Ludwig-Maximilians-Universität München

Haupt-/Bachelorseminar Programmanalyse

Kontrollflussanalyse I

Joschka Rinke

10. Juli 2009

Prof. Martin Hofmann, Ph.D., Hans-Wolfgang Loidl

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Die folgende Ausarbeitung beschäftigt sich mit den Grundlagen der Kontrollflussanalyse.
Dafür wird die Arbeit in drei größere Kapitel unterteilt:

1. Abstrakte Kontrollflussanalyse
2. Strukturelle Operationale Semantik
3. Korrektheit der Kontrollflussanalyse

Zunächst wird der Sinn der Kontrollflussanalyse erläutert, ehe eine einfache funktionale Sprache, im folgenden einfach FUN genannt, eingeführt wird. Anhand der Sprache FUN werden im Laufe dieser Ausarbeitung mehrere Beispiele erläutert. Nach der Einführung in FUN wird sich die Arbeit mit der eigentlichen Kontrollflussanalyse beschäftigen.
Das Kapitel "Abstrakte Kontrollflussanalyse" soll eine erste genauere Einführung in die Thematik der Kontrollflussanalyse bieten und deren Wohldefiniertheit beweisen.
Im Kapitel "Strukturelle Operationale Semantik" werden Axiome für die Sprache FUN eingeführt, mit denen sich die semantische Korrektheit einer Analyse zeigen lässt.
Das Kapitel "Korrektheit der Kontrollflussanalyse" soll erklären, wie die eingeführten Axiome angewendet werden um die semantische Korrektheit einer Analyse zu zeigen. Ausserdem wird die Existenz einer Lösung und speziell die einer kleinsten, bewiesen.
Abschließend wird im Zuge des Nachweises der Existenz einer Lösung der Unterschied zwischen induktiven und coinduktiven Definitionen verdeutlicht.

1.1 Sinn der Kontrollflussanalyse

Die Kontrollflussanalyse ist eine statische Analyse zur Compile-Zeit und überprüft die Abarbeitungsreihenfolge von Programmteilen. Ziel der Kontrollflussanalyse ist es Informationen darüber zu erhalten, welcher "elementare Programmblock" zu welchem "elementaren Programmblock" führt. Das mag in einfachen Programmen vielleicht leicht ersichtlich sein, doch bei komplexen imperativen oder objekt-orientierten Programmen ist nicht immer intuitiv erkennbar, wie einzelne Programmblöcke zusammenhängen und sich gegenseitig beeinflussen. Häufig wird die Kontrollflussanalyse auch Constraint Based Analyses genannt, das folgende Beispiel eines funktionalen Programms soll zeigen weshalb:

1.1.1 Beispiel 1

```
let f = fn x ⇒ x 1;
        g = fn y ⇒ y+2;
        h = fn z ⇒ z+3
    in(f g) + (f h)
```

Das Programm definiert eine Funktion f höherer Ordnung mit dem formalen Parameter x und dem Rumpf x 1. Weiterhin werden die beiden Funktionen g und h definiert, die im Rumpf des let-Konstruktes als aktuelle Parameter an f übergeben werden. Semantisch gesehen wird x an beide Funktionen gebunden und an g und h wird der Wert 1 übergeben. Als Resultat des Programms erhält man dann den Wert 7.
Dabei wird die Kontrolle durch eine Reihung von Applikationen bis an x weitergereicht. Das Problem dabei ist, dass man genau genommen nicht direkt auf den Wert von x schließen kann, denn dafür muss man wissen welche Parameter an f übergeben werden. Die Kontrollflussanalyse ermöglicht es einem zu zeigen **welche Funktionen für jede vorkommende Applikation akzeptiert werden**. Es werden bei der Kontrollflussanalyse also mögliche Belegungen an Grenzen gebunden, daher der Name Constraint Based.

1.2 Die elementaren Programmblöcke

Da in funktionalen Programmiersprachen die elementaren Programmblöcke, die wir untersuchen möchten, häufig verschachtelt sind, ist es sinnvoll alle Unterausdrücke mit einem Label zu kennzeichnen. Betrachten wir das folgende Programm:

1.2.1 Beispiel 2

Das Programm ruft die Identitätsfunktion von x mit der Identitätsfunktion von y als Argument auf.

$$[[\text{fn x} \Rightarrow [\text{x}]^{1}]^{2}[\text{fn y} \Rightarrow [\text{y}]^{3}]^{4}]^{5}$$

Unter Berücksichtigung der Tatsache, dass wir alle elementaren Programmblöcke mit Labeln versehen ergeben sich die in diesem Beispiel rot markierten Labels.

Da wir nur an Informationen bezüglich der gelabelten Programmblöcke interessiert sind betrachten wir keine Seiteneffekte. Das Ergebnis einer Kontrollflussanalyse ist ein Funktionenpaar: $(\hat{C},\hat{\rho})$.

- $\hat{C}(l)$ enthält die Werte, die der elemenare Programmblock mit Label l annehmen kann.
- $\hat{\rho}(x)$ enthält die Werte, an die die Variable x gebunden werden kann.

1.3 Das Constraint System

Es gibt 3 Klassen von Constraints (Einschränkungen), die die folgenden Ergebnisse erzielen:

- Werte der funktionalen Abstraktionen an ihr Label binden
- Werte der Variablen an ihr Label binden
- Applikationen einschränken, indem formale Parameter des Ausdrucks an aktuelle Parameter der Applikation gebunden werden und zeigen, dass Resultate im Funktionsrumpf auch mögliche Resultate der Applikation sind.

Wie dabei genau vorgegangen wird, wird in Abschnitt 2 “Die abstrakte Kontrollflussanalyse“ gezeigt.

1.4 Die kleinste Lösung

Ziel der Kontrollflussanalyse ist es möglichst genaue Angaben über die Constraints zu machen. Dafür müssen die Wertebereiche für $\hat{C}$ und $\hat{\rho}$ möglichst weit eingeschränkt werden, denn je weniger Werte $\hat{C}$ und $\hat{\rho}$ annehmen können, desto genauere Aussagen darüber, welche Funktionen akzeptiert werden, lassen sich mit der Kontrollflussanalyse machen. Wie dabei genau vorgegangen wird, wird in den Abschnitten 2 “Die abstrakte Kontrollflussanalyse“ und 4 “Die Korrektheit der Kontrollflussanalyse“ gezeigt.

1.5 Die Syntax der Sprache FUN

Bei der Sprache FUN handelt es sich um eine funktionale Sprache, die die “Lambda Abstraktion“ um explizite Operatoren für Rekursion, Bedingungen und lokale Definitionen erweitert. Wir bezeichnen ein gelabeltes Programmstück als Expression und ein ungelabeltes Programmstück als Term.

e	$\in$ **Exp**	Expression oder gelabelter Term
t	$\in$ **Term**	Term oder ungelabelte Expression

Weiter nehmen wir an, dass die Anzahl der Variablen abzählbar ist und dass Konstanten, Binäroperatoren und Labels nicht spezifiziert werden:

f,x	$\in$ **Var**	Variablen
c	$\in$ **Const**	Konstanten
op	$\in$ **Const**	Binäroperatoren
l	$\in$ **Lab**	Labels

Nun haben wir die abstrakte Syntax der Sprache FUN wie folgt definiert:

e ::= t^l

t ::= $c|x|$fn x $\Rightarrow$ $e_0|$fun f x $\Rightarrow$ $e_0|e_1\,e_2|$ if e_0 then e_1 else e_2
$|$let x = e_1 in $e_2|e_1$ op e_2

fn x $\Rightarrow$ e_0 ist eine Lambda Abstraktion und definiert eine Funktion.
fun f x $\Rightarrow$ e_0 ist der rekursive Variante von fn x $\Rightarrow$ e_0, wobei alle Vorkommen von f in e_0 sich auf fun f x $\Rightarrow$ e_0 selbst beziehen.
Das Konstrukt let x = e_1 in e_2 ist eine nicht-rekursive Definition und ist äquivalent zu (fn x $\Rightarrow$ e_2)(e_1)
Wenn man nun alle klar definierten Expressions als elementaren Programmblock betrachtet ergeben sich für das Beispiel 2 die bereits eingeführten Labels, zur Erinnerung:

$$[[\text{fn x} \Rightarrow [\text{x}]^1]^2[\text{fn y} \Rightarrow [\text{y}]^3]^4]^5$$

Die zueinander passenden Farben der Labels und Ausdrücke markieren, welches Label welchen Programmblock eingrenzt.

2 Die abstrakte Kontrollflussanalyse

2.1 Einleitung

In diesem Kapitel wird die 0-CFA Analyse betrachtet, die einfachste mögliche Form der Kontrollflussanalyse. Bei dieser Form der Kontrollflussanalyse wird keinerlei Kontextinformation in Betracht gezogen, daher kommt auch der Name "0-CFA". Das Resultat einer 0-CFA Analyse ist, wie gereits erwähnt, das Funktionenpaar ($\hat{\text{C}}$,$\hat{\rho}$).

- $\hat{\text{C}}$ ist der abstrakte Cache, der abstrakte Werte an gelabelte Programmpunkte bindet.
- $\hat{\rho}$ ist die abstrakte Umgebung, die abstrakte Werte an Variablen bindet.

Verdeutlicht durch:

$\hat{v}$	$\in \hat{Val}$	$= P(\mathbf{Term})$	abstrakte Werte → Abstraktion von mehreren Funktionen
$\hat{\rho}$	$\in \hat{Env}$	$= \mathbf{Var} \to \hat{\mathbf{Val}}$	abstrakte Umgebung
$\hat{C}$	$\in \hat{Cache}$	$= \mathbf{Lab} \to \hat{\mathbf{Val}}$	abstraktes Cache

Offensichtlich können $\hat{C}$:**Lab** → $\hat{\mathbf{Val}}$ und $\hat{\rho}$:**Var** → $\hat{\mathbf{Val}}$ zu einer Einheit der Form (**Var**∪**Lab**) kombiniert werden.

2.2 Beispiel 3

Wir betrachten wieder den Ausdruck [[fn x ⇒ [x]^1]^2[fn y ⇒ [y]^3]^4]^5 aus Beispiel 2.
Jetzt betrachten wir drei Annahmen für eine mögliche 0-CFA Analyse:

Drei Analyseannahmen für [[fn x ⇒ [x]^1]^2[fn y ⇒ [y]^3]^4]^5

	$(\hat{C}_e,\hat{\rho}_e)$	$(\hat{C}_e{}',\hat{\rho}_e{}')$	$(\hat{C}_e{}'',\hat{\rho}_e{}'')$
1	{fn y ⇒ y^3}	{fn y ⇒ y^3}	{fn x ⇒ x^1,fn y ⇒ y^3}
2	{fn x ⇒ x^1}	{fn x ⇒ x^1}	{fn x ⇒ x^1,fn y ⇒ y^3}
3	∅	∅	{fn x ⇒ x^1,fn y ⇒ y^3}
4	{fn y ⇒ y^3}	{fn y ⇒ y^3}	{fn x ⇒ x^1,fn y ⇒ y^3}
5	{fn y ⇒ y^3}	{fn y ⇒ y^3}	{fn x ⇒ x^1,fn y ⇒ y^3}
x	{fn y ⇒ y^3}	∅	{fn x ⇒ x^1,fn y ⇒ y^3}
y	∅	∅	{fn x ⇒ x^1,fn y ⇒ y^3}

Die erste Annahme $(\hat{C}_e,\hat{\rho}_e)$ ist intuitiv richtig, denn für $\hat{C}(1)$ ist {fn y ⇒ y^3} eine mögliche Belegung, genauso ist für $\hat{C}(2)$ {fn x ⇒ x^1} eine gültige Belegung. Das lässt sich für die ersten beiden Annahmen so fortführen bis zu $\hat{\rho}(x)$. x hat für $(\hat{C}_e,\hat{\rho}_e)$ mit {fn y ⇒ y^3} eine gültige Belegung, wohingegen x für $(\hat{C}_e{}',\hat{\rho}_e{}')$ mit ∅ eine ungültige Belegung hat, denn diese Annahme hätte zur Folge, dass {fn x ⇒ x^1} nie aufgerufen wird, da $\hat{\rho}_e{}'(x)$ die Leeremenge ∅ als Belegung hat.
Die dritte Annahme ist intuitiv richtig, denn sie schränkt die Belegungen zwar nicht ein, macht aber auch nichts unerlaubtes und ist daher sehr unpräzise aber richtig.

2.3 Die Acceptable-Relation

Um zu zeigen, dass eine Annahme wirklich eine akzeptable 0-CFA Analyse für ein Programm ist müssen wir die Acceptable-Relation der Form $(\hat{C},\hat{\rho}) \models e$ einführen, mit deren Hilfe man leicht zeigen kann ob eine Annahme $(\hat{C},\hat{\rho})$ eine akzeptable Kontrollflussanalyse für den Ausdruck e ist. Die Acceptable-Relation "⊨" bildet das Tripel ($\hat{\mathbf{Cache}}$ x $\hat{\mathbf{Env}}$ x **Exp**) auf {*true,false*} ab und wird durch Klauseln definiert.

2.3.1 Die Klauseln der "Acceptable"-Relation

[***con***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash c^l$ always

[***var***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash x^l$ iff $\hat{\rho}(\mathrm{x}) \subseteq \hat{\mathrm{C}}(l)$

[***fn***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash (\mathrm{fn\ x} \Rightarrow e_0)^l$ iff $(\mathrm{fn\ x} \Rightarrow e_0) \subseteq \hat{\mathrm{C}}(l)$

[***fun***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash (\mathrm{fun}\ f\ \mathrm{x} \Rightarrow e_0)^l$ iff $(\mathrm{fun\ f\ x} \Rightarrow e_0) \subseteq \hat{\mathrm{C}}(l)$

[***op***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash (t_1^{l_1}\ op\ t_2^{l_2})^l$ iff $(\hat{\mathrm{C}},\hat{\rho}) \vDash t_1^{l_1} \wedge (\hat{\mathrm{C}},\hat{\rho}) \vDash t_2^{l_2}$

2.3.2 Beschreibung [*con*], [*var*], [*fn*], [*fun*], [*op*]

Dabei ist die Klausel [*con*] erfüllt unter den Bedingungen, dass wir keine konkreten Werte betrachten und dass Konstanten keine Funktionen sind.
Die Klausel [*var*] bindet die abstrakte Umgebung an den abstrakten Cache, das heisst, dass alle Werte, die die Variable x annehmen kann im Cache mit dem Label l enthalten sein müssen(l:$\hat{\rho}(x) \subseteq \hat{\mathrm{C}}(l)$).
Für die Gültigkeit der Klauseln [*fn*] und [*fun*] muss lediglich der Funktionsterm ($\mathrm{fn\ x} \Rightarrow e_0$ oder $\mathrm{fun}\ f\ \mathrm{x} \Rightarrow e_0$) in $\hat{\mathrm{C}}(l)$ enthalten sein, die Gültigkeit für den Rumpf einer Funktion weisst die Klausel [*app*] nach.
Dass eine Annahme über zwei Terme, die mittels eines Operatoren verknüpft sind, unter der Bedingung, dass die einzelnen Terme für die Annahme Gültigkeit besitzen, gültig ist, besagt die Klausel [*op*].

[***if***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash (\mathrm{if}\ t_0^{l_0}\ \mathrm{then}\ t_1^{l_1}\ \mathrm{else}\ t_2^{l_2})^l$

$$\begin{aligned}&\text{iff } (\hat{\mathrm{C}},\hat{\rho}) \vDash t_0^{l_0} \wedge \\ &\qquad (\hat{\mathrm{C}},\hat{\rho}) \vDash t_1^{l_1} \wedge (\hat{\mathrm{C}},\hat{\rho}) \vDash t_2^{l_2} \wedge \\ &\qquad \hat{\mathrm{C}}(l_1) \subseteq \hat{\mathrm{C}}(l) \wedge \hat{\mathrm{C}}(l_2) \subseteq \hat{\mathrm{C}}(l)\end{aligned}$$

[***let***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash (\mathrm{let\ x} = t_1^{l_1}\ \mathrm{in}\ t_2^{l_2})^l$

$$\begin{aligned}&\text{iff } (\hat{\mathrm{C}},\hat{\rho}) \vDash t_1^{l_1} \wedge (\hat{\mathrm{C}},\hat{\rho}) \vDash t_2^{l_2} \wedge \\ &\qquad \hat{\mathrm{C}}(l_1) \subseteq \hat{\rho}(x) \wedge \hat{\mathrm{C}}(l_2) \subseteq \hat{\mathrm{C}}(l)\end{aligned}$$

2.3.3 Beschreibung [*if*], [*let*]

Da die Klauseln [*if*] und [*let*] rekursive Aufrufe enthalten, müssen ihre Unterausdrücke ebenfalls ausgewertet und die möglichen Werte dieser Unterausdrücke an die möglichen Werte des Hauptausdrucks gebunden werden. Daher werden in der Klauseldefinition alle enthaltenen

Terme seperat auf ihre Gültigkeit geprüft. Dies geschieht in der Definition unter Anwendung der Klausel [*var*]. Das Zusammenspiel der Klauseln [*let*] und [*var*] an einem Bild veranschaulicht:

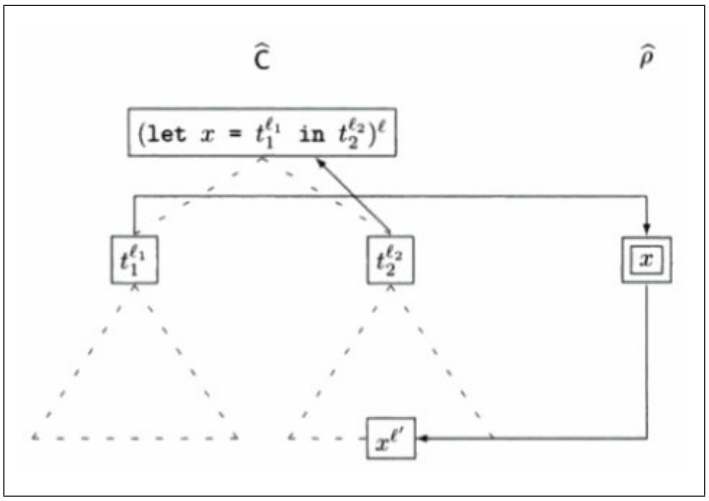

Abbildung 1: Eine bildliche Darstellung der Klauseln [*let*] und [*var*]

$$
\begin{aligned}
&[\textbf{\textit{app}}\ ]\ (\hat{\mathrm{C}},\hat{\rho}) \vDash (t_1^{l_1} t_2^{l_2})^{l} \\
&\qquad \text{iff } (\hat{\mathrm{C}},\hat{\rho}) \vDash t_1^{l_1} \wedge (\hat{\mathrm{C}},\hat{\rho}) \vDash t_2^{l_2} \wedge \\
&\qquad\qquad (\forall(\text{fn } x \Rightarrow t_0^{l_0})) \in \hat{\mathrm{C}}(l_1): \\
&\qquad\qquad\qquad (\hat{\mathrm{C}},\hat{\rho}) \vDash t_0^{l_0} \wedge \\
&\qquad\qquad\qquad \hat{\mathrm{C}}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{\mathrm{C}}(l_0) \subseteq \hat{\mathrm{C}}(l) \wedge \\
&\qquad\qquad (\forall(\text{fun } f\ x \Rightarrow t_0^{l_0})) \in \hat{\mathrm{C}}(l_1): \\
&\qquad\qquad\qquad (\hat{\mathrm{C}},\hat{\rho}) \vDash t_0^{l_0} \wedge \\
&\qquad\qquad\qquad \hat{\mathrm{C}}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{\mathrm{C}}(l_0) \subseteq \hat{\mathrm{C}}(l) \wedge \\
&\qquad\qquad\qquad \{\text{fun } f\ x \Rightarrow t_0^{l_0}\} \subseteq \hat{\rho}(f))
\end{aligned}
$$

2.3.4 Beschreibung [*app*]

Wie die oben beschriebenen Klauseln [*let*] und [*if*] enthält auch die Klausel [*app*] rekursive Aufrufe, die vorraussetzen, dass sowohl der Operator $t_1^{l_1}$ als auch der Operand $t_2^{l_2}$ unter Verwendung einer Annahme $(\hat{\mathrm{C}},\hat{\rho})$ Gültigkeit besitzen. Dies wird nachgewiesen, indem zum einen für alle Terme fn $x \Rightarrow t_0^{l_0}$, die den Operator (l_1) erreichen, vorausgesetzt wird, dass der aktuelle Parameter (l_2) an den formalen Parameter (x) gebunden ist und zum anderen, dass das Resultat der

Funktion (l_0) an das Resultat der Applikation selbst (l) gebunden ist. Natürlich muss auch der eigentliche Rumpf der Funktion unter der Annahme $(\hat{C},\hat{\rho}) \vDash t_0^{l_0}$ Gültigkeit besitzen.

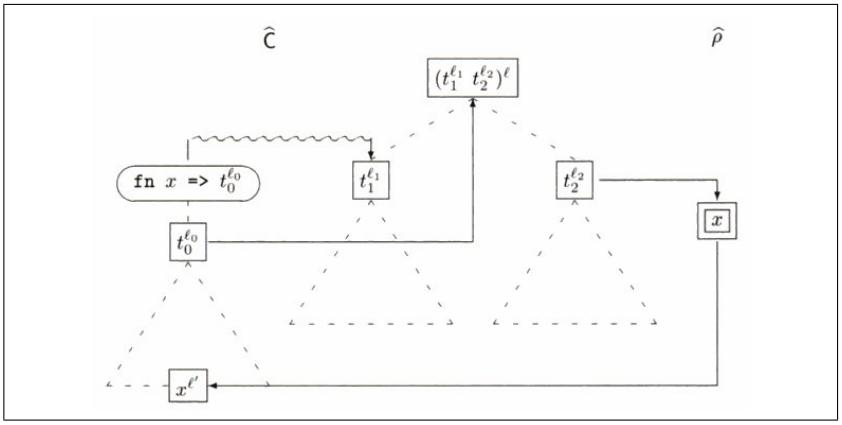

Abbildung 2: Eine bildliche Darstellung der Klauseln [*app*], [*fn*] und [*var*]

2.4 Beispiel 4

Gegeben sei wieder der Ausdruck aus Beispiel 2. Nun zeigen wir, dass die Annahme $(\hat{C}_e,\hat{\rho}_e)$ in Tabellenspalte 1 aus Beispiel 3 eine gültige Annahme ist:

$$(\hat{C}_e,\hat{\rho}_e) \vDash [[\text{fn x} \Rightarrow [\text{x}]^1]^2[\text{fn y} \Rightarrow [\text{y}]^3]^4]^5$$

Nach Verwendung der Klausel [*app*] und mit $\hat{C}_e(2) = \{\text{fn x} \Rightarrow \text{x}^1\}$ müssen wir noch folgendes prüfen:

1.) $(\hat{C}_e,\hat{\rho}_e) \vDash \{\text{fn x} \Rightarrow \text{x}^1\}^2$
2.) $(\hat{C}_e,\hat{\rho}_e) \vDash \{\text{fn y} \Rightarrow \text{y}^3\}^4$
3.) $(\hat{C}_e,\hat{\rho}_e) \vDash \text{x}^1$
4.) $\hat{C}_e(4) \subseteq \hat{\rho}_e(\text{x})$
5.) $\hat{C}_e(1) \subseteq \hat{C}_e(5)$

1.) & 2.) werden unter Anwendung der Klausel [*fn*] gezeigt. 3.) wird unter Anwendung der Klausel [*var*] gezeigt. Der Wert, an den x gebunden wird, ist nach Beispiel 3 $\{\text{fn y} \Rightarrow \text{y}^3\}$ und somit enthalten in $\hat{C}_e(4)$. Und die Gültigkeit für $\hat{C}_e(1) \subseteq \hat{C}_e(5)$ ist leicht aus der Angabe ersichtlich.
Wir erinnern uns, dass die Annahme $(\hat{C}_e\acute{},\hat{\rho}_e\acute{})$ in Tabelle 1 intuitiv ungültig war. Unter Verwendung der [*app*]-Klausel können wir dies nun auch formal nachweisen, denn es gilt $\hat{C}_e(4) \not\subseteq \hat{\rho}_e\acute{}(x)$ und daher ist die Annahme nicht akzeptabel.

2.5 Die Wohldefiniertheit der Analyse

Wir müssen die Relation "$\models$" coinduktiv, als größten Fixpunkt eines Funktionals, definieren, denn die Klausel *app* erlaubt keine strukturelle Induktion über e. Der Grund dafür ist, dass die Klausel *app* die Gültigkeit von $(\hat{C},\hat{\rho})$ für den Ausdruck $t_0^{l_0}$, der kein Unterausdruck der Applikation $(t_1^{l_1} t_2^{l_2})^l$ ist, voraussetzt. Würden wir die "Acceptable"-Relation induktiv, über den *kleinsten* Fixpunkt, definieren wäre diese Vorraussetzung nicht erfüllt. Durch die Definition über den größten Fixpunkt eines Funktionals greift man möglichst weit und kann so auch die Gültigkeit für die Klausel *app* feststellen. Der Abschnitt 4.3 Coinduktion vs Induktion geht genauer darauf ein.
Im folgenden wird lediglich eine formale Definition für die Relation "$\models$" eingeführt. Q sei ein Funktional, das heisst, dass sowohl das Argument, als auch das Ergebnis von Q Funktionen sind. Die Funktion sei wie folgt definiert:

$$\begin{aligned} Q{:}((\hat{\mathbf{Cache}} \text{ x } \hat{\mathbf{Env}} \text{ x } \mathbf{Exp}) &\to \{true, false\}) \\ &\to ((\hat{\mathbf{Cache}} \text{ x } \hat{\mathbf{Env}} \text{ x } \mathbf{Exp}) \to \{true, false\}) \end{aligned}$$

Daraus folgt, dass das konstruierte Funktional Q eine monotone Funktion der Form

$$((\hat{\mathbf{Cache}} \text{ x } \hat{\mathbf{Env}} \text{ x } \mathbf{Exp}) \to \{true, false\}, \sqsubseteq)$$

ist, wobei die Ordnungsfunktion $\sqsubseteq$ wie folgt definiert ist:

$$Q_1 \sqsubseteq Q_2 \text{ iff } \forall\ (\hat{C},\hat{\rho},e){:}(Q_1(\hat{C},\hat{\rho},e){=}true) \Rightarrow (Q_2(\hat{C},\hat{\rho},e){=}true)$$

Das heisst, dass alle untergeordneten Funktionen Q_2 auf den gleichen Wert abgebildet werden, wie übergeordnete Funktionen. Daher ist es sinnvoll "$\models$" coinduktiv als den größten Fispunkt zu definieren, so dass alle in diesem Funktional enthaltenen Funktionen auf den gleichen Wert abgebildet werden.

3 Die strukturelle operationale Semantik

In diesem Kapitel werden Axiome zur strukturellen operationalen Semantik eingeführt, mit denen sich anhand von Transitionen zeigen lässt, dass die Informationen aus der Analyse tatsächlich eine korrekte und sichere Beschreibung des Programmablaufs sind. Die Anwendung der Axiome wird im 4ten Kapitel über die Korrektheit der Kontrollflussanalyse verdeutlicht, ausserdem weisen wir dann die Existenz einer "kleinsten" Lösung nach.

3.1 Eine Erweiterung der Sprache FUN

Wir müssen die Sprache FUN um eine strukturelle operationale Semantik erweitern, die die Identität von Funktionen (und im speziellen abstrakten Werten) während der Programmausführung wahrt. Dafür muss die Funktion an "Closures" in ihrer Umgebung gebunden werden und in der Umgebung werden die freien Variablen der Funktion an Werte gebunden.

v	$\in$ **Val**	Werte
ρ	$\in$ **Env**	Umgebung

Die folgendermaßen definiert sind:

$$\begin{array}{lll} v & ::= & c|\text{close } t \text{ in } p \\ \rho & ::= & []|\rho[x \rightarrow v] \end{array}$$

Weiterhin werden Zwischenausdrücke (intermediate Expressions) und Zwischenterme (intermediate Terms) benötigt, die die Zwischenzustände, die sich auf dem Weg zu Closures ergeben, beschreiben. Natürlich ist eigentlich jeder Ausdruck e ein Zwischenausdruck ie, aber da wir den Rumpf einer Funktion nicht auswerten bevor sie aufgerufen wird, bezeichnen wir Funktionsrümpfe weiterhin als Ausdrücke e.

ie	$\in$ **IExp**	Zwischenausdruck
it	$\in$ **ITerm**	Zwischenterm

Dadurch wird die Syntax für Ausdrücke und Terme wie folgt erweitert:

$$\begin{array}{lll} ie & ::= & it^l \\ it & ::= & c|x|\text{fn } x \Rightarrow e_0|\text{fun } f\ x \Rightarrow e_0|ie_1 ie_2 \\ & & |\text{if } ie_0 \text{ then } ie_1 \text{ else } ie_2|\text{let } x = ie_1 \text{ in } ie_2|ie_1\ op\ ie_2 \\ & & |\text{bind } \rho \text{ in } ie|\text{close } t \text{ in } \rho \end{array}$$

Das bind-Konstrukt stellt sicher, dass der Zwischenausdruck ie in der Umgebung ρ ausgeführt wird. Im Gegensatz zu dem close-Konstrukt, das ein komplett ausgewerteter Ausdruck ist, betrachten wir bei bind Zwischenausdrücke, die sich, wie bereits erwähnt, auf dem Weg zu Closures ergeben.

3.2 Axiome und Transitions-Regeln Teil 1

Die Transitions-Regeln, sind alle von der Form $\rho \vdash ie_1 \rightarrow ie_2$. Die Idee dahinter ist, dass ein Ausführungsschritt ie_1 in der Umgebung ρ nach ie_2 transformiert wird. Ein Ausdruck der Form "ie" ist noch nicht ausgewertet worden, ein Ausdruck der Form "ie'" wurde ausgewertet. Ein Ausdruck der Form "close(t)in ρ" erzeugt ein Closure, das sicherstellt, dass t in ρ enthalten ist.

$[\boldsymbol{var}\ ]\ \rho \vdash x^l \to v^l$ if $x \in dom(\rho)$ and $v = \rho(x)$

$[\boldsymbol{fn}\ ]\ \rho \vdash (\text{fn } x \Rightarrow e_0)^l \to (\text{close}(\text{fn } x \Rightarrow e_0) \text{ in } \rho_0)^l$
where $\rho_0 = \rho \mid FV(\text{fn } x \Rightarrow e_0)$

$[\boldsymbol{fun}\ ]\ \rho \vdash (\text{fun } f\ x \Rightarrow e_0)^l \to (\text{close}(\text{fun } f\ x \Rightarrow e_0) \text{ in } \rho_0)^l$
where $\rho_0 = \rho \mid FV(\text{fun } f\ x \Rightarrow e_0)$

3.2.1 Beschreibung [*var*], [*fn*], [*fun*]

Das Axiom [*var*] zeigt, dass der Wert einer Variable an die Umgebung gebunden ist. Die beiden Axiomen [*fn*] und [*fun*] konstruieren die Closures, das heisst, dass die Umgebung ρ mit den freien Variablen verkoppelt wird.

$$[\boldsymbol{app}_1\ ]\ \frac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1 ie_2)^l \to (ie_1' ie_2)^l}$$

$$[\boldsymbol{app}_2\ ]\ \frac{\rho \vdash ie_2 \to ie_2'}{\rho \vdash (v_1^{l_1} ie_2)^l \to (v_1^{l_1} ie_2')^l}$$

$[\boldsymbol{app}_{fn}\ ]\ \rho \vdash ((\text{close}(\text{fn } x \Rightarrow e_1) \text{ in } \rho_1)^{l_1}\ v_2^{l_2})^l \to$
$(\text{bind } \rho_1[x \to v_2] \text{ in } e_1)^l$

$[\boldsymbol{app}_{fun}\ ]\ \rho \vdash ((\text{close}(\text{fun } f\ x \Rightarrow e_1) \text{ in } \rho_1)^{l_1}\ v_2^{l_2})^l \to$
$(\text{bind } \rho_1[x \to v_2] \text{ in } e_1)^l$
where $\rho_2 = \rho_1[f \to \text{close}(\text{fun } f\ x \Rightarrow e_1) \text{ in } \rho_1]$

3.2.2 Beschreibung [$\boldsymbol{app}_1$], [$\boldsymbol{app}_2$], [$\boldsymbol{app}_{fn}$], [$\boldsymbol{app}_{fun}$]

Bei Applikationen wird erst durch Anwendung von $[app_1]$ der Operator, dann durch Anwendung von $[app_2]$ der Operand ausgewertet. Dann wird, je nach Art der Funktion, entweder $[app_{fn}]$ oder $[app_{fun}]$ verwendet um den aktuellen Parameter an den formalen Parameter zu binden. Im Fall von $[app_{fun}]$ wird ausserdem die Rekursion behandelt, so dass spätere rekursive Aufrufe korrekt gebunden werden.

$$[\boldsymbol{bind}_1\ ]\ \frac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (\text{bind } \rho_1 \text{ in } ie_1)^l \to (\text{bind } \rho_1 \text{ in } ie_1')^l}$$

$[\boldsymbol{bind}_2\ ]\ \rho \vdash (\text{bind } \rho_1 \text{ in } v_1^{l_1})^l \to v_1^l$

3.2.3 Beschreibung [$bind_1$], [$bind_2$]

Durch die Verwendung der [*bind*]-Konstrukte wird sichergestellt, dass der Rumpf einer Funktion in der passenden Umgebung enthalten ist. Das [*bind*]-Konstrukt wird durch Anwendung von [$bind_1$] ausgeführt. Das Ergebnis der Applikation erhält man schließlich durch Anwendung von [$bind_2$].

3.3 Beispiel 5

Wir betrachten wieder den Ausdruck $((\text{fn x} \Rightarrow \text{x}^1)^2(\text{fn y} \Rightarrow \text{y}^3)^4)^5$, der unter der Umgebung [], die, wenn ein Ausdruck korrekt ist immer für diesen Ausdruck eine gültige Umgebung ist, akzeptabel ist.

$$
\begin{array}{lll}
[] \vdash & ((\text{fn x} \Rightarrow \text{x}^1)^2(\text{fn y} \Rightarrow \text{y}^3)^4)^5 & \\
\rightarrow & ((\text{close}(\text{fn x} \Rightarrow \text{x}^1) \text{ in } [])^2(\text{fn y} \Rightarrow \text{y}^3)^4)^5 & [fn],\ [app_1] \\
\rightarrow & ((\text{close}(\text{fn x} \Rightarrow \text{x}^1) \text{ in } [])^2(\text{close}(\text{fn y} \Rightarrow \text{y}^3) \text{ in } [])^4)^5 & [fn],\ [app_2] \\
\rightarrow & (\text{bind}[\text{x} \rightarrow (\text{close}(\text{fn y} \Rightarrow \text{y}^3) \text{ in } [])] \text{ in } \text{x}^1)^5 & [app_{fn}] \\
\rightarrow & (\text{bind}[\text{x} \rightarrow (\text{close}(\text{fn y} \Rightarrow \text{y}^3) \text{ in } [])] \text{ in} & \\
 & \quad (\text{close}(\text{fn y} \Rightarrow \text{y}^3) \text{ in } [])^1)^5 & [bind_1],\ [var] \\
\rightarrow & (\text{close}(\text{fn y} \Rightarrow \text{y}^3) \text{ in } [])^5 & [bind_2]
\end{array}
$$

Als erstes verwenden wir [*fn*], [app_1] um den Operator, dann [*fn*], [app_2] um den Operand auszuwerten. Durch die Anwendung von [app_{fn}] wird das [*bind*]-Konstrukt eingeführt, das eine Closure für die lokale Umgebung [x $\rightarrow$(close(fn y $\Rightarrow$ y^3) in [])] einführt, die wir benötigen um den Rumpf betrachten zu können. Dadurch können wir nun x^1, den Rumpf, unter Verwendung von [$bind_1$] und [*var*] ausführen um dann unter Verwendung von [$bind_2$] das Resultat zu erhalten und die lokale Umgebung zu verlassen.

3.4 Axiome und Transitions-Regeln Teil 2

[$\boldsymbol{if}_1$] $\dfrac{\rho \vdash ie_0 \rightarrow ie_0'}{\rho \vdash (\text{if } ie_0 \text{ then } e_1 \text{ else } e_2)^l \rightarrow (\text{if } ie_0' \text{ then } e_1 \text{ else } e_2)^l}$

[$\boldsymbol{if}_2$] $\rho \vdash (\text{if true}^{l_0} \text{ then } t_1^{l_1} \text{ else } t_2^{l_2})^l \rightarrow t_1^l$

[$\boldsymbol{if}_3$] $\rho \vdash (\text{if false}^{l_0} \text{ then } t_1^{l_1} \text{ else } t_2^{l_2})^l \rightarrow t_2^l$

3.4.1 Beschreibung [$\boldsymbol{if}_1$], [$\boldsymbol{if}_2$], [$\boldsymbol{if}_3$]

Zuerst wird unter Verwendung von [if_1] der Bedingungszweig ausgewertet, ehe dann der passende Zweig unter Verwendung von [if_2] oder [if_3] ausgewertet werden kann.

$[\boldsymbol{let}_1\ ]\ \dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (\text{let } x = ie_1 \text{ in } e_2)^l \to (\text{let } x = ie_1' \text{ in } e_2)^l}$

$[\boldsymbol{let}_2\ ]\ \rho \vdash (\text{let } x = v^{l_1} \text{ in } e_2)^l \to (\text{bind } \rho_0[x \to v] \text{ in } e_2)$
where $\rho_0 = \rho|FV(e_2)$

3.4.2 Beschreibung $[\boldsymbol{let}_1]$, $[\boldsymbol{let}_2]$

Unter Verwendung von $[let_1]$ wird die zu bindende Variable ausgwertet ehe unter Verwendung von $[let_2]$ ein $[bind]$-Konstrukt eingeführt wird, das zeigt, dass der Rumpf des $[let]$-Konstruktes bei seiner Auswertung an eine erweiterte Umgebung gebunden werden muss. Dann können wieder die Konstrukte $[bind_1]$ und $[bind_2]$ angewendet werden um das Resultat zu erzeugen.

$[\boldsymbol{op}_1\ ]\ \dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (ie_1\ op\ ie_2)^l \to (ie_1'\ op\ ie_2)^l}$

$[\boldsymbol{op}_2\ ]\ \dfrac{\rho \vdash ie_1 \to ie_1'}{\rho \vdash (v_1^{l_1}\ op\ ie_2)^l \to (v_1^{l_1}\ op\ ie_2')^l}$

$[\boldsymbol{op}_3\ ]\ \rho \vdash (v_1^{l_1}\ op\ v_2^{l_2})^l \to v^l \qquad \text{if } v = v_1\ \mathbf{op}\ v_2$

3.4.3 Beschreibung $[\boldsymbol{op}_1]$, $[\boldsymbol{op}_2]$, $[\boldsymbol{op}_3]$

Im Fall von Binärausdrücken werden erst die Argumente unter Verwendung von $[op_1]$ und $[op_2]$ ausgewertet, ehe die Operation **op** an sich unter Verwendung von $[op_3]$ ausgeführt werden kann.

4 Die Korrektheit der Kontrollflussanalyse

Wir können die semantische Korrektheit der Kontrollflussanalyse auch als Subject Reduction Result, also ein Ergebnis, dass wir unter Verwendung von vereinfachenden Klauseln erhalten, bezeichnen. Die Kernaussage ist die, dass eine einmal gültige Analyse auch nach der Programmausführung gültig bleibt.

4.1 Die Analyse von Zwischenausdrücken

Um die semantische Korrektheit nachweisen zu können, müssen wir unsere in Abschnitt “Die abstrakte Kontrollflussanalyse“ eingeführten Klauseln um Klauseln, die Zwischenausdrücke behandeln, erweitern.

4.1.1 Die Klauseln [*bind*] und [*close*]

[***bind***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash (\text{bind } \rho \text{ in } it_0^{l_0})^l$
$$\text{iff } (\hat{\mathrm{C}},\hat{\rho}) \vDash it_0^{l_0} \wedge \hat{\mathrm{C}}(l_0) \subseteq \hat{\mathrm{C}}(l) \wedge \rho \; R \; \hat{\rho}$$

[***close***] $(\hat{\mathrm{C}},\hat{\rho}) \vDash (\text{close } t_0 \text{ in } \rho)^l$
$$\text{iff } \{t_0\} \subseteq \hat{\mathrm{C}}(l) \wedge \rho \; R \; \hat{\rho}$$

Die Klausel [*bind*] zeigt, dass der Rumpf ausgeführt wird und das Ergebnis der Ausführung auch immer ein möglicher Wert für das Konstrukt ist. Das gilt insbesondere unter der Bedingung, dass die lokale Umgebung ρ in einer Relation R mit der abstrakten Umgebung $\hat{\rho}$ steht.
Für die Klausel [*close*] gilt, dass der Term t auf den das Closure angewendet wird ein möglicher Wert des Konstrukts sein muss und, dass wie bei [*bind*] die Umgebungen wieder in einer Relation R stehen.

4.1.2 Die Korrektheitsrelation R

Der Zweck der verwendeten abstrakten Umgebung $\hat{\rho}$ ist der, dass darin alle während der Ausführung auftretenden lokalen Umgebungen ρ enthalten sind. Daher setzen wir für alle lokalen Umgebungen ρ, die in Zwischenausdrücken vorkommen, das Verhältnis $\rho \; R \; \hat{\rho}$ vorraus. Dieses Verhältnis bezeichnen wir als Korrektheitsrelation R, die wir so definieren:

$$R{:}(\mathbf{Env} \text{ x } \hat{\mathbf{Env}}) \rightarrow \{true, false\}$$

Die Relation R ist wohldefiniert, denn jeder rekursive Aufruf wird unter einer lokalen Umgebung ausgeführt, die sicher *kleiner* als die Umgebung der aufrufenden, äusseren Funktion.

Theorem 4.1 *Wenn $\rho \; R \; \hat{\rho}$ und $\rho \vdash ie \rightarrow ie'$ dann impliziert $\hat{C},\hat{\rho}) \vDash ie$ auch $\hat{C},\hat{\rho}) \vDash ie'$.*

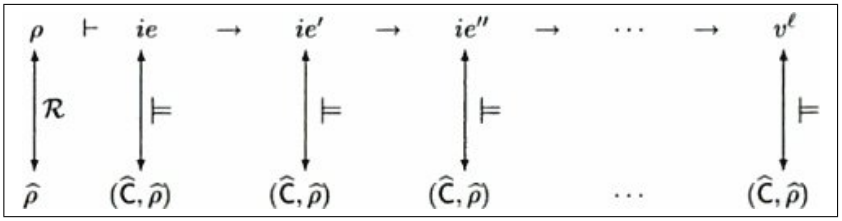

Abbildung 3: Erhalt der Gültigkeit bei Anwendung der "Acceptable"-Klauseln

Existiert für ein Programm eine Ausführung der Art, dass eine Funktion an einem bestimmten Punkt zu einer Abstrktion wird, dann muss diese

Abstraktion enthalten sein in der Menge aller möglichen Abstraktionen, die unter der Analyse auftreten können. Das ist immer der Fall, wenn das Programm durch Closures abgeschlossen ist, insbesondere, wenn es keine freien Variablen FV enthält, denn dann ist ρ [] und die Bedingung ρ R $\hat{\rho}$ trivialerweise erfüllt. Das Theorem besagt, dass eine einmal gültige Analyse auch nach Ausführung des Programms gültig bleibt, daher müssen wir uns nicht auf die Existenz einer "kleinsten" oder "besten" Lösung festlegen um das Resultat einer Analyse zu formulieren. Vielmehr ist eine einmal gefundene Lösung entweder akzeptabel oder, wenn es keine Lösung ist, eben nicht.
Weiterhin gilt:
Wenn $(\hat{\mathrm{C}},\hat{\rho}) \vDash it^{l_1}$ gilt und $\hat{\mathrm{C}}(l_1) \subseteq \hat{\mathrm{C}}(l_2)$, dann gilt auch $(\hat{\mathrm{C}},\hat{\rho}) \vDash it^{l_2}$
Dies kann leicht anhand struktureller Induktion über den Folgebaum von $\rho \vdash ie \rightarrow ie'$ nachgewiesen werden (siehe Abbildung 3).

4.2 Die Existenz einer Lösung

Unter Betrachtung der Analysemethode stellt sich einem die Frage, ob es für jeden Ausdruck e eine gültige Kontrollflussanalyse gibt.
Und weiter, ob es unter den vielen unterschiedlichen Analysen eine "kleinste" gültige Kontrollflussanalyse gibt.
Die Antwort auf diese beiden Fragen ist zweimal die Gleiche, nämlich "ja".
Um zu beweisen, dass es auch wirklich eine kleinste Lösung gibt, müssen wir die Menge einer Moore Familie einführen.

4.2.1 Die Moore Familie

Eine Teilmenge Y einer Menge $L = (L, \sqsubseteq)$ ist eine Moore Familie, genau dann,

$$\text{wenn } (\sqcap\, Y') \in Y \text{ für alle } Y' \subseteq Y \text{ gilt.}$$

Das heisst, dass die Lösung unter dem Schnitt geschlossen ist, also der Schnitt von Lösungen wieder eine Lösung ist.
Daraus können wir schließen, dass folgende Proposition gilt:

Proposition 4.2 *Für alle ie* $\in$ ***IExp*** *gilt, dass die Menge* $\{(\hat{C},\hat{\rho}) | (\hat{C},\hat{\rho}) \vDash ie\}$ *eine Moore Familie ist.*

Intuitiv können wir daraus schließen, dass alle Zwischenausdrücke ie eine gültige Kontrollflussanalyse besitzen. Denn wenn Y' die leere Menge ist, dann ist $\sqcap Y'$ ein Element von $\{(\hat{\mathrm{C}},\hat{\rho}) | (\hat{\mathrm{C}},\hat{\rho}) \vDash ie\}$, das heisst, dass es mindestens eine gültige Analyse gibt.
Und wenn Y' die Menge aller gültigen Analysen $\{(\hat{\mathrm{C}},\hat{\rho}) | (\hat{\mathrm{C}},\hat{\rho}) \vDash ie\}$ ist, dann ist $\sqcap Y'$ in dieser Menge enthalten und somit auch eine gültige Analyse für

ie. Daraus folgt, dass $\sqcap Y' \sqsubseteq (\hat{C},\hat{\rho})$ für alle anderen Analysen $(\hat{C},\hat{\rho})$ gilt und daher die kleinste gültige Anaylse ist, womit wir die Existenz einer kleinsten gültigen Analyse nachgewiesen haben.
Formal lässt sich diese Aussage coinduktiv beweisen, da wir auf eine coinduktive Definition Bezug nehmen.

4.2.2 Beispiel 6

Analyseergebnisse für $((\text{fn x} \Rightarrow \text{x}^1)^2(\text{fn y} \Rightarrow \text{y}^3)^4)^5$

	$(\hat{C}_e,\hat{\rho}_e)$	$(\hat{C}_e{}',\hat{\rho}_e{}')$	$(\hat{C}_e{}'',\hat{\rho}_e{}'')$
1	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$
2	$\{\text{fn x} \Rightarrow \text{x}^1\}$	$\{\text{fn x} \Rightarrow \text{x}^1\}$	$\{\text{fn x} \Rightarrow \text{x}^1\}$
3	$\emptyset$	$\{\text{fn x} \Rightarrow \text{x}^1\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$
4	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$
5	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$
x	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$
y	$\emptyset$	$\{\text{fn x} \Rightarrow \text{x}^1\}$	$\{\text{fn y} \Rightarrow \text{y}^3\}$

Wenn wir die drei Analysen genau vergleichen erkennen wir leicht, dass der Schnitt aus den Analysen in Spalte 2) und 3) wieder eine Lösung ist, nämlich die Analyse in Spalte 1). Die erste Analyse ist also der Schnitt aus den Analysen 2) und 3) und deshalb, wie soeben gezeigt, wieder eine gültige Lösung.

4.3 Coinduktion vs Induktion

Nun zeigen wir weshalb es so wichtig ist zwischen Induktion und Coinduktion zu unterscheiden und weshalb wir für die Acceptable-Relation der Kontrollflussanalyse stets von der coinduktiven Definition ausgegangen sind.
Wir erinnern uns, dass wir die Relation "$\models$" als den *größten* Fixpunkt eines Funktionals Q definiert haben.
Alternativ hätten wir "$\models'$" auch induktiv als den *kleinsten* Fixpunkt eines Funktionals Q definieren können.
Dann würden wir aber unter Betrachtung der Definition einer Moore Familie die folgende Proposition erhalten:

Proposition 4.3 *Es existiert ein e_*, so dass $\{(\hat{C},\hat{\rho}) | (\hat{C},\hat{\rho}) \models' ie_*\}$ keine Moore Familie ist.*

Das würde bedeuten, dass es Ausdrücke gibt, deren Lösungsmenge keine Moore Familie ist. Diese Spezifikation ist zum Beispiel für die Selbstapplikation nicht hinreichend. Um also die "Acceptable"-Relation so zu definieren, dass sie für alle Ausdrücke und insbesondere auch für die Selbstapplikation gültig ist, müssen wir die "Acceptable"-Relation coinduktiv als den *größten* Fixpunkt definieren.

5 Zusammenfassung

Zu Beginn haben wir die 0-CFA Analyse betrachtet, die einfachste Variante der Kontrollflussanalyse, bei der keinerlei kontextsensitive Informationen betrachtet werden. Als Ergebnis einer 0-CFA Analyse haben wir ein Paar $(\hat{C},\hat{\rho})$ erhalten, dessen Gültigkeit wir unter Einführung der "Acceptable"-Relation überprüfen konnten. Die 0-CFA Analyse ermöglicht es uns statische Informationen über das Zusammenspiel einzelner Programmblöcke zu untersuchen. Weiterhin haben wir nachgewiesen, dass die Analyse wohldefiniert ist, unter der Bedingung, dass wir die "Acceptable"-Relation coinduktiv definieren.
Unter Betrachtung der strukturellen operationalen Semantik, deren Axiome und Transitionen wir eingeführt haben, konnten wir zeigen, dass es, unter Verwendung dieser Axiome und Transitionen, möglich ist, nachzuweisen, dass die Analyse tatsächlich eine sichere Untersuchung des Programmablaufs ist. Unter Anwendung dieser Klauseln konnten wir Programmblöcke vereinfachen. Am Ende der Vereinfachung wurde die lokale Umgebung aufgelöst und nachgewiesen, dass das betrachtete Programm unter einer abstrakten Umgebung, die wir ρ genannt haben, gültig ist.
Weiterhin haben wir die semantische Korrektheit nachgewiesen, indem wir die Korrektheitsrelation R eingeführt haben. Dadurch konnten wir zeigen, dass die Gültigkeit für eine Annahme unter Anwendung der "Acceptable"-Klauseln erhalten bleibt, dass also eine einmal gültige Annahme gültig bleibt.
Schließlich haben wir die Existenz einer Lösung, speziell die einer kleinsten, unter Einführung der Moore Familie nachgewiesen.
Die Kontrollflussanalyse ist also eine gültige Analysemethode mit deren Anwendung sich das Zusammenspiel einzelner Programmblöcke überprüfen lässt.

Literaturverzeichnis

Flemming Nielson, Hanne Riis Nielson, Chris Hankin:
Principles of Program Analysis.
Springer (Corrected 2nd printing, 452 pages, ISBN 3-540-65410-0), 2005.