste Klausel erfüllt Reduktionsbedingung:
$CT \models (true \rightarrow$
$\exists \texttt{X},\texttt{Y},\texttt{Z}(\texttt{min}(1,2,\texttt{C})\doteq\texttt{min}(\texttt{X},\texttt{Y},\texttt{Z}) \wedge \texttt{X} \leq \texttt{Y}))$

Zweite Klausel erfüllt nicht die Reduktionsbedingung:
$CT \not\models (true \rightarrow$
$\exists \texttt{X},\texttt{Y},\texttt{Z}(\texttt{min}(1,2,\texttt{C})\doteq\texttt{min}(\texttt{X},\texttt{Y},\texttt{Z}) \wedge \texttt{Y} \leq \texttt{X}))$

Anmerkung

Dies ist die einzig mögliche Ableitung für $\langle \mathtt{min}(\mathtt{1}, \mathtt{2}, \mathtt{C}), \mathit{true}\rangle$, während in der CLP-Formulierung ein Scheitern bei Auswahl der zweiten Klausel die Folge war.

Der Anfangszustand $\langle \mathtt{min}(\mathtt{A}, \mathtt{2}, \mathtt{1}), \mathit{true}\rangle$ erlaubt keine Reduktionen, da nicht über die Beziehung von `A` und `2` entschieden werden kann. Der Zustand ist daher verklemmt. Ebenso mit `min(A,2,2)` und `min(A,2,3)`. Im CLP-Kalkül war es hier möglich, durch Suche „rückwärts“ zu rechnen.

Wenn der Anfangszustand $\langle \mathtt{min}(\mathtt{A}, \mathtt{A}, \mathtt{B}), \mathit{true}\rangle$ ist, erfüllen beide Klauseln die Reduktionsbedingung von *Entfalten.* Nur eine Klausel wird gewählt. Egal welche Klausel das ist, das Antwortconstraint ist $\mathtt{A} \doteq \mathtt{B}$.

Der Zustand $\langle \mathtt{min}(\mathtt{A}, \mathtt{B}, \mathtt{C}), \mathtt{A} \leq \mathtt{B}\rangle$ reduziert mit der ersten Klausel zur Antwort $\mathtt{A} \leq \mathtt{B} \wedge \mathtt{A} \doteq \mathtt{C}$. Analog reduziert der Zustand $\langle \mathtt{min}(\mathtt{A}, \mathtt{B}, \mathtt{C}), \mathtt{A} \geq \mathtt{B}\rangle$ mit der zweiten Klausel zur Antwort $\mathtt{A} \geq \mathtt{B} \wedge \mathtt{B} \doteq \mathtt{C}$.

Beispiel `hamming` in NCLP (I)

Das Hamming'sche Programmierproblem:
Berechne eine geordnete Sequenz aller Zahlen, deren Primfaktoren 2, 3 oder 5 sind.

Die Sequenz beginnt mit den Zahlen
$2 \diamond 3 \diamond 4 \diamond 5 \diamond 6 \diamond 8 \diamond 9 \diamond 10 \diamond 12 \diamond 15 \diamond 16 \diamond 18 \diamond 20 \diamond 24 \diamond 25 \diamond \ldots.$

Zwei benachbarte Zahlen später in der Sequenz sind 79164837199872 und 79254226206720.

Beispiel `hamming` in NCLP (I)

Idee:

- Wir gehen von einem nichtterminierenden Prädikat `hamming(S)` aus, das alle Zahlen geordnet berechnet.
- Wir erzeugen drei Prozesse `mults`, die die Zahlen der Ergebnissequenz `S` mit 2, 3 bzw. 5 multiplizieren, sobald sie bekannt werden.
- Zwei `merge`-Prozesse sorgen dafür, daß die drei entstehenden Sequenzen geordnet ohne Duplikate in eine Sequenz verzahnt werden.
- Diese Sequenz ist wieder die Ergebnissequenz `S`, sodaß dadurch die `mults`-Prozesse wiederum mit weiteren Zahlen versorgt werden.

hamming in NCLP (III)

NCL-Programm

```
hamming(S) ←
     mults(1◇S,2,S2) ∧
     mults(1◇S,3,S3) ∧
     mults(1◇S,5,S5) ∧
     merge(S2,S3,S23) ∧ merge(S5,S23,S),

 mults(X◇S,N,XSN) ←
     XSN ≐ X*N◇SN ∧ mults(S,N,SN),

 merge(X◇In1,Y◇In2,XYOut) ← X ≐ Y |
     XYOut ≐ X◇Out ∧ merge(In1,In2,Out),
 merge(X◇In1,Y◇In2,XYOut) ← X < Y |
     XYOut ≐ X◇Out ∧ merge(In1,Y◇In2,Out),
 merge(X◇In1,Y◇In2,XYOut) ← X > Y |
     XYOut ≐ Y◇Out ∧ merge(X◇In1,In2,Out)
```

Anmerkung

Das Ziel `mults(1◇S,2,S2)` reduziert mit *Enfalten* und *Vereinfachen* zu

$\texttt{S2} \doteq \texttt{2}\diamond\texttt{S2N} \wedge \texttt{mults(S,2,S2N)}$. Während das Ziel im CLP-Kalkül zu unendlichen Ableitungen führt, verzögert es im NCLP-Kalkül solange, bis das erste Argument von `mults` ausreichend bekannt ist.

Ausgehend vom Ziel `hamming(S)` reduzieren die drei entstehenden `mults`-Prozesse daher zu
$\texttt{S2} \doteq \texttt{2}\diamond\texttt{S2N} \wedge \texttt{S3} \doteq \texttt{3}\diamond\texttt{S3N} \wedge \texttt{S5} \doteq \texttt{5}\diamond\texttt{S5N}$
$\wedge\ \texttt{mults(S,2,S2N)} \wedge \texttt{mults(S,3,S3N)} \wedge$
$\texttt{mults(S,5,S5N)} \wedge$
$\texttt{merge(S2,S3,S23)} \wedge \texttt{merge(S5,S23,S)}$.

Da die ersten Zahlen der Sequenzen `S2` und `S3` bekannt sind, kann das Ziel `merge(S2,S3,S23)` mit der zweiten Klausel für `merge` entfaltet werden und so die erste Zahl der Sequenz `S23` berechnen:
$\texttt{S23} \doteq \texttt{2}\diamond\texttt{S23N} \wedge \texttt{merge(S2N,S3,S23N)}$
Nun kann `merge(S5,S23,S)` reduzieren zu:
$\texttt{S} \doteq \texttt{2}\diamond\texttt{SN} \wedge \texttt{merge(S5,S23N,SN)}$.

Um bedingtes Einfügen erweiterte NCLP Syntax

NCL-Klauseln: K $::= A \leftarrow C : D \mid G$

Um bedingtes Einfügen erweiterte NCLP Reduktionen

Entfalten:

$$\frac{\begin{array}{c}(B \leftarrow D_1 : D_2 \mid H) \text{ aus } P\\ CT \models C \rightarrow \exists \bar{x}((B \doteq A) \wedge D_1)\\ CT \models \exists((B \doteq A) \wedge D_1 \wedge D_2 \wedge C)\end{array}}{\langle A \wedge G, C\rangle \mapsto_{Entfalten} \langle G \wedge H, (B \doteq A) \wedge D_1 \wedge D_2 \wedge C\rangle}$$

Vereinfachen:

$$\frac{CT \models (C \wedge D_1) \leftrightarrow D_2}{\langle C \wedge G, D_1\rangle \mapsto_{Vereinfachen} \langle G, D_2\rangle}$$

Anmerkung

Der NCLP-Kalkül kann um das *bedingte Einfügen* von Constraints (engl. atomic tell) erweitert werden. Die Constraints stehen syntaktisch in einem zusätzlichen Teil des Wächters.

Das Einfügen von Constraints aus dem Rumpf der Klausel wird im Gegensatz dazu als "eventual tell" bezeichnet.

Die Reduktion *Entfalten* kann nur stattfinden, wenn zusätzlich das Einfügen von D den Constraintspeicher konsistent erhält.

Mithilfe dieses erweiterten Kalküls ist es möglich, den Constraintspeicher immer konsistent zu halten und so das gefürchtete globale Scheitern zu vermeiden, indem man Constraints nur im Wächter erlaubt, aber nicht mehr im Rumpf.

Beispiel Bedingtes Einfügen

$$\{\texttt{min(X,Y,Z)} \leftarrow \texttt{X} \leq \texttt{Y} : \texttt{X} \doteq \texttt{Z} \mid \textit{true},$$
$$\texttt{min(X,Y,Z)} \leftarrow \texttt{Y} \leq \texttt{X} : \texttt{Y} \doteq \texttt{Z} \mid \textit{true}\}$$

Ziel `min(1,2,3)` verklemmt.

Anmerkung

Die Zustände reduzieren zu den gleichen Antwortconstraints. Der Unterschied äußert sich in einem Ziel wie `min(1,2,3)`. Während die Ableitungen für dieses Ziel im ursprünglichen NCLP-Kalkül scheitern, wird es hier im erweiterten Kalkül verzögert und kann daher nicht die Berechnungen in den anderen Prozessen stören. Dieses Verhalten läßt sich nicht ohne bedingtes Einfügen von Constraints erreichen.

Stabilitätseigenschaft für NCLP-Sprachen

Wenn $\langle G, C\rangle \mapsto_{Entfalten} \langle G', C'\rangle$ und D ist ein Constraint und $C' \wedge D$ ist erfüllbar, dann ist $\langle G \wedge H, C \wedge D\rangle \mapsto_{Entfalten} \langle G' \wedge H, C' \wedge D\rangle$ möglich.

Für NCLP ohne bedingtes Einfügen gilt sogar die **Monotonieeigenschaft**:
Wenn $\langle G, C\rangle \mapsto_{Entfalten} \langle G', C'\rangle$ und D ist ein Constraint und $C \wedge D$ ist erfüllbar, dann ist $\langle G \wedge H, C \wedge D\rangle \mapsto_{Entfalten} \langle G' \wedge H, C' \wedge D\rangle$ möglich.

Anmerkung

Wenn eine Reduktion *Entfalten* zu einem Folgezustand mit Constraintspeicher C' führt, dann kann diese Reduktion auch auf jeden Zustand mit zusätzlichem Ziel und speziellerem Constraintspeicher als C' (solange er konsistent bleibt) angewendet werden, und der Folgezustand unterscheidet sich nur durch das zusätzliche Ziel und das zusätzliche Constraint.

Deklarative Semantik NCLP

Clark'sche Vervollständigung wie in CLP.

Die logische Leseweise einer NCL-Klausel
$A \leftarrow C : D \mid G$ ist

$$A \leftarrow C \wedge D \wedge G.$$

Anmerkung

Die Symbole „|“ und “:” werden dabei als Konjunktionen interpretiert.

Die Diskrepanz zwischen operationaler und deklarativer Semantik ist in NCLP-Sprachen aber wegen des verwendeten „don’t care“ Nichtdeterminismus größer. Diese Form von Nichtdeterminismus bedeutet ja, daß ein Ziel immer nur eine Ableitung hat. Man wünscht sich daher Vollständigkeitssätze für NCLP-Sprachen, die sich im Gegensatz zu denen von CLP-Sprachen auf eine einzige Ableitung beziehen. D. h. die Disjunkte aus dem entsprechenden CLP-Theorem werden zu einem einzigen Constraint eingeschränkt.

Korrektheit

Theorem 1 Korrektheit von erfolgreichen Ableitungen Sei P ein NCL-Programm und G ein Ziel. Wenn G eine erfolgreiche Ableitung mit einem Antwortconstraint C hat, dann gilt

$$P^{\leftrightarrow}, CT \models \forall(C \rightarrow G)$$

Problem

Beispiel 1 Werfen einer Münze:

$$\{\texttt{mwerfen(Seite)} \leftarrow \ \texttt{Seite} \doteq \texttt{kopf,}$$
$$\texttt{mwerfen(Seite)} \leftarrow \ \texttt{Seite} \doteq \texttt{zahl}\}$$

Die deklarative Semantik $P^{\leftrightarrow}$ ist:

$$\{\texttt{mwerfen(Seite)} \Leftrightarrow$$
$$\texttt{Seite} \doteq \texttt{kopf} \vee \texttt{Seite} \doteq \texttt{zahl}\}$$

Nach dem gewünschten Vollständigkeitssatz müßte z. B. aus
$P^{\leftrightarrow}, CT \models (\texttt{Seite}\doteq\texttt{kopf} \rightarrow \texttt{mwerfen}(\texttt{Seite}))$
folgen, daß das Ziel $\texttt{mwerfen}(\texttt{Seite})$ eine erfolgreiche Ableitung mit dem Antwortconstraint C' hat, sodaß
$CT \models \forall(\texttt{Seite}\doteq\texttt{kopf} \rightarrow C')$.
Wenn Ableitung das Antwortconstraint $\texttt{Seite}\doteq\texttt{zahl}$ hat, dann gilt aber nicht
$CT \models \forall(\texttt{Seite}\doteq\texttt{kopf} \rightarrow \texttt{Seite}\doteq\texttt{zahl})$ $\diamond$

Anmerkung

Die Korrektheitssätze von erfolgreichen und erfolglosen Ableitungen vom CLP-Abschnitt gelten analog, aber die gewünschten Vollständigkeitssätze gelten nicht, weil man nicht weiß, welche Ableitung mit welcher Antwort durchgeführt wird und es daher die “falsche” Ableitung sein kann.

Je nachdem, welche Klausel ausgewählt wird, kommt es zu unterschiedlichen Ergebnissen. Der Ausgang von `mwerfen` ist nicht vorhersehbar. Insbesondere kann das Ziel `mwerfen(kopf)` scheitern oder erfolgreich sein.

Eine konkrete NCLP-Implementierung kann natürlich eine bestimmte, feste Selektionsregel verwenden.

Deterministisches NCLP

In *deterministischen* Programmen schließen sich die Wächter der Klauseln eines Prädikates gegenseitig aus:

$$
\begin{array}{l}
\{\texttt{min(X,Y,Z)} \leftarrow \texttt{X} \leq \texttt{Y} : \texttt{X} \doteq \texttt{Z} \mid \textit{true}, \\
\ \texttt{min(X,Y,Z)} \leftarrow \texttt{Y} < \texttt{X} : \texttt{Y} \doteq \texttt{Z} \mid \textit{true}\}
\end{array}
$$

Anmerkung

In deterministischen Programmen ist für jedes Atom die Reduktionsbedingung für höchstens eine Klausel erfüllbar. Endliche Ableitungen unterscheiden sich aufgrund der Stabilitätseigenschaft von NCLP dann nur in der Reihenfolge der Reduktionen, die Reduktionen der einzelnen Atome sind aber immer diesselben, ebenso wie die Antwortconstraints.

Mit einer Ausnahme führen die Zustände für `min` zu den gleichen Antwortconstraints. Zwar kann für das Ziel `min(A,A,B)` nur mehr die erste Klausel die Reduktionsbedingung erfüllen, das Antwortconstraint ist aber wie gehabt `A=B`. Auch der Zustand
$\langle \mathtt{min}(\mathtt{A},\mathtt{B},\mathtt{C}), \mathtt{A} \leq \mathtt{B} \rangle$ reduziert erwartungsgemäß zu $\mathtt{A} \leq \mathtt{B} \wedge \mathtt{A} = \mathtt{C}$. Jedoch kann der Zustand $\langle \mathtt{min}(\mathtt{A},\mathtt{B},\mathtt{C}), \mathtt{A} \geq \mathtt{B} \rangle$ nicht reduzieren, er ist verklemmt. Die deterministische Version von `min` ist also operational schwächer.

Vollständigkeit

Definition 3 Eine Ableitung ist *fair*, wenn sie entweder scheitert oder wenn jedes Atom, das in der Ableitung vorkommt, nach endlich vielen Reduktionen ausgewählt wird.

Theorem 2
Vollständigkeit von erfolgreichen Ableitungen [Maher87] Sei P ein deterministisches NCL-Programm und G ein Ziel mit mindestens einer fairen Ableitung. Wenn gilt $P^{\leftrightarrow}, CT \models C \rightarrow G$ und C in CT erfüllbar ist, dann hat jede erfolgreiche Ableitung von G ein Antwortconstraint C', sodaß

$$CT \models \forall (C \rightarrow C').$$

Fairness

Das Vollständigkeitstheorem gilt nicht, wenn G keine fairen Ableitungen hat:

$\{\texttt{min(X,Y,Z)} \leftarrow \texttt{X} \leq \texttt{Y} \mid \texttt{X} \doteq \texttt{Z},$
$\texttt{min(X,Y,Z)} \leftarrow \texttt{Y} \leq \texttt{X} \mid \texttt{Y} \doteq \texttt{Z} \}$

Sei G das Ziel `min(A,B,C)`. Es gilt

$$\mathcal{P}^{\leftrightarrow}, CT \models \forall(\texttt{A} \leq \texttt{B} \wedge \texttt{A} \doteq \texttt{C} \rightarrow G).$$

Aber der Anfangszustand $\langle G, \mathit{true} \rangle$ ist verklemmt (daher gibt es keine faire Ableitung, deren Antwort von $(\texttt{A} \leq \texttt{B} \wedge \texttt{A} \doteq \texttt{C})$ impliziert werden könnte)

Erfolglose Ableitungen

Theorem 3 Korrektheit und Vollständigkeit von erfolglosen Ableitungen [Maher87] Sei P ein deterministisches NCL-Programm und G ein Ziel mit mindestens einer fairen Ableitung. Dann sind folgende Aussagen äquivalent:

- $P^{\leftrightarrow}, CT \models \neg(\exists G)$
- G hat eine endlich gescheiterte Ableitung.
- Jede faire Ableitung von G scheitert endlich.