LMU München
Institut für Informatik

SS 2009

Haupt-/Bachelorseminar: Programmanalyse

Kontrollflussanalyse 2

Bernhard Hering

Juni 2009

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

Inhaltsverzeichnis

1 Kontrollflussanalyse

Die Kontrollflussanalyse ist eine statische Analyse, das heißt sie wird vor dem Ausführen eines Programms durchgeführt, und grenzt sich damit von den dynamischen Analysen die während eines Programmablaufs durchgeführt werden, ab. Ziel der Kontrollflussanalyse ist herauszufinden von welchen Block im Programm die Kontrolle an welchen Block übergeben wird, und welche Funktionen dabei erreichbar sind. Des weiteren überprüft die Kontrollflussanalyse welche Werte für eine Funktion möglicherweise zur Ausführung kommen.

Zur Analyse des Kontrollflusses wird ein Graph erstellt, in dem die Knoten die Codeblöcke, im folgenden mit Labels bezeichnet, darstellen. Die Kanten zwischen den Koten symbolisieren die Bedingungen (Im Buch mit Constraints bezeichnet) die vorgeben wie die Kontrolle weitergeleitet wird.

In dieser Arbeit werden wir uns nun mit dem 0-CFA Verfahren beschäftigen. Im Kapitel 3.1 des mir als Grundlage vorliegenden Buches von Nielson, Nilson und Hankin wird die abstrakte Spezifikation und die strukturelle operationale Semantik des Verfahren vorgestellt und seine Korrektheit bewiesen. In diesem Teil der Arbeit wird nun die Syntax orientiert Herangehensweise (Syntax Directed 0-CFA Analysis) sowie der Algorithmus zum Erstellen und Lösen der Bedingungen vorgestellt (Constraint Based 0-CFA Analysis).

Zunächst werden ein paar Begrifflichkeiten aus den vorhergehenden Kapitel geklärt. e bezeichnet eine gelabelte Expression, t im Gegensatz dazu einen Term ohne Labels. Des weiteren sind 4 Mengen von Relevanz. Var für die Variablen, $Const$ für die Konstanten, Op beinhaltet Binäre Operatoren und schlussendlich die Menge Lab für die Labels.

2 0-CFA Verfahren

Das kurze Programmstück $((fn\ x \Rightarrow x)(fn\ y \Rightarrow y))$ wird als laufendes Beispiel betrachtet. Dieser Code ist im der Syntax der Programmiersprache FUN, in der $fn\ x \Rightarrow e_0$ eine Funktion darstellt. Somit haben wir hier die zwei Identitätsfunktionen $id_x \mapsto (fn\ x \Rightarrow x)$ und $id_y \mapsto (fn\ y \Rightarrow y)$ wobei die id_x Funktion auf die id_y Funktion angewandt wird.

2.1 Die Analyse

Das 0-CFA Verfahren ist die einfachste Form einer Kontrollflussanalyse in der keine Kontextinformationen verarbeitet werde. Diese ist auch der Grund für die "0" im Namen. Das Ergebnis der 0-CFA Analyse ist das Tupel $(\hat{C}, \hat{\rho})$.

$\hat{C}$ ist die Menge der Labels, die die abstrakten Werte der Programmblöcke abbilden.

$\hat{\rho}$ symbolisiert die Werte an die die Variable x gebunden wird.

Genauer ausgedrückt:

$\hat{v} \in \widehat{Val}$	$=$	$\mathcal{P}(Term)$	abstrakter Wert
$\hat{\rho} \in \widehat{Env}$	$=$	$Var \to \widehat{Val}$	abstrakte Umgebung
$\hat{C} \in \widehat{Cache}$	$=$	$Lab \to \widehat{Val}$	abstrakter Cache

$\hat{v}$ ist ein abstrakter Wert der eine Funktion definiert. In unserem Beispiel wäre $fn\ x \Rightarrow e_0$ ein möglicher Wert von $\hat{v}$. In $\hat{\rho}$ liegt beispielsweise die Zuordnung $x \to fn\ y \Rightarrow y$. Das bedeutet das x den Wert $fn\ y \Rightarrow y$ annehmen kann.

Abbildung 2.1: Mögliche Werte von $(\hat{C}, \hat{\rho})$

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

Um dieses Programmstück mit dem 0-CFA Verfahren analysieren zu können, müssen die Labels definiert werden. Daraus ergibt sich folgende Struktur.

$$((fn\ x \Rightarrow x^1)^2(fn\ y \Rightarrow y^3)^4)^5$$

Zunächst gilt es $\hat{C}$ zu finden. Da uns nur der Kontrollfluss interessiert, werden auf die abstakten Werte nur Funktionsterme wie fn oder $funf$ abgebildet. Hier wird nicht das Verfahren dargestellt wie die Menge gefunden wird, sonder nur die Begründung warum diese Menge den Kontrollfluss korrekt modeliert.

Zum leichteren Verständnis ist in Tabelle 2.1 die möglichen Werte von $(\hat{C}, \hat{\rho})$ angegeben, wie es womöglich als Lösung herauskommen könnte.

Zu diesem Ergebnis gelangt man, wenn man sich überlegt welche Werte die Labels annehmen können. Label 1 ist die Lösung der id_x Funktion, in die $fn\ y \Rightarrow y$ eigesetzt wird, und somit auch die Lösung ist. Die anderen Lösungen für die Labels könne analog überlegt werden.

Dies ist allerdings nur eine Überlegung und somit wird im nächsten Kapitel die Syntax orientierte Vorgehensweise vorgestellt.

Tabelle 2.1: Syntax orientierte Spezifikation der 0-CFA Analyse

$[con]$ $(\hat{C}, \hat{\rho}) \models_s c^l \; always$

$[var]$ $(\hat{C}, \hat{\rho}) \models_s x^l$

iff $\hat{\rho}(x) \subseteq \hat{C}(l)$

$[fn]$ $(\hat{C}, \hat{\rho}) \models_s (fn\ x \Rightarrow e_0)^l$

iff $\{fn\ x \Rightarrow e_0\} \subseteq \hat{C}(l) \quad \wedge \quad (\hat{C}, \hat{\rho}) \models_s e_0$

$[fun]$ $(\hat{C}, \hat{\rho}) \models_s (fun\ f\ x \Rightarrow e_0)^l$

iff $\{fun\ f\ x \Rightarrow e_0\} \subseteq \hat{C}(l) \quad \wedge \quad (\hat{C}, \hat{\rho}) \models_s e_0 \quad \wedge \quad \{fun\ f\ x \Rightarrow e_0\} \subseteq \hat{\rho}(f)$

$[app]$ $(\hat{C}, \hat{\rho}) \models_s (t_1^{l_1}\ t_2^{l_2})^l$

iff $(\hat{C}, \hat{\rho}) \models_s t_1^{l_1} \wedge (\hat{C}, \hat{\rho}) \models_s t_2^{l_2} \quad \wedge$

$(\forall (fn\ x \Rightarrow t_0^{l_0}) \in \hat{C}(l_1) : \quad \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l))$

$(\forall (fun\ f\ x \Rightarrow t_0^{l_0}) \in \hat{C}(l_1) : \quad \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l))$

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

iff $(\hat{C}, \hat{\rho}) \models_s t_0^{l_0} \quad \wedge \quad (\hat{C}, \hat{\rho}) \models_s t_1^{l_1} \quad \wedge \quad (\hat{C}, \hat{\rho}) \models_s t_2^{l_2} \quad \wedge$

$\hat{C}(l_1) \subseteq \hat{C}(l) \quad \wedge \quad \hat{C}(l_2) \subseteq \hat{C}(l))$

$[let]$ $(\hat{C}, \hat{\rho}) \models_s (let\ t_1^{l_1}\ in\ t_2^{l_2})^l$

iff $(\hat{C}, \hat{\rho}) \models_s t_1^{l_1} \quad \wedge \quad (\hat{C}, \hat{\rho}) \models_s t_2^{l_2} \quad \wedge$

$\hat{C}(l_1) \subseteq \hat{\rho}(x) \quad \wedge \quad \hat{C}(l_2) \subseteq \hat{C}(l))$

$[op]$ $(\hat{C}, \hat{\rho}) \models_s (t_1^{l_1}\ op\ t_2^{l_2})^l$

iff $(\hat{C}, \hat{\rho}) \models_s t_1^{l_1} \quad \wedge \quad (\hat{C}, \hat{\rho}) \models_s t_2^{l_2}$

2.2 Syntax orientierte Vorgehensweise

2.2.1 Spezifikation

Wie schon erwähnt geht Nielson, Nielson und Hankin im Kapitel 3.1 auf die abstrakt Spezifikation " $\models e_*$" ein. (Nielson, Nielson & Hankin, 1999) Nun wird die etwas leichter rechenbaren Spezifikation " $\models_s e_*$" betrachtet, und anschließende ihre Richtigkeit bezüglich " $\models e_*$" gezeigt.

Tabelle 2.1 definiert die Syntax orientierte Spezifikation der 0-CFA Analyse.

Zur Erklärung:

Die $[con]$ Regel, die für den Wert einer Variablen angewandt wird, ist für eine 0-CFA Analyse nicht relevant da bei einer Kontrollflussanalyse die Werte vernachlässigt werden. Es könne daraus folgend auch $(\hat{C}, \hat{\rho}) \models_s c^l$ iff $\emptyset \subseteq \hat{C}(l)$ geschrieben werden.

In der $[var]$ Klausel wird die Verbindung zwischen der abstrakten Umgebung $\hat{\rho}$ und dem Cache symbolisiert. Das bedeutet dass der Cache an der Stelle I alle Werte enthält, die x annehmen kann.

Die Abkürzungen $[fn]$ und $[fun]$ werden bei Funktionen bzw rekursiven Funktionen aufgerufen. Hier wird lediglich die Struktur der Funktion, also die Funktion ohne dem Rumpf, in den Cache aufgenommen. Durch einen rekursiven Aufruf wird sichergestellt dass der Rumpf auch Teil des Caches ist.

In $[if]$ und $[let]$ werden die einzelnen Labels zunächst einzeln aufgerufen, zum Beispiel $(\hat{C}, \hat{\rho}) \models_s t_0^{l_0}$. Damit wird der Kontrollfluss modeliert. Bei $[if]$ ist es nicht wichtig dass l_0 Teil des übergeordneten Caches l ist, da die Auswertung von l_0 im Gegensatz zu l_1 und l_2 nicht Lösung des Konstruktes sein kann. In $[let]$ wird l_1 an $\hat{\rho}(x)$ gebunden ($\hat{C}(l_1) \subseteq \hat{\rho}(x)$). Die Bedingung $\hat{C}(l_2) \subseteq \hat{C}(l))$ modelliert die Rückgabe des Resultats.

Etwas schwieriger ist das Konstrukt $[app]$ zu verstehen. Hier werden die Unterlabels von l, also l_1 und l_2 auch rekursiv aufgerufen ($(\hat{C}, \hat{\rho}) \models_s t_1^{l_1} \wedge (\hat{C}, \hat{\rho}) \models_s t_2^{l_2}$). Für alle Therme t der form $fnx \Rightarrow t_0^{l_0}$ wird mit dem Konstrukt $\hat{C}(l_2) \subseteq \hat{\rho}(x)$, $\hat{C}(l_2)$ an x gebunden, und durch $\hat{C}(l_0) \subseteq \hat{C}(l))$ die Parameterrückgabe modelliert.

2.2.2 Beispiel

Nun wird diese Spezifikation auf das Beispiel $((fn\ x \Rightarrow x^1)^2(fn\ y \Rightarrow y^3)^4)^5$ angewandt.

Wir lösen dieses Programmstück syntax-orientiert das heißt gesteuert durch den äussersten Konstruktur des Terms, und beginnen mit dem Label 5. Zum Lösen benutzen wir das Konstrukt $[app]$. Daraus ergibt sich zunächst $(\hat{C}, \hat{\rho}) \models_s (fn\, x \Rightarrow$

Abbildung 2.2: Menge der Bedingungen

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

$x^1)^2$ und $(\hat{C}, \hat{\rho}) \models_s (fn\ y \Rightarrow y^3)^4$, das heißt wir müssen uns rekursiv um diese Terme kümmern. Es muss aber auch noch geklärt werden an wen die Kontrolle übergeben wird und welcher Wert mitgegeben wird. $\hat{C}(4) \subseteq \hat{\rho}(x)$ bedeutet, dass das Ergebnis aus $\hat{C}(4)$, an die Variable x gebunden wird. Letzendlich wird die Lösung des Ausdrucks in Label 1 stehen. Daraus folgt die Bedingung $\hat{C}(1) \subseteq \hat{C}(5)$ die die Parameterrückgabe modelliert.

Wir benutzen den Fall $[fn]$ um die Labels 2 und 4 zu lösen. Daraus ergibt sich $fn\ x \Rightarrow x^1 \subseteq \hat{C}(2)$ und $fn\ y \Rightarrow y^3 \subseteq \hat{C}(4)$, wenn Label 1 beziehungsweise Label 3 die Seitenbedingungen $(\hat{C}, \hat{\rho}) \models_s x^1 \wedge (\hat{C}, \hat{\rho}) \models_s y^3$ erfüllen.

Deshalb betrachten wir anschließend Label 1 und 3. Laut Spezifikation $[var]$ müssen wir sicherstellen dass $\hat{\rho}(x) \subseteq \hat{C}(1)$ und $\hat{\rho}(y) \subseteq \hat{C}(3)$ gilt. $\hat{\rho}(x)$ ist der Wert von x also $fn\ y \Rightarrow y^3$. Das ist anschaulich gesehen der Wert der in die Funktion id_x eingesetzt wird. Da dies darausfolgend ein möglicher Wert ist, sehen wir $fn\ y \Rightarrow y^3 \subseteq \hat{C}(1)$. Analog sehen wir dass $\hat{\rho}(y)$ keinen Wert annimmt, da die Funktion id_y nie aufgerufen wird. Deshalb setzen wir für $\hat{C}(3)$ die leere Menge $\emptyset$ bzw allgemein id_y.

Daraus folgt für die Menge an Bedingungen Abbildung 2.2:

2.2.3 Korrektheit der Analyse

Die Werte die aus der oben vorgestellte Spezifikation resultieren, könnten eine unendlich große Menge werden. Dies ist aber gar nicht notwendig, da die An-

zahl der Labels des ursprünglichen Ausdrucks endlich ist. Deshalb kann man diese Spezifikation mit der aus Kapitel 3.1 (Nielson et al., 1999) vergleichen. Definieren wir mit $Lab_* \subseteq Lab$ die endliche Menge der Labels die in einem Programm e_* vorhanden sind. $Var_* \subseteq Var$ ist die endliche Menge der Variablen in e_*. Der $Term_*$ beschreibt die Menge der Unterausdrücke die in e_* auftreten.

Damit definieren wir $(\hat{C}_*^T, \hat{\rho}_*^T)$:

$$\hat{C}_*^T(l) = \begin{cases} \emptyset & \text{if} \quad l \notin Lab_* \\ Term_* & \text{if} \quad x \in Lab_* \end{cases} .$$

$$\hat{\rho}_*^T(l) = \begin{cases} \emptyset & \text{if} \quad l \notin Var_* \\ Term_* & \text{if} \quad x \in Var_* \end{cases} .$$

Die Bedingung $(\hat{C}, \hat{\rho}) \sqsubseteq (\hat{C_*^T}, \hat{\rho_*^T})$ drückt aus, dass in $(\hat{C}, \hat{\rho})$ nur Ausdrücke aus e_* auftreten.

Dies kann folgendermaßen umformulieren werden : $(\hat{C}, \hat{\rho}) \in \widehat{Cache}_* \times \widehat{Env}_*$ wobei $\widehat{Cache}_* = Lab_* \to \widehat{Val}_*$, $\widehat{Env}_* = Var_* \to \widehat{Val}_*$ und $\widehat{Val}_* = \rho(Term_*)$ ist.

Daraus können wir folgern dass alle Lösungen zu $\models_s e_*$ die "weniger" als $(\hat{C_*^T}, \hat{\rho_*^T})$ sind, auch Lösungen zu $\models e_*$ sind.

Zusammengefasst ergibt dies.

Satz 1

$$wenn \quad (\hat{C}, \hat{\rho}) \models_s e_* \quad und \quad (\hat{C}, \hat{\rho}) \sqsubseteq (\hat{C_*^T}, \hat{\rho_*^T}) \quad dann \quad (\hat{C}, \hat{\rho}) \models e_*$$

Der Beweis hierzu, bedient sich der Coinduktion. Die beiden Konstrukte $(\hat{C}, \hat{\rho}) \models_s e_* \quad und \quad (\hat{C}, \hat{\rho}) \sqsubseteq (\hat{C_*^T}, \hat{\rho_*^T})$ werden vorausgesetzt. Damit muss nur noch bewiesen werden dass jede Lösung aus $\hat{C}, \hat{\rho}) \models_s e_*$ auch eine Lösung von $\hat{C}, \hat{\rho}) \models e_*$ ist. Dazu werden alle rechten Seiten der Konstrukte aus der Spezifikation verglichen. Diese sind alle identisch bis auch $[app]$. Da aber $(\hat{C}, \hat{\rho})$ von oben durch $(\hat{C_*^T}, \hat{\rho_*^T})$ beschränkt ist, kann der Unterschied vernachlässigt werden.

2.3 Algorithmus

2.3.1 Algorithmus für die Erstellung der Bedingungen

Die eben vorgestellte Spezifikation lässt sich nicht lauffähig Programmieren, deshalb wird im folgenden auf den Algorithmus $C_*[\![e_*]\!]$ eingegangen. Als Eingabe wird ein Term e_* erwartet. Dieser Term wird zu einer Menge an Bedingungen verarbeitet.

Die Bedingungen haben entweder die Form $lhs \subseteq rhs$ oder $(\{t\} \subseteq rhs' \Rightarrow lhs) \subseteq rhs$ oder $\{t\}$. Wobei rhs hier entweder von der Form $C(l)$ oder $r(x)$ ist und lhs von der Form $C(l), r(x)$. Alle Ausprägungen von t sind entweder eine iterative Funktion $fn\ x \Rightarrow e_0$ oder eine rekursive $fn\ x \Rightarrow e_0$.

Ziel ist es die Bedingungen aus $(\hat{C}, \hat{\rho}) \models_s e_*$ in ein endliche Menge $C_*[\![e_*]\!]$ zu schreiben. Dies geschieht dadurch, dass man alle Vorkommnisse von $\hat{C}$ in C und $\hat{\rho}$ in r ändert. Der Unterschied zwischen diesen Mengen liegt darin, dass $\hat{C}(l)$ eine Menge von Ausdrücken beinhaltet wohingegen in $C(l)$ lediglich die reine Syntax festgehalten ist. Analog bei $\hat{\rho}(x)$ und $r(x)$.

Die Funktion C_* der Constraint based 0-CFA Analyse ist in Tabelle 2.2 definiert.

Alle Konstrukte sind identisch zu denen in Tabelle 2.1, bis auf $[app]$. Hier werden jetzt nicht nur diejenigen Therme t betrachtet die in l1 sind sonder alle Therme.

Beispiel

Die Erklärung sowie das Beispiel generieren sich bis auf das Konstrukt $[app]$ analog zu Kapitel 2.2. Das Konstrukt $[app]$ aus Tabelle 2.2 werden alle Terme t der Form $fn\ x \Rightarrow t_0^{l_0}$ bzw. $fun\ f\ x \Rightarrow t_0^{l_0}$ betrachtet. In der Syntax orientierten Spezifikation (2.1) darf t nur aus Label 1 sein $((\forall(fn\ x \Rightarrow t_0^{l_0}) \in \hat{C}(l_1) : \quad \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l))$ bzw $(\forall(fun\ f\ x \Rightarrow t_0^{l_0}) \in \hat{C}(l_1) : \quad \hat{C}(l_2) \subseteq \hat{\rho}(x) \wedge \hat{C}(l_0) \subseteq \hat{C}(l)))$. Deshalb müssen die Bedingungen $(fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(4) \subseteq r(y)$ und $(fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(3) \subseteq C(5)$ der Menge zusätzlich hinzugefügt werden.

Tabelle 2.2: Constraint based 0-CFA Analyse

$[con] \quad C_*[\![c^l]\!] = \emptyset$

$[var] \quad C_*[\![c^l]\!] = \{r(x) \subseteq C(l)\}$

$[fn] \quad C_*[\![(fn\ x \Rightarrow e_0)^l]\!] = \{\{fn\ x \Rightarrow e_0\} \subseteq C(l)\} \quad \cup \quad C_*[\![e_0]\!]$

$[fun] \quad (C_*[\![(fun\ x \Rightarrow e_0)^l]\!] = \{\{fun\ f\ x \Rightarrow e_0\} \subseteq C(l)\}$

$\cup \quad C_*[\![e_0]\!] \quad \cup \quad \{\{fun\ f\ x \Rightarrow e_0\} \subseteq r(f)\}$

$[app] \quad C_*[\![(t_2^{l_2}\ t_2^{l_2})^l]\!] = C_*[\![t_1^{l_1}]\!] \quad \cup \quad C_*[\![t_2^{l_2}]\!]$

$\cup \quad \{t\} \subseteq C(l_1) \Rightarrow C(l_2) \subseteq r(x) \mid t = (fn\ x \Rightarrow t_0^{l_0}) \in Term_*$

$\cup \quad \{t\} \subseteq C(l_1) \Rightarrow C(l_0) \subseteq C(l) \mid t = (fn\ x \Rightarrow t_0^{l_0}) \in Term_*$

$\cup \quad \{t\} \subseteq C(l_1) \Rightarrow C(l_2) \subseteq r(x) \mid t = (fun\ f\ x \Rightarrow t_0^{l_0}) \in Term_*$

$\cup \quad \{t\} \subseteq C(l_1) \Rightarrow C(l_0) \subseteq C(l) \mid t = (fun\ f\ x \Rightarrow t_0^{l_0}) \in Term_*$

$[if] \quad C_*[\![if\ t_0^{l_0}\ then\ t_1^{l_1}\ else\ t_2^{l_2})^l]\!] = C_*[\![t_0^{l_0}]\!] \quad \cup \quad C_*[\![t_1^{l_1}]\!] \quad \cup \quad C_*[\![t_2^{l_2}]\!] \quad \cup$

$\{C(l_1) \subseteq C(l)\} \quad \cup \quad \{C(l_2) \subseteq C(l)\}$

$[let] \quad C_*[\![(let\ x = t_1^{l_1}\ in\ t_2^{l_2})^l]\!] = C_*[\![t_1^{l_1}]\!] \quad \cup \quad C_*[\![t_2^{l_2}]\!] \quad \cup$

$\{\{C(l_1) \subseteq r(x)\} \quad \cup \quad \{C(l_2) \subseteq C(l)\}\}$

$[op] \quad C_*[\![(t_1^{l_1}\ op\ t_2^{l_2})^l]\!] = C_*[\![t_1^{l_1}]\!] \quad \cup \quad C_*[\![t_2^{l_2}]\!]$

Abbildung 2.3: Menge der Bedingungen

$C_*[\![((fn\ x \Rightarrow x^1)^2(fn\ y \Rightarrow y^3)^4)^5]\!] =$
$\{r(x) \subseteq C(1),$
$\{fn\ x \Rightarrow x^1\} \subseteq C(2),$
$r(y) \subseteq C(3)\ ,$
$\{fn\ y \Rightarrow y^3\} \subseteq C(4),$
$(fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(4) \subseteq r(x),$
$(fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(1) \subseteq C(5),$
$(fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(4) \subseteq r(y),$
$(fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(3) \subseteq C(5)\}$

Damit ergibt sich die Menge in Abbildung 2.3 für die Bedingungen.

Korrektheit des Algorithmus

Hierbei muss man sich Fragen ob diese Algorithmus die Spezifikation 2.1 errechnet.

Der Satz

Satz 2

$$wenn\ (\hat{C}, \hat{\rho}) \sqsubseteq (\hat{C}_*^T, \hat{\rho}_*^T)\ dann\ \hat{C}, \hat{\rho} \models_s e_*\ genau\ dann\ wenn\ \hat{C}, \hat{\rho} \models_c C_*[\![e_*]\!]$$

beschreibt das dies genau dann der Fall ist wenn die obere Schranke aus Kapitel 2.2.3 gilt und der Spezifikation $\models_s$ gegeben ist.

Dies lässt sich durch Strukturelle Induktion über die Expression e zeigen.

2.3.2 Algorithmus zum Lösen der Bedingungen

Es gibt nun zwei verschiedene Möglichkeiten aus diesen Bedingungen die kleinst mögliche Lösung zu finden. Die erste Möglichkeit ist einen Fixpunkt der Funktion $C_*[\![e_*]\!]$ zu finden. Dies ist in der Komplexität $O(n^5)$ möglich wenn n die Größe des Ausdruckes e_*4 ist.

Besser ist jedoch eine Graphen auszugeben, da dies in $O(n^3)$ möglich ist.

Formulieren des Graphs

Tabelle 2.3 zeigt den Algorithmus zum Lösen der Bedingungen. Er bekommt als Eingabe die Menge der Bedingungen $C_*[\![e_*]\!]$ und gibt als die Lösung $(\hat{C}, \hat{\rho})$ aus. Der Algorithmus arbeitet auf drei Datensätzen. Der Arbeitsliste W, die eine Liste der noch zu bearbeitenden Knoten hält, dem Datenfeld D das für jeden Knoten ein Element vom Typ $\widehat{Val}_*$ hält und die Kantenmenge E.

Die beste Möglichkeit die kleinstmögliche Lösung zu finden ist eine Graph zu modellieren. Der Knoten des Graphs sind alle $C(l)$ und $r(x)$. Die Kanten symbolisieren die Bedingungen aus $C_*[\![e_*]\!]$. Eine Bedingung der Form $p_1 \subseteq p_2$ erzeugt eine Kante von p_1 zu p_2, Bedingungen der Form $\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$ Kanten von p_1 zu p_2 und mit der Bedingung $\{t\} \subseteq p$. Zu jedem Knoten existiert ein Datenfeld $D[p]$, das folgendermaßen initialisiert wird. $D[p] = \{t \mid (\{t\} \subseteq p =\in C_*[\![e_*]\!]\}$. Das bedeutet das für alle Bedingungen der Form $\{t\} \subseteq p$, $\{t\}$ in das Datenfeld $D[p]$ des Knoten p eingetragen wird.

Im ersten Schritt wird die Datenhaltung initialisiert, der zweite Schritt baut den Graph auf und schreibt die Startwerte in das Datenfeld. Dies geschieht in dem Unterprogrammaufruf add(q, d), der d in D$[q]$ schreibt und q in die Arbeitslist W aufnimmt wenn d nicht schon Teil von D$[q]$ war.

Im dritten Schritt wird die Arbeitsliste abgearbeitet. Dabei wird der erste Eintrag der Arbeitsliste betrachtet und ausgewertet. Die Belegungen in D$[q]$ werden Schrittweise entlang der Kanten (Einträge in E) ausgeweitet. Der letzte Schritt sammelt die Belegungen des Graphs auf, und dokumentiert sie in der Menge $(\hat{C}, \hat{\rho})$

Beispiel

Im Kapitel 2.3.2 haben wir die Menge der Bedingungen erstellt. Im Schritt 1 werden die Datenstrukturen erstellt. Schritt 2 erstellt die Konten und die Arbeitsliste. Es ergibt sich für jedes $C(l)$ und jedes $r(x)$ ein Konten.

Tabelle 2.3: Algorithmus zum Lösen der Bedingungen

Schritt 1:	Inizialisieren	
	W := nil;	
	for q in Nodes do $D[q] := \emptyset$;	
	for q in Nodes do $E[q] := nil$;	
Schritt 2:	Erstellen des Graphs	
	for cc in $C_*[\![e_*]\!]$ do	
	case cc of	
	$\{t\} \subseteq p$:	add$(p, \{t\})$;
	$p_1 \subseteq p_2$:	$E[p_1] := cons(cc, E[p_1])$;
	$\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$:	$E[p_1] := cons(cc, E[p_1])$;
		$E[p] := cons(cc, E[p])$;
Schritt 3:	Iteration	
	while W $\neq$ nil do	
	$q := haed(W)$; $W := tail(W)$;	
	for cc in $E[q]$ do	
	case cc of	
	$p_1 \subseteq p_2$:	add$(p_2, D[p_1])$;
	$\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$:	if $t \in D[p]$ then add $(p_2, D[p_1])$;
Schritt 4:	Aufzeichnen der Lösungen	
	for l in Lab_* do $\hat{C}(l) := D[C(l)]$;	
	for x in Var_* do $\hat{\rho}(x) := D[r(x)]$;	
Unterprogramme:	procedure add(q, d) is	
	iff $\neg(d \subseteq D[q])$	
	then $D[q] := D[q] \cup d$; $W :=$cons(q, W);	

Abbildung 2.4: Initialisierung der Datenstrukturen

p	$D[p]$	$E[p]$
C(1)	$\emptyset$	$[(fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(1) \subseteq C(5)]$
C(2)	id_x	$[\ (fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(4) \subseteq r(x),\ (fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(1) \subseteq C(5),$ $(fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(4) \subseteq r(y),\ (fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(3) \subseteq C(5)]$
C(3)	$\emptyset$	$[\ (fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(3) \subseteq C(5)]$
C(4)	id_y	$[\ (fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(4) \subseteq r(x),\ (fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(4) \subseteq r(y)]$
C(5)	$\emptyset$	[]
r(x)	$\emptyset$	$[r(x) \subseteq C(1)]$
r(y)	$\emptyset$	$[r(y) \subseteq C(3)]$

In der Arbeitsliste stehen nach Schritt 2, C(4) und C(2), da in ihnen ein Wert von Typ t steht. ($\{t\} \subseteq p$; add $(p, \{t\})$;

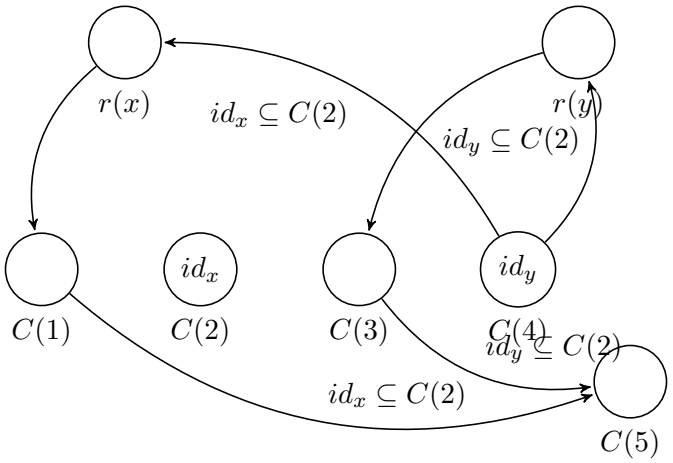
W = $[C(4), C(2)]$

In Schritt 2 werden zusätzlich Werte in $D[p]$ und $E[p]$ geschrieben. Ist cc, also die Bedingung, vom Typ ($\{t\} \subseteq p$; wird t in dem Unterprogrammaufruf add $(p, \{t\})$; in $D[p]$ aufgenommen. Ist cc in der Form $p_1 \subseteq p_2$: oder $\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$: werden die Bedingungen in E[p] eingetragen. Beispielsweiße wird für C(1) die Bedingung $(fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(1) \subseteq C(5)$ da sie in der Form $\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$: ist, in der Programmzeile $E[p_1] := cons(cc, E[p_1])$; in $E[p]$ eingetragen.

Daraus ergibt sich nach der Initialisierung folgende Abbildung 2.4 für p $D[p]$ und $E[p]$. Wie diese Daten (Abbildung 2.4) als Graph interpretiert werden, zeigt Abbildung 2.5.

Damit ist Schritt 2 abgeschlossen.

Abbildung 2.5: Graph nach Initialisierung



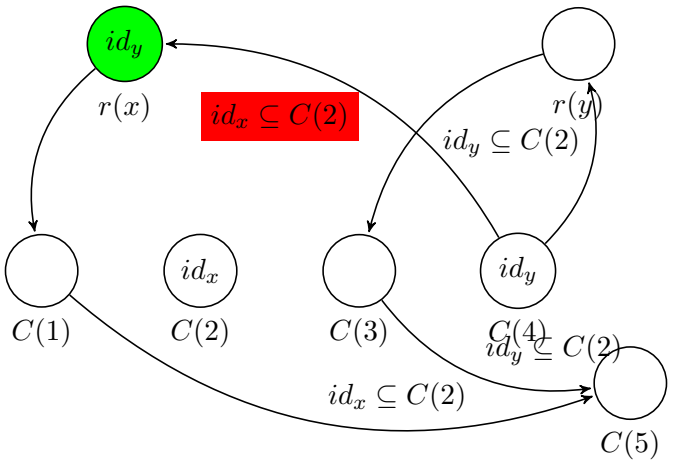
Schritt 3 arbeitet die Arbeitsliste ab. Wir beginnen mit C(4). Es müssen die beiden [$(fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(4) \subseteq r(x)$, $(fn\ y \Rightarrow y^3) \subseteq C(2) \Rightarrow C(4) \subseteq r(y)$] gelöst werden. Anschaulich gesprochen muss id_y an dem Graphen entlang weiter gereicht werden. Beide Bedingungen sind in der Form $\{t\} \subseteq p \Rightarrow p_1 \subseteq p_2$. Im Algorithmus sehen wir, dass für diese Bedingungen $(fn\ x \Rightarrow x^1) \subseteq C(2) \Rightarrow C(4) \subseteq r(x)$ geprüft ob $(fn\ x \Rightarrow x^1) \in D[C(2)]$ ist. Dies ist bei der ersten Bedingung der Fall, deshalb wird der Unterprogrammaufruf add(q,d) mit den Werten q=r(x), d=id_y aufgerufen. Zu D[r(x)] wird id_y hinzugefügt, und in die Arbeitslist wird r(x) eingefügt. Induktiv wird id_y an r(x) weitergereicht. Daraus ergibt sich die neue Arbeitsliste W = [r(x),C(2)]. Da die Bedingung $t \in D[p]$ für die zweite Bedingung nicht erfüllt ist, müssen wir sie nicht weiter betrachten.

Induktiv gesprochen geben wir das id_y aus C(4) an r(x) weiter, da die Bedingung auf der Kante C(4),r(x) im Gegensatz zur Bedingung auf C(4),r(y), erfüllt ist.

Daraus ergibt sich ein neuer Graph 2.6

Als nächstes wird r(x) abgearbeitet. Hiervon geht nur eine Kante aus, die nicht bedingt ist. Deshalb wird id_y an C(1) weitergeleitet, und C(1) in die Arbeitsliste aufgenommen. Der Graph nach diesem Schritt ist in Abbildung 2.7 ersichtlich. Da wir C(1) in die Worklist eigetragen haben, arbeiten wir sie in diesem Schritt

Abbildung 2.6: Graph nach Abarbeitung von C(4)



ab. Von C(1) geht eine allerdings bedingte Kante aus. Die Bedingung $id_x \subseteq C(2)$ ist jedoch erfüllt, weshalb wir id_y an C(5) weiterleiten dürfen und sich der Graph aus Abbildung 2.8 ergibt.

Die Abarbeitung von C(5) und C(2) muss nicht weiter betrachtet werden weil die Knoten keine ausgehenden Kanten besitzen.

Daraus folgt nach der Abarbeitung zusammenfassend Tabelle 2.9.

Der Graph im Endzustand ist in Abbildung 2.10 zu sehen

Korrektheit des Algorithmus

Der Algorithmus terminiert da er als Eingabe eine endliche Menge als Eingabe erhält. Schwieriger ist zu Zeigen dass der Algorithmus auch wirklich die kleinste Lösung $(\hat{C}, \hat{\rho})$ berechnet.

Satz 3

$$(\hat{C}, \hat{\rho}) = \sqcap\{(\hat{C`}, \hat{\rho`}) \mid (\hat{C`}, \hat{\rho`}) \models_c C_*[\![e_*]\!]\}$$

Diese Gleichung beschreibt einfach gesagt die Tatsache, das der Algorithmus mit der kleinsten Belegung beginnt, und diese Schrittweise erweitert. Damit ist gewährleistet, dass der Algorithmus die kleinste Lösung berechnet.

Abbildung 2.7: Graph nach Abarbeitung von r(x)

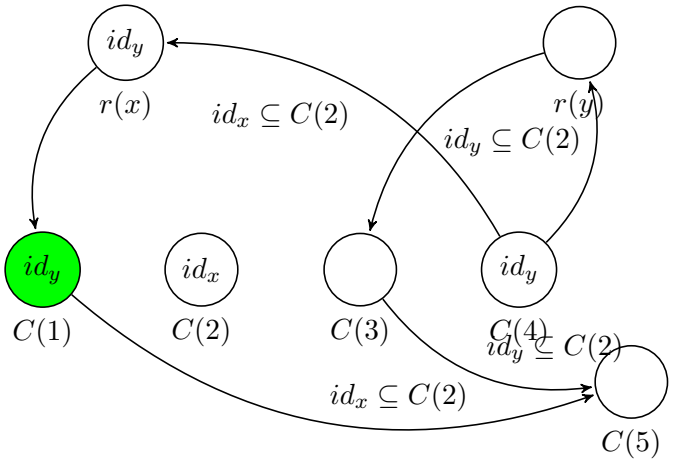


Abbildung 2.8: Graph nach Abarbeitung von r(x)

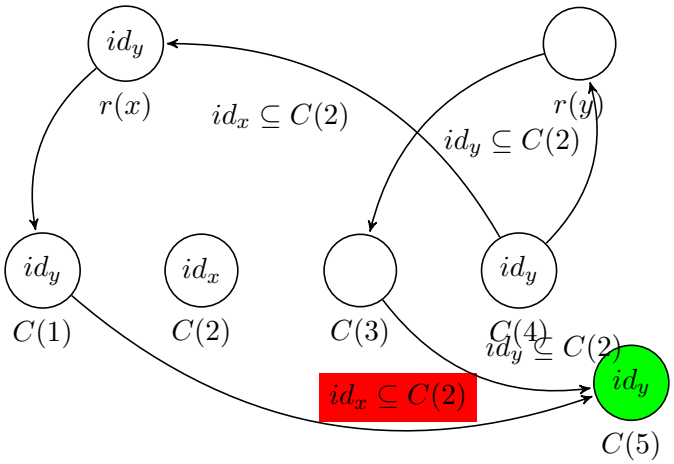
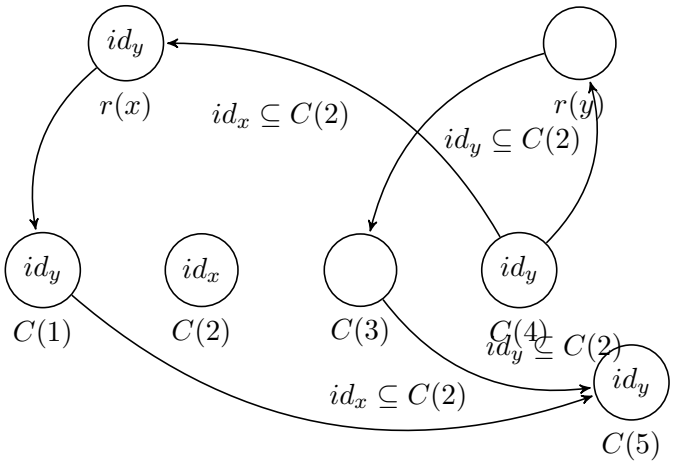


Abbildung 2.9: Abarbeitung

W	[C(4),C(2)]	[r(x),C(2)]	[C(1),C(2)]	[C(5),C(2)]	[C(2)]	[ ]
p	$D[p]$	$D[p]$	$D[p]$	$D[p]$	$D[p]$	$D[p]$
C(1)	$\emptyset$	$\emptyset$	id_y	id_y	id_y	id_y
C(2)	id_x	id_x	id_x	id_x	id_x	id_x
C(3)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$
C(4)	id_y	id_y	id_y	id_y	id_y	id_y
C(5)	$\emptyset$	$\emptyset$	$\emptyset$	id_y	id_y	id_y
r(x)	$\emptyset$	id_y	id_y	id_y	id_y	id_y
r(y)	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Abbildung 2.10: Graph nach Abarbeitung der Arbeitsliste



2.4 Zusammenfassung

Es muss noch abschließend gezeigt werden, dass der Algorithmus $C * [\![e_*]\!]$ eine Lösung der abstrakten Spezifikation des 0-CFA Algorithmus erzeugt. In Satz 3 wurde gezeigt dass der Algorithmus die kleinste Lösung aus den Bedingungen die der Algorithmus in 2.3 ausgibt, berechnet. Dies sind nach Satz 2 die gleichen Bedingungen wie diejenigen aus der Syntax orientierten Analyse, welche auch eine Lösung der abstrakten Spezifikation sind (Satz 1).

Daraus ergibt sich folgendes Schaubild dass dies nochmals verdeutlicht.

$$\Downarrow \quad (Satz\ 3)$$

$$(\hat{C}, \hat{\rho}) \models_c C[\![e_0]\!]$$

$$\Downarrow \quad (Satz\ 2)$$

$$(\hat{C}, \hat{\rho}) \models_s e_*$$

$$\Downarrow \quad (Satz\ 1)$$

$$(\hat{C}, \hat{\rho}) \models e_*$$

Der Algorithmus berechnet die kleinste Lösung $(\hat{C}, \hat{\rho})$ aus $C[\![e_0]\!]$ $(\hat{C}, \hat{\rho}) \models_c C[\![e_0]\!]$

Literaturverzeichnis

Nielson, F., Nielson, H. R. & Hankin, C. L. (1999). *Principles of program analysis*. Springer. (Second printing, 2005)