A Modular, Extensible Proof Method for Small-step Flow Analyses

Mitchell Wand and Galen B. Williamson

College of Computer Science
Northeastern University
Boston, MA 02115, USA
{wand,gwilliam}@ccs.neu.edu

Abstract. We introduce a new proof technique for showing the correctness of 0CFA-like analyses with respect to small-step semantics. We illustrate the technique by proving the correctness of 0CFA for the pure λ-calculus under arbitrary β-reduction. This result was claimed by Palsberg in 1995; unfortunately, his proof was flawed. We provide a correct proof of this result, using a simpler and more general proof method. We illustrate the extensibility of the new method by showing the correctness of an analysis for the Abadi-Cardelli object calculus under small-step semantics.

1 Introduction

Palsberg [8] was the first to consider the correctness of 0CFA analysis [11] with respect to arbitrary β-reduction. Sestoft [9, 10] had shown the correctness of 0CFA with respect to call-by-value and call-by-name evaluation, using an evaluation semantics for the former and the Krivine machine for the latter. Palsberg attempted to show that 0CFA was correct with respect to small-step semantics under arbitrary β-reduction; unfortunately, his proof was flawed. Our attempts to extend Palsberg's proof to more complex languages led us to discover flaws in the proof of one of the main theorems upon which his correctness result depends.

In this paper, we fix Palsberg's proof, working out some key details omitted from his paper and introducing a new proof technique that we believe will be easier to extend to more complex languages. Our proof is based on the observation that reduction carries most of the local structure of the source expression into the result expression, modifying only a few key terms. We illustrate this extensibility by showing the correctness of an analysis for the Abadi-Cardelli object calculus [1].

We begin in Sect. 2 by presenting the syntax and semantics of the language we will analyze, the λ-calculus, along with a few syntactic annotations and their properties. In Sect. 3, we present our control flow analysis, following Palsberg's constraint-generation system. In Sect. 4, we present our reformulation of Palsberg's proof of correctness, along with a description of precisely where his proof goes wrong. In Sect. 5, we demonstrate the extensibility of our proof technique by applying it to an analysis of the Abadi-Cardelli object calculus.

2 The Language

The syntax of our language is shown in Fig. 1. It is the standard untyped lambda calculus consisting of variable, abstraction, and application expressions. However, our analysis will require that expressions carry labels, so we define the set of expression labels *Lab*, and we use l to range over this set. We define two restrictions of *Lab*: Lab^{b} to the set of binding labels, and Lab^{λ} to the set of abstraction labels. We use β to range over Lab^{b}. Binding labels appear only on variable expressions, and on the bound variables in lambda expressions. (We have adapted the use of binding labels from [4].)

$$
\begin{array}{rcll}
l & \in & Lab & \text{Labels} \\
\beta & \in & Lab^{\mathsf{b}} & \text{Binding Labels} \\
x & \in & Var & \text{Variables} \\
e & ::= & x^{\beta} \;\mid\; (e_1\ e_2)^l \;\mid\; \lambda^l x^{\beta}.e_0 & \text{Expressions}
\end{array}
$$

Fig. 1. Syntax

In Fig. 2 we show the definitions of change of variable, $e\{y/x\}$, and substitution, $e[e'/x]$, to make explicit these two operations' effects (or lack thereof) on expression labels. Our semantics is unrestricted β-reduction, and we define the redexes and reduction contexts in Fig. 3.

$$
\begin{array}{rcl@{\qquad}rcl}
x^{\beta}\{y/x\} & = & y^{\beta} & x^{\beta}[e/x] & = & e \\
z^{\beta}\{y/x\} & = & z^{\beta} & y^{\beta}[e/x] & = & y^{\beta} \\
(e_1\ e_2)^l\{y/x\} & = & (e_1\{y/x\}\ e_2\{y/x\})^l & (e_1\ e_2)^l[e/x] & = & (e_1[e/x]\ e_2[e/x])^l \\
(\lambda^l x^{\beta}.e_0)\{y/x\} & = & (\lambda^l x^{\beta}.e_0) & (\lambda^l x^{\beta}.e_0)[e/x] & = & (\lambda^l x^{\beta}.e_0) \\
(\lambda^l z^{\beta}.e_0)\{y/x\} & = & \lambda^l z^{\beta}.(e_0\{y/x\}) & (\lambda^l y^{\beta}.e_0)[e/x] & = & (\lambda^l z^{\beta}.e_0\{z/y\}[e/x]) \\
 & & (z \neq y) & & & (z \text{ fresh})
\end{array}
$$

Fig. 2. Change of variables and substitution

$$
((\lambda^{l_0} x^{\beta}.e_0)\ e)^l \rightarrow e_0[e/x]
$$
$$
E ::= [\,] \mid \lambda^l x^{\beta}.E \mid (E\ e)^l \mid (e\ E)^l
$$

Fig. 3. Semantics: the redex and reduction contexts

Our analysis will require us to consider variable expressions and their binding abstractions in isolation from the abstractions themselves. Therefore, we introduce a syntactic notion of *well-labeledness* for expressions, along with *binding* *environments*. Intuitively, an expression is well-labelled when all of the labels on variable expressions match the binding labels appearing on the lambda expressions that bind them.

A binding environment Γ is a finite map from variables to the labels of their respective binders:

$$\Gamma : \mathit{Var} \to \mathit{Lab}^{\mathsf{b}} \ .$$

Consider the expression $\lambda^l x^\beta.e_0$. A binding environment Γ would be "correct" for the expression e_0 only if $\Gamma(x) = \beta$. We formalize this notion of a binding environment being correct for an expression into *labelling judgements* of the form $\Gamma \vdash e$ wl, which says simply that the expression e is well-labelled under the binding environment Γ. Well-labelledness for expressions is captured precisely by the set of rules for deriving labelling judgements shown in Fig. 4. For convenience, we say that an expression e is well-labelled, written $\vdash e$ wl, iff there exists a labelling environment Γ such that $\Gamma \vdash e$ wl.

$$\frac{\Gamma(x) = \beta \quad \beta \in \mathit{Lab}^{\mathsf{b}} \quad \beta = \beta'}{\Gamma \vdash x^{\beta'} \ \mathsf{wl}} \ [\textbf{var-wl}]$$

$$\frac{\Gamma \vdash e_1 \ \mathsf{wl} \quad \Gamma \vdash e_2 \ \mathsf{wl}}{\Gamma \vdash (e_1 \ e_2)^l \ \mathsf{wl}} \ [\textbf{app-wl}]$$

$$\frac{\Gamma[x \mapsto \beta] \vdash e_0 \ \mathsf{wl}}{\Gamma \vdash \lambda^l x^\beta.e_0 \ \mathsf{wl}} \ [\textbf{abs-wl}]$$

Fig. 4. The rules for deriving labelling judgements

Finally, we define a subterm relation $\in_{\mathsf{E}}$, shown in Fig. 5. The lemma which follows expresses some properties of well-labelledness that should be familiar from typing. (See [6], pp. 244-5.)

$$\begin{aligned} e_0 \in_{\mathsf{E}} e \ \text{ iff } \ & (e = e_0) \\ & \vee \ (e = (\lambda^l x^\beta.e_1) \wedge e_0 \in_{\mathsf{E}} e_1) \\ & \vee \ (e = (e_1 \ e_2)^{l'} \wedge (e_0 \in_{\mathsf{E}} e_1 \vee e_0 \in_{\mathsf{E}} e_2)) \end{aligned}$$

Fig. 5. The subterm relation $\in_{\mathsf{E}}$

Lemma 2.1. *Let e be an expression and let Γ be a binding environment.*

1. *If $\Gamma \vdash e$ wl and $e' \in_{\mathsf{E}} e$, then there exists a Γ' extending Γ such that $\Gamma' \vdash e'$ wl.*
2. *If $x \notin \mathsf{fv}(e)$, then for all $\beta \in \mathit{Lab}^{\mathsf{b}}$, $\Gamma \vdash e$ wl iff $\Gamma[x \mapsto \beta] \vdash e$ wl.*
3. *If $\Gamma[x \mapsto \beta] \vdash e$ wl, then $\Gamma[y \mapsto \beta] \vdash e\{y/x\}$ wl.*
4. *If $\Gamma[x \mapsto \beta] \vdash e$ wl and $\Gamma \vdash e_0$ wl, then $\Gamma \vdash e[e_0/x]$ wl.*

Proof. Each property is proved by a straightforward induction, the first by induction on the definition of $e' \in_{\mathsf{E}} e$, and the remaining three by induction on the size of e.

The following lemma shows that well-labelledness is preserved under reduction.

Lemma 2.2. *If $\Gamma \vdash e$ wl and $e \to e'$, then $\Gamma \vdash e'$ wl.*

Proof. Since $e \to e'$, choose the reduction context E and redex r such that $r \to s$, $e = E[r]$, and $e' = E[s]$. We proceed by induction on the structure of E.

In the base case, we have $E = [\,]$. Then $e = r$, $e' = s$, $\Gamma \vdash r$ wl, and we must show $\Gamma \vdash s$ wl. Since r is a redex, $r = ((\lambda^{l_0} x^{\beta}.e_0)\ e_1)^l$, so $s = e_0[e_1/x]$. Since $\Gamma \vdash r$ wl, we have $\Gamma \vdash \lambda^{l_0} x^{\beta}.e_0$ wl and $\Gamma \vdash e_1$ wl by [**app-wl**]. Thus, $\Gamma[x \mapsto \beta] \vdash e_0$ wl by [**abs-wl**]. Then by Lemma 2.1, part 4, we have $\Gamma \vdash e_0[e_1/x]$ wl and thus $\Gamma \vdash s$ wl.

In the induction step, consider first $E = \lambda^{l_0} x^{\beta}.E_0$. In this case, $e = E[r] = \lambda^{l_0} x^{\beta}.E_0[r]$, and we have

$$\begin{aligned}
\Gamma \vdash \lambda^{l_0} x^{\beta}.E_0[r] \ \mathsf{wl} &\Rightarrow \Gamma[x \mapsto \beta] \vdash E_0[r] \ \mathsf{wl} && \text{(by [}\mathbf{abs\text{-}wl}\text{])}\\
&\Rightarrow \Gamma[x \mapsto \beta] \vdash E_0[s] \ \mathsf{wl} && \text{(by IH)}\\
&\Rightarrow \Gamma \vdash \lambda^{l_0} x^{\beta}.E_0[s] \ \mathsf{wl} && \text{(by [}\mathbf{abs\text{-}wl}\text{])}
\end{aligned}$$

Consider next $E = (E_1\ e_2)^l$. Then $e = E[r] = (E_1[r]\ e_2)^l$, and we have

$$\begin{aligned}
\Gamma \vdash (E_1[r]\ e_2)^l \ \mathsf{wl} &\Rightarrow \Gamma \vdash E_1[r] \ \mathsf{wl} \wedge \Gamma \vdash e_2 \ \mathsf{wl} && \text{(by [}\mathbf{app\text{-}wl}\text{])}\\
&\Rightarrow \Gamma \vdash E_1[s] \ \mathsf{wl} && \text{(by IH)}\\
&\Rightarrow \Gamma \vdash (E_1[s]\ e_2)^l \ \mathsf{wl} && \text{(by [}\mathbf{app\text{-}wl}\text{])}
\end{aligned}$$

In the final case, $E = (e_1\ E_2)^l$, and the proof is similar to the previous case.

3 The Analysis

We derive our analysis from the constraint-based analysis of [8]. The analysis of an expression is specified by a constraint system (see Defn. 4.1) that is generated from the program text [2]. A solution to these constraints will give an approximation to the possible results of evaluating each program point.

The constraint system consists of a collection of conditional inclusions between sets. In general, there will be one set for each program point and for each bound variable. Palsberg uses two sets of metavariables $[\![\lambda^l]\!]$ and $[\![\nu^l]\!]$ as names for these sets; Nielson and Nielson [7] formulate these as abstract caches $\widehat{C}$ and $\widehat{\rho}$. Other researchers, e.g. [3, 5], convert terms to A-normal form or the like, and use variables as program points. We introduce labels for each program point and for each bound variable (following [4]). Because denoted and expressed values in our language coincide, we need only a single cache Φ.

The labelling behavior of our semantics follows Palsberg's: when one term reduces to another, the label on the source term will disappear, and the result term will keep its previous label. This means that a given label may appear in many different places as its term is copied. Our notion of well-labelledness ensures that the label on a variable instance matches the label on its binder.

We express flow information using a single map. This abstract cache maps labels to *abstract values*, which are sets of abstraction labels.

$$\begin{aligned} \Phi \in \widehat{Cache} &= Lab \rightarrow \widehat{Val} \\ \widehat{Val} &= \mathcal{P}(Lab^{\lambda}) \end{aligned}$$

An abstract value is a set of abstraction labels because abstractions are the only values in our language. If our language had scalars, they would also be abstracted into the set $\widehat{Val}$.

Define lab to be a map from expressions to their labels:

$$\begin{aligned} \mathsf{lab}(x^{\beta}) &= \beta \\ \mathsf{lab}(\lambda^{l} x^{\beta}.e_0) &= l \\ \mathsf{lab}((e_1\ e_2)^{l}) &= l \end{aligned}$$

The analysis of an expression e is the following set of constraints [8]:

$$[\![e]\!] = \bigcup_{\substack{\lambda^{l_0} x^{\beta}.e_0 \in_{\mathsf{E}} e \\ (e_1\ e_2)^{l} \in_{\mathsf{E}} e}} R((e_1\ e_2)^{l}, \lambda^{l_0} x^{\beta}.e_0) \quad \cup \bigcup_{\lambda^{l_0} x^{\beta}.e_0 \in_{\mathsf{E}} e} \{\ l_0 \in \Phi(l_0)\ \}\ ,$$

where the set $R((e_1\ e_2)^{l}, (\lambda^{l_0} x^{\beta}.e_0))$ consists of the following two constraints:

$$\begin{aligned} &\{l_0\} \subseteq \Phi(\mathsf{lab}(e_1)) \Rightarrow \Phi(\mathsf{lab}(e_2)) \subseteq \Phi(\beta) \\ &\{l_0\} \subseteq \Phi(\mathsf{lab}(e_1)) \Rightarrow \Phi(\mathsf{lab}(e_0)) \subseteq \Phi(l)\ . \end{aligned}$$

The first comprehension ensures that every abstraction term in the expression is matched against every application term in the expression. Palsberg gives a nice description of the two constraints in R:

- *The first constraint.* If the operator of [the application] evaluates to an abstraction with label l_0, then the bound variable of that abstraction may be substituted with everything to which the operand of [the application] can evaluate.
- *The second constraint.* If the operator of [the application] evaluates to an abstraction with label l_0, then everything to which the body of the abstraction evaluates is also a possible result of evaluating the whole application. [8, p. 280]

The second comprehension consists of a single constraint $l_0 \in \Phi(l_0)$ for every abstraction label appearing in the source expression. These constraints ensure that each abstraction is predicted as a possible value for itself.

4 Correctness

As in [8], our proof of correctness for the analysis is based on an entailment relation between constraint systems, $A \vdash A'$. This entailment expresses that all of the constraints in A' can be derived logically from those in A. (Definition 4.2 and Lemma 4.3 are verbatim from [8], except as noted.) While we have formulated our proof entirely in terms of constraint systems and deductions, it could easily be reformulated in terms of solutions alone.

We begin by formalizing our constraint language.

Definition 4.1. *Let V and U be sets (of* set variables *and* elements*, respectively) A* constraint system *A over V and U is a finite set of Horn clauses, defined by the following grammar:*

$$
\begin{array}{rcll}
s & \in & V & \textit{Set Variables} \\
c & \in & U & \textit{Constants} \\
I & ::= & c \in s \ \mid \ s \subseteq s & \textit{Atomic Formulas} \\
H & ::= & I \ \mid \ I \Rightarrow H & \textit{Clauses} \\
A & \in & \mathfrak{F}(H) & \textit{Constraint Systems}
\end{array}
$$

A solution *of a constraint system of V and U is a map $\Phi : V \rightarrow \mathcal{P}(U)$ such that $\Phi \models A$, where $\Phi \models A$ is defined by:*

$$
\begin{array}{ll}
\Phi \models A & \textit{iff } \forall\, H \in A, \Phi \models H \\
\Phi \models c \in s & \textit{iff } c \in \Phi(s) \\
\Phi \models s_1 \subseteq s_2 & \textit{iff } \Phi(s_1) \subseteq \Phi(s_2) \\
\Phi \models I \Rightarrow H & \textit{iff } \Phi \models I \Rightarrow \Phi \models H
\end{array}
$$

For our constraint systems, V will be *Lab* and U will be Lab^{λ}.

Definition 4.2. *If A is a constraint system, and H is a Horn clause, then the judgement $A \vdash H$ ("A entails H") holds if it is derivable using the following five rules:*

$$
\frac{}{A \vdash H} \quad \textit{if } H \in A \qquad \textit{(Discharge)}
$$

$$
\frac{}{A \vdash P \subseteq P} \qquad \textit{(Reflexivity)}
$$

$$
\frac{A \vdash P \subseteq P' \qquad A \vdash P' \subseteq P''}{A \vdash P \subseteq P''} \qquad \textit{(Transitivity)}
$$

$$
\frac{A \vdash X \qquad A \vdash X \Rightarrow Y}{A \vdash Y} \qquad \textit{(Modus Ponens)}
$$

$$
\frac{A \vdash P' \subseteq P'' \qquad A \vdash P \subseteq P'' \Rightarrow Q' \subseteq Q'' \qquad A \vdash Q \subseteq Q'}{A \vdash P \subseteq P' \Rightarrow Q \subseteq Q''} \qquad \textit{(Weakening)}
$$

If A, A' are constraint systems, then $A \vdash A'$ if and only if for every $H \in A'$, we have $A \vdash H$.

We choose to diverge in notation from [8] in that we overload the "turnstyle" operator, $\vdash$, so that the right hand side can be either a single constraint or a constraint system (set of clauses) with the conjunction of these clauses implied, as it already is for $\models$.

Lemma 4.3. *$\vdash$ is reflexive, transitive, and solution-preserving. If $A \supseteq A'$, then $A \vdash A'$.*

Proof. Trivial, relying on the definition of $\Phi \models A$ above. See Lem. 4.2 in [8].

Lemma 4.4. *If $\Gamma[x \mapsto \beta] \vdash e_0$ wl and $\Gamma \vdash e$ wl, then $\mathsf{lab}(e_0[e/x]) = \mathsf{lab}(e_0)$, unless $e_0 = x^\beta$.*

Proof. By inspection of the definition of substitution, observing that variable renaming preserves labels.

The following relation captures the possible differences between terms in the result of a reduction and the terms in the source expression that gave rise to them.

Definition 4.5. *If r is a β-redex and s is an expression such that $r \to s$, E is a reduction context, and e, e' are expressions such that $e \in_{\mathsf{E}} E[r]$ and $e' \in_{\mathsf{E}} E[s]$, then define $\mathbb{S}(e, e')$ iff either*

1. *$\mathsf{lab}(e) = \mathsf{lab}(e')$, or*
2. *$e = r$ and $e' = s$, or else*
3. *$r = ((\lambda^{l_0} x^\beta.e_0)\ e_1)^l$, $s = e_0[e_1/x]$, $e = x^\beta$ and $e' = e_1$.*

Since our analysis is a comprehension over applications and abstractions that occur in the expression to be analyzed, it will be useful to have a characterization of the applications and abstractions that occur in the result expression in terms of the applications and abstractions that occur in the source expression. The following lemma provides such a characterization, pointing out that every abstraction or application in the result expression arises from an abstraction or application in the original term with the same label and with immediate subterms that either have the same label as the subterms in the original, or else whose subterms differ in the very specific ways delineated by Definition 4.5.

Lemma 4.6. *Assume r is a redex and s an expression such that $r \to s$, and E is a reduction context such that $\vdash E[r]$ wl. Then*

1. *If $(e_1'\ e_2')^l \in_{\mathsf{E}} E[s]$, then there exists a $(e_1\ e_2)^l \in_{\mathsf{E}} E[r]$ such that $\mathbb{S}(e_1, e_1')$ and $\mathbb{S}(e_2, e_2')$.*
2. *If $\lambda^l x^\beta.e_1' \in_{\mathsf{E}} E[s]$, then there exists a $\lambda^l x^\beta.e_1 \in_{\mathsf{E}} E[r]$ such that $\mathbb{S}(e_1, e_1')$.*

Proof. Since r is a redex, it must be of the form $((\lambda^{l_0} y^\beta.e_0)\ e)^l$ and thus $s = e_0[e/y]$. We must show that for each $(e_1'\ e_2')^{l'} \in_{\mathsf{E}} E[s]$, there exists an application $(e_1\ e_2)^{l'} \in_{\mathsf{E}} E[r]$, and for each $\lambda^{l_0'} x^{\beta'}.e_3' \in_{\mathsf{E}} E[s]$, there exists an abstraction $\lambda^{l_0'} x^{\beta'}.e_3 \in_{\mathsf{E}} E[r]$, such that for $i = 1 \ldots 3$, $\mathbb{S}(e_i, e_i')$.

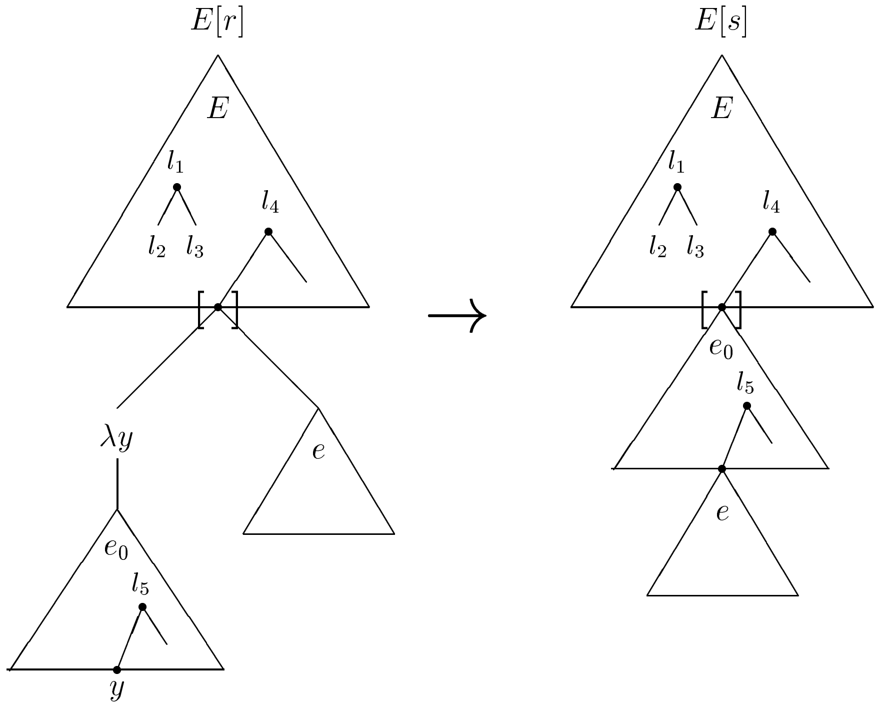

Fig. 6. The tree representations of $E[r]$ and $E[s]$

Consider every abstraction or application node in $E[s]$. Each such node in e and in the interiors of E and e_0 looks like the node labelled l_1 in Fig. 6. These nodes were unaffected by the reduction, and their children are covered by case (1) of Defn. 4.5.

The remaining nodes in $E[s]$ are like those labelled l_4 and l_5. Since there is a unique hole in E, there is exactly one node like that labelled l_4, which has the hole in E as a child. This node is covered by case (2) of Defn. 4.5, since it has s as a child after the reduction, and it had r as a child before the reduction.

This leaves nodes in $E[s]$ like that labelled l_5 in the figure. These nodes are covered by case (3) of Defn. 4.5, since they have a copy of e as a child after the reduction, as a result of substituting for an occurrence of y in the redex. However, Defn. 4.5 requires that the labels on these occurrences of y must all have been β, the label on the bound variable in the redex. This requirement is satisfied easily using the well-labelledness of $E[r]$. Since $\vdash E[r]\ \mathsf{wl}$, choose a labelling environment Γ such that $\Gamma \vdash E[r]\ \mathsf{wl}$. Thus, since $r \in_{\mathsf{E}} E[r]$, by part (1) of Lem. 2.1, there exists a Γ' extending Γ such that $\Gamma' \vdash r\ \mathsf{wl}$. Then by [**app-wl**] and [**abs-wl**], we have $\Gamma'[y \mapsto \beta] \vdash e_0\ \mathsf{wl}$. Thus, since any instance of y substituted for in the reduction must have been a free occurrence of y in e_0, by inspection of the rules for labelling judgements, we can see that the label on each such occurrence must have been $\Gamma'[y \mapsto \beta](y) = \beta$.

The next lemma states that terms in the result of a reduction have smaller flow information than the terms in the source expression to which they are related by $\$$.

Lemma 4.7. *If r is a redex and s an expression such that $r \to s$, E is a reduction context such that $\vdash E[r]$ wl, and e and e' are expressions such that $e \in_{\mathsf{E}} E[r]$, $e' \in_{\mathsf{E}} E[s]$, and $\$(e, e')$, then $[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e')) \subseteq \Phi(\mathsf{lab}(e))$.*

Proof. We note that since r is a redex and $r \to s$, we have $r = ((\lambda^{l_0} x^\beta.e_0)\ e_1)^l$ and $s = e_0[e_1/x]$. We have

$$[\![E[r]]\!] \vdash [\![r]\!] = [\![((\lambda^{l_0} x^\beta.e_0)\ e_1)^l]\!] \ ,$$

and thus,

$$\begin{aligned}[\![r]\!] \vdash \{\ & l_0 \in \Phi(l_0)\ , \\ & l_0 \in \Phi(l_0) \Rightarrow \Phi(\mathsf{lab}(e_1)) \subseteq \Phi(\beta)\ , \\ & l_0 \in \Phi(l_0) \Rightarrow \Phi(\mathsf{lab}(e_0)) \subseteq \Phi(l)\ \}\ .\end{aligned}$$

Therefore, by two applications of Modus Ponens, we have

$$[\![r]\!] \vdash \Phi(\mathsf{lab}(e_1)) \subseteq \Phi(\beta)\ , \tag{1}$$

and

$$[\![r]\!] \vdash \Phi(\mathsf{lab}(e_0)) \subseteq \Phi(l)\ . \tag{2}$$

We proceed by cases on the definition of $\$(e, e')$:

1. Trivial.
2. Since $e = r$ and $e' = s$, we have $\mathsf{lab}(e) = \mathsf{lab}(r) = l$ and $\mathsf{lab}(e') = \mathsf{lab}(s) = \mathsf{lab}(e_0[e_1/x])$. By Lem. 4.4, unless $e_0 = x$, $\mathsf{lab}(e_0[e_1/x]) = \mathsf{lab}(e_0)$, and the result follows trivially. Consider instead $e_0 = x$. Since $\vdash E[r]$ wl, choose a labelling environment Γ such that $\Gamma \vdash E[r]$ wl. Then by Lem. 2.1, there exists a Γ' extending Γ such that $\Gamma' \vdash r$ wl, and thus, we have $\Gamma'[x \mapsto \beta] \vdash e_0$ wl by [**app-wl**] and [**abs-wl**]. Now, since $\Gamma'[x \mapsto \beta] \vdash e_0$ wl, $\Gamma'[x \mapsto \beta](x) = \beta$, and $e_0 = x$, we must have $\mathsf{lab}(e_0) = \beta$ by [**var-wl**]. We also have $e_0[e_1/x] = x^\beta[e_1/x] = e_1$, and thus, $\mathsf{lab}(s) = \mathsf{lab}(e_1)$. By (1) and (2) then, we have $[\![r]\!] \vdash \Phi(\mathsf{lab}(e_1)) \subseteq \Phi(\beta) \subseteq \Phi(l)$, and thus, since $e = r$ and $e' = s$, we have $[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e')) \subseteq \Phi(\mathsf{lab}(e))$, by Transitivity of both $\vdash$ and $\subseteq$.
3. Since $e = x^\beta$ and $e' = e_1$, $\mathsf{lab}(e) = \beta$ and $\mathsf{lab}(e') = \mathsf{lab}(e_1)$, by (1) and Transitivity of $\vdash$, we have $[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e')) \subseteq \Phi(\mathsf{lab}(e))$.

Now we turn to the main theorem, which shows that constraints are preserved under reduction. That is, the constraints of the result term are entailed by the constraints of the source term. The correctness of the analysis is a consequence of this entailment: the flow information of the result term is predicted by (contained in) the flow information of the source term.

We replace the inductive structure of Palsberg's proof of the equivalent theorem ([8, Thm. 4.10]) by a flat structure that matches the flat, non-inductive structure of the definition of the analysis.

Theorem 4.8. *If $e_X \to e_Y$, and there exists a Γ_X such that $\Gamma_X \vdash e_X$ wl, then $[\![e_X]\!] \vdash [\![e_Y]\!]$ and $[\![e_X]\!] \vdash \Phi(\mathsf{lab}(e_Y)) \subseteq \Phi(\mathsf{lab}(e_X))$.*

Proof. Since $e_X \to e_Y$, choose a reduction context E, redex r, and corresponding reductum s such that $e_X = E[r] \to E[s] = e_Y$. Since r is a redex, it must be of the form $((\lambda^{l_0} y^{\beta}.e_0)\ e)^l$ and thus $s = e_0[e/y]$. We need to show $[\![E[r]]\!] \vdash [\![E[s]]\!]$, where

$$[\![E[s]]\!] = \bigcup_{\substack{\lambda^{l'_0} x^{\beta'}.e'_3 \in_{\mathsf{E}} E[s] \\ (e'_1\ e'_2)^{l'} \in_{\mathsf{E}} E[s]}} R((e'_1\ e'_2)^{l'}, \lambda^{l'_0} x^{\beta'}.e'_3) \cup \bigcup_{\lambda^{l'_0} x^{\beta'}.e'_3 \in_{\mathsf{E}} E[s]} \{\ l'_0 \in \Phi(l'_0)\ \}\ .$$

Let $(e'_1\ e'_2)^{l'} \in_{\mathsf{E}} E[s]$ and $\lambda^{l'_0} x^{\beta'}.e'_3 \in_{\mathsf{E}} E[s]$. Since $e_X = E[r]$ is well-labelled, by Lem. 4.6, there exists an application $(e_1\ e_2)^{l'} \in_{\mathsf{E}} E[r]$, and there exists an abstraction $\lambda^{l'_0} x^{\beta'}.e_3 \in_{\mathsf{E}} E[r]$, such that $\$(e_i, e'_i)$ for $i = 1 \ldots 3$. By Lem. 4.7 then, we have $[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e'_i)) \subseteq \Phi(\mathsf{lab}(e_i))$ for $i = 1 \ldots 3$.

Now, by the definition of $[\![E[r]]\!]$, since $(e_1\ e_2)^{l'} \in_{\mathsf{E}} E[r]$ and $\lambda^{l'_0} x^{\beta'}.e_3 \in_{\mathsf{E}} E[r]$, we have

$$[\![E[r]]\!] \vdash R((e_1\ e_2)^{l'}, \lambda^{l'_0} x^{\beta'}.e_3)\ .$$

Thus we have

$$[\![E[r]]\!] \vdash \{l'_0\} \subseteq \Phi(\mathsf{lab}(e_1)) \Rightarrow \Phi(\mathsf{lab}(e_2)) \subseteq \Phi(\beta')$$

and

$$[\![E[r]]\!] \vdash \{l'_0\} \subseteq \Phi(\mathsf{lab}(e_1)) \Rightarrow \Phi(\mathsf{lab}(e_3)) \subseteq \Phi(l')\ .$$

Now, since we have $[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e'_i)) \subseteq \Phi(\mathsf{lab}(e_i))$ for each e'_i and its corresponding e_i, we have

$$\frac{\begin{array}{l}[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e'_1)) \subseteq \Phi(\mathsf{lab}(e_1)) \\ [\![E[r]]\!] \vdash \{l'_0\} \subseteq \Phi(\mathsf{lab}(e_1)) \Rightarrow \Phi(\mathsf{lab}(e_2)) \subseteq \Phi(\beta') \\ [\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e'_2)) \subseteq \Phi(\mathsf{lab}(e_2))\end{array}}{[\![E[r]]\!] \vdash \{l'_0\} \subseteq \Phi(\mathsf{lab}(e'_1)) \Rightarrow \Phi(\mathsf{lab}(e'_2)) \subseteq \Phi(\beta')}$$

and

$$\frac{\begin{array}{l}[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e'_1)) \subseteq \Phi(\mathsf{lab}(e_1)) \\ [\![E[r]]\!] \vdash \{l'_0\} \subseteq \Phi(\mathsf{lab}(e_1)) \Rightarrow \Phi(\mathsf{lab}(e_3)) \subseteq \Phi(l') \\ [\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e'_3)) \subseteq \Phi(\mathsf{lab}(e_3))\end{array}}{[\![E[r]]\!] \vdash \{l'_0\} \subseteq \Phi(\mathsf{lab}(e'_1)) \Rightarrow \Phi(\mathsf{lab}(e'_3)) \subseteq \Phi(l')}$$

by Weakening in both cases. But these two constraints are exactly the constraints in $R((e'_1\ e'_2)^{l'}, \lambda^{l'_0} x^{\beta'}.e'_3)$, and so we have shown

$$[\![E[r]]\!] \vdash \bigcup_{\substack{\lambda^{l'_0} x^{\beta'}.e'_3 \in_{\mathsf{E}} E[s] \\ (e'_1\ e'_2)^{l'} \in_{\mathsf{E}} E[s]}} R((e'_1\ e'_2)^{l'}, \lambda^{l'_0} x^{\beta'}.e'_3)\ .$$

Now, since $\lambda^{l'_0} x^{\beta'}.e_0 \in_{\mathsf{E}} E[r]$, we have $[\![E[r]]\!] \vdash \{\ l'_0 \in \Phi(l'_0)\ \}$, and so we have

$$[\![E[r]]\!] \vdash \bigcup_{\lambda^{l'_0} x^{\beta'}.e'_3 \in_{\mathsf{E}} E[s]} \{\ l'_0 \in \Phi(l'_0)\ \}\ .$$

Thus we have $[\![E[r]]\!] \vdash [\![E[s]]\!]$, which is $[\![e_X]\!] \vdash [\![e_Y]\!]$.

Finally, we show $[\![e_X]\!] \vdash \Phi(\mathsf{lab}(e_Y)) \subseteq \Phi(\mathsf{lab}(e_X))$. Since $e_X = E[r]$ and $e_Y = E[s]$, obviously we have $e_X \in_{\mathsf{E}} E[r]$ and $e_Y \in_{\mathsf{E}} E[s]$. Now, if $E = [\ ]$, then $E[r] = r$ and $E[s] = s$, and we have $\$(E[r], E[s])$. If instead $E \neq [\ ]$, then $\mathsf{lab}(E[r]) = \mathsf{lab}(E[s])$, and again we have $\$(E[r], E[s])$. Thus, since $r \to s$, and $\vdash E[r]\ \mathsf{wl}$, by Lem. 4.7, we have $[\![e_X]\!] \vdash \Phi(\mathsf{lab}(E[r])) \subseteq \Phi(\mathsf{lab}(E[s]))$.

The following corollary states that if an expression converges to a value, then the label on the value is among those predicted by the analysis of the expression.

Corollary 4.9. *If* $\vdash e\ \mathsf{wl}$ *and e converges to a value v, then* $[\![e]\!] \vdash \mathsf{lab}(v) \in \Phi(\mathsf{lab}(e))$.

Note that much information is lost by this corollary: while it shows that the label of a result value is predicted, it does not tell us anything about the internal structure of this value, which may be arbitrarily complex, but is guaranteed to obey the constraints of $[\![e]\!]$.

We can now characterize the difficulty in Palsberg's theorem [8, Thm. 4.10] corresponding to our Thm. 4.8. Palsberg's proof proceeds by induction on the structure of e_X. While considering the case in which $e_X = (e_1\ e_2)^l$ and $e_1 \to e'_1$ (that is, the reduction occurs in the operator subterm), he invokes the induction hypothesis to get $[\![e_1]\!] \vdash [\![e'_1]\!]$. He then shows that for every $\lambda^{l_0} x.e'$ in $(e'_1\ e_2)^l$, $[\![(e_1\ e_2)^l]\!] \vdash R((e'_1\ e_2)^l, \lambda^{l_0} x.e')$. However, he needs to show $[\![(e_1\ e_2)^l]\!] \vdash R((e''_1\ e''_2)^l, \lambda^{l_0} x.e')$ for *every* $(e''_1\ e''_2)^l$ in $(e'_1\ e_2)^l$.

Any repair of this error would require an induction hypothesis that accounted for the context in which the reduction appeared. This approach is complicated by the mismatch between any inductive approach and the non-inductive formulation of the analysis used by Palsberg. We made several increasingly baroque attempts to repair the proof, and were eventually led to abandon a formal inductive structure in favor of the "flat" characterization of Lem. 4.6.

5 The Abadi-Cardelli Object Calculus

We presume the reader is familiar with the Abadi-Cardelli Object Calculus [1]. Our syntax is shown in Fig. 7. We have adopted a slightly non-standard syntax to facilitate the presentation. Subterms are labelled with superscripts; bound variables of comprehensions are presented as subscripts, e.g. $\langle \ldots \rangle^l_i$. We elide the range of the bound variable i, relying on the fact that this calculus neither adds nor removes methods from an object. For any program, the set of method names is finite.

$$
\begin{array}{ll}
m & \text{Method Names} \\
x & \text{Identifiers} \\
\mu ::= m =^l (b)e & \text{Methods} \\
b ::= x^\beta & \text{Variables} \\
e ::= b \ \mid \ \langle \mu_i \rangle_i^l \ \mid \ (e.m)^l \ \mid \ (e \Leftarrow (\mu))^l & \text{Expressions}
\end{array}
$$

Fig. 7. Syntax of the Object Calculus

$$
\begin{array}{ll}
(\langle m_i =^{l_i} (x_i^{\beta_i})e_i \rangle_i^{l_1}.m_j)^{l_2} & \\
\quad \rightarrow e_j[\langle m_i =^{l_i} (x_i^{\beta_i})e_i \rangle_i^{l_1}/x_j] & \text{(Red Sel)} \\
& \\
(\langle m_i =^{l_i} (x_i^{\beta_i})e_i \rangle_i^{l_1} \Leftarrow (m_j =^{l_2} (x^\beta)e))^{l_3} & \\
\quad \rightarrow \langle m_j =^{l_2} (x^\beta)e, \ m_i =^{l_i} (x_i^{\beta_i})e_i \,|_{i, \ i \neq j} \rangle^{l_3} & \text{(Red Upd)}
\end{array}
$$

Fig. 8. Reduction Rules of the Object Calculus

The reduction rules of the object calculus are shown in Fig. 8. Reductions may be carried out in arbitrary contexts, and well-labelledness works in the usual manner.

We think of objects as sets of abstractions, indexed by method name. As with the lambda-calculus, we label each object by the expression in which it was created, either by an object expression or by an update expression. Methods are similarly identified by labels.

For this analysis, the cache will have two arguments: a label and a method name, and will return a set of method labels. The intention is that $\Phi(l, m)$ will contain all the labels of methods that might be the m-method of an object labelled l.

We define $\Phi(l) \sqsubseteq \Phi(l')$ as the conjunction of the formulas $\Phi(l, m) \subseteq \Phi(l', m)$ for all method names m.

For dealing with the analysis, ordinary comprehensions are cumbersome. We introduce reverse comprehensions:

$$(\mathsf{ForEach}\ x \in X \ \ \triangleright \ \ f(x))$$

in place of the more usual $\{f(x) \ \mid \ x \in X\}$ or $\bigcup_{x \in X} f(x)$, depending on whether $f(x)$ is a formula or a set of formulas.

We may now define the analysis $[\![e]\!]$ of a term e as the set of Horn clauses defined by:

$$
\begin{aligned}
[\![e]\!] = {} & (\mathsf{ForEach}\ \langle m_i =^{l_i} (x_i^{\beta_i})e_i\rangle_i^l \in_{\mathsf{E}} e \ \ \triangleright\ l_i \in \Phi(l, m_i)) \\
& \cup (\mathsf{ForEach}\ (e_1.m)^l \in_{\mathsf{E}} e, m =^{l'} (x^\beta)e_2 \in_{\mathsf{E}} e \\
& \qquad \triangleright\ l' \in \Phi(\mathsf{lab}(e_1), m) \Rightarrow \Phi(\mathsf{lab}(e_1)) \sqsubseteq \Phi(\beta), \\
& \qquad\ \ l' \in \Phi(\mathsf{lab}(e_1), m) \Rightarrow \Phi(\mathsf{lab}(e_2)) \sqsubseteq \Phi(l)) \\
& \cup (\mathsf{ForEach}\ (e_1 \Leftarrow (m =^{l_1} (x^\beta)e_1'))^l \in_{\mathsf{E}} e \\
& \qquad \triangleright\ (\forall n \neq m)(\Phi(\mathsf{lab}(e_1), n) \subseteq \Phi(l, n)), \\
& \qquad\ \ l_1 \in \Phi(l, m))
\end{aligned}
$$

As in Sect. 3, each comprehension in the analysis has an intuitive relation to the semantics:

- Every m-method that appears in an object is a possible m-method of that object.
- If an m-method of an object is selected, then the object is among the possible values of the method's bound variable, and the value of the method's body is among the possible values of the selection expression.
- If an object is created by updating the m-method of some object, then every n-method of the old object is a possible n-method of the new object (for $n \neq m$), and the new method is a possible m-method of the new object.

We now proceed as in Sect. 4.

Definition 5.1. *If r is a redex and s is an expression such that $r \to s$, E is a reduction context, and e, e' are expressions such that $e \in_{\mathsf{E}} E[r]$ and $e' \in_{\mathsf{E}} E[s]$, then define $\mathbb{S}(e, e')$ iff either*

1. $\mathsf{lab}(e) = \mathsf{lab}(e')$,
2. *$e = r$ and $e' = s$, or else*
3. *$r = (\langle m_i =^{l_i} (x_i^{\beta_i})e_i\rangle_i^{l_1}.m_j)^{l_2}$ and $s = e_j[\langle m_i =^{l_i} (x_i^{\beta_i})e_i\rangle_i^{l_1}/x_j]$ and $e = x_j^{\beta_j}$ and $e' = \langle m_i =^{l_i} (x_i^{\beta_i})e_i\rangle_i^{l_1}$.*

Lemma 5.2. *Assume r is a redex and s an expression such that $r \to s$, E is a reduction context, and $\vdash E[r]$ wl. Then*

1. *If $(m =^l (x^\beta)e_1') \in_{\mathsf{E}} E[s]$, then there exists a $(m =^l (x^\beta)e_1) \in_{\mathsf{E}} E[r]$ such that $\mathbb{S}(e_1, e_1')$.*
2. *If $(e_1'.m)^l \in_{\mathsf{E}} E[s]$, then there exists a $(e_1.m)^l \in_{\mathsf{E}} E[r]$ such that $\mathbb{S}(e_1, e_1')$.*
3. *If $(e_1' \Leftarrow (m =^{l_2} (x^\beta)e_2'))^{l_1} \in_{\mathsf{E}} E[s]$, then there exists a $(e_1 \Leftarrow (m =^{l_2} (x^\beta)e_2))^{l_1}$ such that $\mathbb{S}(e_1, e_1')$ and $\mathbb{S}(e_2, e_2')$.*
4. *If $\langle m_i =^{l_i'} (b_i)e_i'\rangle_i^l \in_{\mathsf{E}} E[s]$ then either*
   (a) *There exists $\langle m_i =^{l_i'} (b_i)e_i\rangle_i^l \in_{\mathsf{E}} E[r]$ such that $\mathbb{S}(e_i, e_i')$ for all i, or*
   (b) *$r \to s$ is an instance of (Red Upd) and $s = \langle m_i =^{l_i'} (b_i)e_i'\rangle_l^i$ and there exists $(\langle m_i =^{l_i} (x_i^{\beta_i})e_i\rangle_i^{l_1} \Leftarrow (m_j =^{l_2} (x^\beta)e))^l \in_{\mathsf{E}} E[r]$ such that for all i,*

$$
\begin{aligned}
& l_i' = l_i,\ b_i = x_i^{\beta_i},\ e_i' = e_i \quad \text{for } i \neq j \\
& l_i' = l_2,\ b_i = x^\beta,\ e_i' = e \quad\ \ \text{for } i = j
\end{aligned}
$$

Proof. By an analysis like the one for Lem. 4.6. The only additional case is that of an object constructed by (Red Upd). But in this case we know exactly what the redex must have looked like, and it is described in the last case of the lemma.

Lemma 5.3. *If r is a redex and s an expression such that $r \to s$, E is a reduction context, $\vdash E[r]$ wl, and e and e' are expressions such that $e \in_{\mathsf{E}} E[r]$, $e' \in_{\mathsf{E}} E[s]$, and $\$(e, e')$, then $[\![E[r]]\!] \vdash \Phi(\mathsf{lab}(e')) \sqsubseteq \Phi(\mathsf{lab}(e))$.*

Proof. By cases on the definition of $\$(e, e')$, as in the proof of Lem. 4.7.

Theorem 5.4. *If $e_X \to e_Y$, and $\vdash e_X$ wl, then $[\![e_X]\!] \vdash [\![e_Y]\!]$ and $[\![e_X]\!] \vdash \Phi(\mathsf{lab}(e_Y)) \sqsubseteq \Phi(\mathsf{lab}(e_X))$.*

Proof. From the constraints in $[\![E[r]]\!]$, we need to deduce each of the constraints in $[\![E[s]]\!]$, given by

$$\begin{aligned}
[\![E[s]]\!] = {} & (\mathsf{ForEach}\ \langle m_i =^{l'_i} (x_i^{\beta_i})e_i\rangle_i^l \in_{\mathsf{E}} E[s] \ \triangleright\ l'_i \in \Phi(l, m_i)) \\
& \cup (\mathsf{ForEach}\ (e_1.m)^l \in_{\mathsf{E}} E[s], m =^{l'} (x^\beta)e_2 \in_{\mathsf{E}} E[s] \\
& \qquad \triangleright\ l' \in \Phi(\mathsf{lab}(e_1), m) \Rightarrow \Phi(\mathsf{lab}(e_1)) \sqsubseteq \Phi(\beta), \\
& \qquad\quad l' \in \Phi(\mathsf{lab}(e_1), m) \Rightarrow \Phi(\mathsf{lab}(e_2)) \sqsubseteq \Phi(l)) \\
& \cup (\mathsf{ForEach}\ (e_1 \Leftarrow (m =^{l_1} (x^\beta)e'_1))^l \in_{\mathsf{E}} E[s] \\
& \qquad \triangleright\ (\forall n \neq m)(\Phi(\mathsf{lab}(e_1), n) \subseteq \Phi(l, n)), \\
& \qquad\quad l_1 \in \Phi(l, m))
\end{aligned}$$

First consider the case $(\mathsf{ForEach}\ \langle m_i =^{l'_i} (x_i^{\beta_i})e_i\rangle_i^l \in_{\mathsf{E}} E[s] \ \triangleright\ l'_i \in \Phi(l, m_i))$. By Lem. 5.2, if $\langle m_i =^{l'_i} (b_i)e'_i\rangle_i^l \in_{\mathsf{E}} E[s]$ then either

1. There exists $\langle m_i =^{l'_i} (b_i)e_i\rangle_i^l \in_{\mathsf{E}} E[r]$ such that $\$(e_i, e'_i)$ for all i, or
2. $r \to s$ is an instance of (Red Upd) and $s = \langle m_i =^{l'_i} (b_i)e'_i\rangle_l^i$ and there exists $r = (\langle m_i =^{l_i} (x_i^{\beta_i})e_i\rangle_i^{l_1} \Leftarrow (m_j =^{l_2} (x^\beta)e))^l \in_{\mathsf{E}} E[r]$ such that for all i,

$$\begin{array}{ll}
l'_i = l_i,\ b_i = x_i^{\beta_i},\ e'_i = e_i & \text{for } i \neq j \\
l'_i = l_2,\ b_i = x^\beta,\ e'_i = e & \text{for } i = j
\end{array}$$

In the first case, each of the formulas $l'_i \in \Phi(l, m_i)$ is already in $[\![E[r]]\!]$. In the second case, since $E[r]$ contains the update term r, $[\![E[r]]\!]$ contains the formulas

$$\begin{array}{ll}
\Phi(l_1, m_i) \subseteq \Phi(l, m_i) & i \neq j \\
l_2 \in \Phi(l, m_j)
\end{array}$$

which describe the possible methods in abstract object l. Since r contains an object term, $[\![E[r]]\!]$ also contains the formulas $l_i \in \Phi(l_1, m_i)$ for each i.

We can now consider each of the formulas $l'_i \in \Phi(l, m_i)$. For $i \neq j$, $[\![E[r]]\!] \vdash l_i \in \Phi(l_1, m_i) \subseteq \Phi(l, m_i)$, and $l'_i = l_i$, so $[\![E[r]]\!] \vdash l'_i \in \Phi(l, m_i)$, as desired. For $i = j$, $[\![E[r]]\!] \vdash l_2 \in \Phi(l, m_j)$, but $l'_i = l_2$ and $m_i = m_j$, so $[\![E[r]]\!] \vdash l'_i \in \Phi(l, m_j)$ as desired.

For the remaining cases, we rely on Lem. 4.7, as in Thm. 4.8. The second half follows analogously to the second half of Thm. 4.8.

6 Conclusion and Further Work

We have introduced a novel proof technique for the correctness of constraint-based flow analyses, and used it to provide a complete proof for the central theorem of [8]. By matching the structure of the proof to the flat structure of the analysis, we have not only clarified the overall flow of the proof, but we have also exposed the crucial effects of reduction on the subterms of the expression under analysis. Furthermore, by structuring the case analysis of the proof around reduction contexts and redices instead of around the reductions of whole expressions, we have simplified the necessary steps of the proof. Finally, we have demonstrated our technique's extensibility by applying it to an analysis for a larger language.

We intend to investigate how well our proof technique will scale up to analyses of larger, more realistic languages. In particular, we would like to know how well it handles analyses that do not take the whole program into account. A related investigation is to consider coinductive analyses like [7], in which not all expressions are analyzed.

References

[1] M. Abadi and L. Cardelli. A theory of primitive objects: Untyped and first-order systems. *Information and Computation*, 125(2):78–102, Mar. 1996.

[2] A. Aiken and N. Heintze. Constraint-based program analysis. In *POPL'95 Tutorial*, January 1995.

[3] P. D. Blasio, K. Fisher, and C. Talcott. Analysis for concurrent objects. In H. Bowman and J. Derrick, editors, *Proc. 2nd IFIP Workshop on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, pages 73–88, Canterbury, UK, July 1997. Chapman and Hall, London.

[4] C. Bodei, P. Degano, F. Nielson, and H. R. Nielson. Control flow analysis for the π-calculus. In *Proceedings of CONCUR'98*, pages 611–638, Berlin, Heidelberg, and New York, 1998. Springer-Verlag.

[5] C. Flanagan and M. Felleisen. Set-based analysis for full scheme and its use in soft-typing. Technical Report COMP TR95-253, Department of Computer Science, Rice University, Oct. 1995.

[6] J. C. Mitchell. *Foundations for Programming Languages*. MIT Press, Cambridge, MA, 1996.

[7] F. Nielson and H. R. Nielson. Infinitary control flow analysis: a collecting semantics for closure analysis. In *Proceedings 24th Annual ACM Symposium on Principles of Programming Languages*, pages 332–345. ACM, Jan. 1997.

[8] J. Palsberg. Closure analysis in constraint form. *ACM Transactions on Programming Languages and Systems*, 17(1):47–62, January 1995.

[9] P. Sestoft. Replacing function parameters by global variables. Master's thesis, DIKU, University of Copenhagen, Copenhagen, 1989.

[10] P. Sestoft. *Analysis and efficient implementation of functional programs*. PhD thesis, DIKU, University of Copenhagen, Copenhagen, 1991.

[11] O. Shivers. *Control-Flow Analysis of Higher-Order Languages*. PhD thesis, Carnegie-Mellon University, May 1991.