Closure Analysis in Constraint Form

JENS PALSBERG
Aarhus University

Flow analyses of untyped higher-order functional programs have in the past decade been presented by Ayers, Bondorf, Consel, Jones, Heintze, Sestoft, Shivers, Steckler, Wand, and others. The analyses are usually defined as abstract interpretations and are used for rather different tasks such as type recovery, globalization, and binding-time analysis. The analyses all contain a global *closure analysis* that computes information about higher-order control-flow. Sestoft proved in 1989 and 1991 that closure analysis is correct with respect to call-by-name and call-by-value semantics, but it remained open if correctness holds for arbitrary beta-reduction.

This article answers the question; both closure analysis and others are correct with respect to arbitrary beta-reduction. We also prove a subject-reduction result: closure information is still valid after beta-reduction. The core of our proof technique is to define closure analysis using a constraint system. The constraint system is equivalent to the closure analysis of Bondorf, which in turn is based on Sestoft's.

Categories and Subject Descriptors: D.3.1 [**Programming Languages**]: Formal Definitions and Theory—*semantics*; D.3.2 [**Programming Languages**]: Language Classifications—*applicative languages*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*logics of programs*

General Terms: Languages, Theory

Additional Key Words and Phrases: Constraints, correctness proof, flow analysis

1. INTRODUCTION

1.1 Background

The optimization of higher-order functional languages requires powerful program analyses. The traditional framework for such analyses is *abstract interpretation*, and for *typed* languages, suitable abstract domains can often be defined by induction on the structure of types. For example, function spaces can be abstracted into function spaces. For *untyped* languages such as the λ-calculus, or dynamically typed languages such as Scheme, abstract domains cannot be defined by abstracting function spaces into function spaces. Other domains can be used, but it may then be difficult to relate the abstract interpretation to the *denotational* semantics. In this article we consider a style of program analysis where the result is an abstraction of the *operational* semantics.

In the past decade, program analyses of untyped languages has been presented

Author's address: Computer Science Department, Aarhus University, Ny Munkegade, DK–8000 Aarhus C, Denmark; email: palsberg@daimi.aau.dk.

ACM Transactions on Programming Languages and Systems, 17(1):47–62, January 1995. Also in Proc. CAAP'94, pages 276–290. ©

by Ayers [1992], Bondorf [1991], Consel [1990], Jones [1981], Heintze [1992], Sestoft [1989; 1991], Shivers [1991a; 1991b], Wand and Steckler [1994], and others. Although the analyses are used for rather different tasks such as type recovery, globalization, and binding-time analysis, they are all based on essentially the same idea:

> *Key idea.* In the absence of types, define the abstract domains in terms of *program points*.

For example, consider the following λ-term:

> $(\lambda x.\lambda y.y(xI)(xK))\Delta$
> where $I = \lambda a.a$, $K = \lambda b.\lambda c.b$, and $\Delta = \lambda d.dd$.

Giannini and Rocca [1988] proved that this strongly normalizing term has no higher-order polymorphic type. Still, a program analysis might answer basic questions such as:

(1) For every application point, which abstractions can be applied?
(2) For every abstraction, to which arguments can it be applied?

Each answer to such questions should be a subset of the program points in this particular λ-term. Thus, let us label all abstractions and applications. Also variables will be labeled: if a variable is bound, then it is labeled with the label of the λ that binds it, and if it is free, then with an arbitrary label. By introducing an explicit application symbol, we get the following abstract syntax for the above λ-term.

> $(\lambda^1 x.\lambda^2 y.y^2\ @_7\ (x^1\ @_8\ I)\ @_9\ (x^1\ @_{10}\ K))\ @_{11}\ \Delta$
> where $I = \lambda^3 a.a^3$, $K = \lambda^4 b.\lambda^5 c.b^4$, and $\Delta = \lambda^6 d.d^6\ @_{12}\ d^6$.

An analysis might be able to find out that no matter how reduction proceeds:

—"I can only be applied to I," that is, an abstraction with label 3 can only be applied to abstractions with label 3;
—"At the application point dd (in Δ) both I and K can be applied," that is, at an application point labeled 12 there can only be applied abstractions with labels 3 and 4; and
—"the abstraction $\lambda c.b$ will never be applied," that is, at no application point can an abstraction with label 5 be applied.

The quoted sentences give the intuitive understanding of the precise statements that follow. In this particular example, the labels are rather unnecessary because no name clashes happen during any reduction and because I, K, and $\lambda c.b$ are in normal form. In the presence of name clashes or reduction under a λ, however, it is crucial to use sets of program points as the abstract values.

The above questions have turned out to be of paramount importance in many analyses of untyped functional programs. Following Sestoft and Bondorf, we will call any analysis that can answer them conservatively a *closure analysis*. On top of a closure analysis, one can build for example type recovery analyses, globalization analyses, and binding-time analyses. The closure analysis answers questions about higher-order control flow, and the extension answers the questions one is really interested in, for example, about type recovery. The role of closure analysis is thus as follows:

"*Higher-order analysis = first-order analysis + closure analysis.*"

Closure analysis is useful for higher-order languages in general, for example, object-oriented languages (see Palsberg and Schwartzbach [1991; 1994b]). It is also useful for typed functional languages because type information is usually not specific enough to tell which functions among the type-correct ones are called at each application point.

Closure analysis and its extensions can be defined as abstract interpretations. They differ radically from traditional abstract interpretations, however, in that the abstract domain is defined in terms of the program to be analyzed. This means that such analyses are *global*: before the abstract domain can be defined, the complete program is required. Moreover, the program cannot take higher-order input because that would add program points. Also the minimal function graph approach to program analysis uses abstract domains defined in terms of the input program. In contrast, traditional abstract interpretations can analyze pieces of a program in isolation. We will refer to all analyses based on closure analysis as *flow analyses*.

Examples of large-scale implementations of such analyses can be found in the Similix system of Bondorf [Bondorf 1993; Bondorf and Danvy 1991], the Schism system of Consel [1990], and the system of Agesen et al. [1993] for analyzing Self programs [Ungar and Smith 1987]. The last of these implementations demonstrates that closure analysis can handle dynamic and multiple inheritance.

Closure analysis and its extensions have been formulated using constraints by others, for example, Heintze [1992; 1994], and Wand and Steckler [1994]. Their constraint systems are in spirit close to ours, although they are technically somewhat different. A key difference between Heintze's definition [Heintze 1994] and ours is that he attempts to avoid analyzing code that will not be executed under call-by-value. This goal is shared by an analysis of Palsberg and Schwartzbach [1992a]. The idea of defining program analyses using constraints over set variables is called *set-based analysis* by Heintze.

Sestoft [1989; 1991] proved that closure analysis is correct with respect to call-by-name and call-by-value semantics, but it remained open if correctness holds for arbitrary beta-reduction.

1.2 Our Results

We prove that closure analysis is correct with respect to arbitrary beta-reduction. We also prove a subject-reduction result: closure information is still valid after beta-reduction. The correctness result implies that closure analysis is correct with respect to any reduction strategy.

—We present a novel *specification* of closure analysis that allows arbitrary beta-reduction to take place and which subsumes all previous specifications.

—We present a closure analysis that uses a *constraint system*. The constraint system characterizes the result of the analysis without specifying how it is computed. An example of such a constraint system is given in Section 1.3.

—We prove that the constraint-based analysis is equivalent to the closure analysis of Bondorf [1991], which in turn is based on Sestoft's [Sestoft 1989]. We also

prove that these analyses are equivalent to a novel simplification of Bondorf's definition.

The proofs of correctness and subject-reduction then proceed by considering only the constraint-based definition of closure analysis.

In contrast to the closure analyses by abstract interpretation, the one using a constraint system does *not* depend on labels being distinct. This makes it possible to analyze a λ-term, beta-reduce it, and then analyze the result *without* relabeling first. The abstract interpretations might be modified to have this property also, but it would be somewhat messy. This indicates that a direct proof of correctness of such a modified abstract interpretation would be more complicated than the proof presented in this article.

Our technique for proving correctness generalizes without problems to analyses based on closure analysis. The following two results are not proved in this article:

—The *safety analysis* of Palsberg and Schwartzbach [1992a; 1992b] is correct with respect to arbitrary beta-reduction. This follows from the subject-reduction property: terms stay safe after beta-reduction.

—The *binding-time analysis* of Palsberg and Schwartzbach [1994a] that was proved correct by Palsberg [1993], can be proved correct more elegantly with our new technique.

The constraint-based definition of closure analysis is straightforward to extend to practical languages. For a medium-sized example see Palsberg and Schwartzbach [1994b] where the analysis is defined for an object-oriented language.

1.3 Example

The constraint system that expresses closure analysis of a λ-term is a set of Horn clauses. If the λ-term contains n abstractions and m applications, then the constraint system contains $n + (2 \times m \times n)$ constraints. Thus, the size of a constraint system is in the worst-case quadratic in the size of the λ-term. Space constraints disallow us to show a full-blown example involving name clashes and reduction under a λ, so consider instead the λ-term $(\lambda x.xx)(\lambda y.y)$ which has the abstract syntax $(\lambda^1 x.x^1\ \mathtt{@}_3\ x^1)\ \mathtt{@}_4\ (\lambda^2 y.y^2)$. The constraint system that expresses closure analysis of this λ-term looks as follows.

$$
\begin{array}{ll}
\text{From } \lambda^1 & \{1\} \subseteq [\![\lambda^1]\!] \\
\text{From } \lambda^2 & \{2\} \subseteq [\![\lambda^2]\!] \\
\text{From } \mathtt{@}_3 \text{ and } \lambda^1 & \left\{ \begin{array}{l} \{1\} \subseteq [\![\nu^1]\!] \Rightarrow [\![\nu^1]\!] \subseteq [\![\nu^1]\!] \\ \{1\} \subseteq [\![\nu^1]\!] \Rightarrow [\![\mathtt{@}_3]\!] \subseteq [\![\mathtt{@}_3]\!] \end{array} \right. \\
\text{From } \mathtt{@}_3 \text{ and } \lambda^2 & \left\{ \begin{array}{l} \{2\} \subseteq [\![\nu^1]\!] \Rightarrow [\![\nu^1]\!] \subseteq [\![\nu^2]\!] \\ \{2\} \subseteq [\![\nu^1]\!] \Rightarrow [\![\nu^2]\!] \subseteq [\![\mathtt{@}_3]\!] \end{array} \right. \\
\text{From } \mathtt{@}_4 \text{ and } \lambda^1 & \left\{ \begin{array}{l} \{1\} \subseteq [\![\lambda^1]\!] \Rightarrow [\![\lambda^2]\!] \subseteq [\![\nu^1]\!] \\ \{1\} \subseteq [\![\lambda^1]\!] \Rightarrow [\![\mathtt{@}_3]\!] \subseteq [\![\mathtt{@}_4]\!] \end{array} \right. \\
\text{From } \mathtt{@}_4 \text{ and } \lambda^2 & \left\{ \begin{array}{l} \{2\} \subseteq [\![\lambda^1]\!] \Rightarrow [\![\lambda^2]\!] \subseteq [\![\nu^2]\!] \\ \{2\} \subseteq [\![\lambda^1]\!] \Rightarrow [\![\nu^2]\!] \subseteq [\![\mathtt{@}_4]\!] \end{array} \right.
\end{array}
$$

Symbols of the forms $[\![\nu^l]\!]$, $[\![\lambda^l]\!]$, and $[\![\mathtt{@}_i]\!]$ are metavariables. They relate to variables with label l, abstractions with label l, and applications with label i, respectively. Notice that we do *not* assume, for example, that there is just one abstraction

with label l. The reason is that we want to do closure analysis of *all* terms, also those arising after beta-reduction which may copy terms and hence labels.

To the left of the constraints, we have indicated from where they arise. The first two constraints express that an abstraction may evaluate to an abstraction with the same label. The rest of the constraints come in pairs. For each application point $@_i$ and each abstraction with label l there are two constraints of the form:

$$\{l\} \subseteq \text{“metavar. for operator of } @_i\text{”} \quad \Rightarrow \quad \text{“metavar. for operand of } @_i\text{”} \subseteq [\![\nu^l]\!]$$
$$\{l\} \subseteq \text{“metavar. for operator of } @_i\text{”} \quad \Rightarrow \quad \text{“metavar. for body of abst.”} \subseteq [\![@_i]\!]$$

Such constraints can be read as:

—*The first constraint.* If the operator of $@_i$ evaluates to an abstraction with label l, then the bound variable of that abstraction may be substituted with everything to which the operand of $@_i$ can evaluate.

—*The second constraint.* If the operator of $@_i$ evaluates to an abstraction with label l, then everything to which the body of the abstraction evaluates is also a possible result of evaluating the whole application $@_i$.

In a solution of the constraint system, metavariables are assigned closure information. The minimal solution of the above constraint system is a mapping L where:

$$L[\![\lambda^1]\!] = \{1\}$$
$$L[\![\lambda^2]\!] = L[\![\nu^1]\!] = L[\![\nu^2]\!] = L[\![@_3]\!] = L[\![@_4]\!] = \{2\}$$

For example, the whole λ-term will, if normalizing, evaluate to an abstraction with label 2 ($L[\![@_4]\!] = \{2\}$); at the application point $@_3$ there can only be applied abstractions with label 2 ($L[\![\nu^1]\!] = \{2\}$); the application point $@_3$ is the only point where abstractions with label 2 can be applied ($L[\![\lambda^1]\!] = \{1\}$); and such abstractions can only be applied to λ-terms that either do not normalize or evaluate to an abstraction with label 2 ($L[\![\nu^2]\!] = \{2\}$).

One of our theorems says that the computed closure information is correct. One might also try to do closure analysis of the above λ-term using Bondorf's abstract interpretation; another of our theorems says that we will get the same result.

Now contract the only redex in the above λ-term. The result is a λ-term with abstract syntax $(\lambda^2 y.y^2)\ @_3\ (\lambda^2 y.y^2)$. One third of our theorems says that the mapping L above gives correct closure information also for this λ-term.

In the following section we define three closure analyses: Bondorf's, a simpler abstract interpretation, and one in constraint form. In Section 3 we prove that they are equivalent, and in Section 4 we prove that they are correct.

2. CLOSURE ANALYSIS

Recall the λ-calculus [Barendregt 1981].

Definition 2.1. The language Λ of λ-terms has an abstract syntax which is defined by the grammar:

$$\begin{array}{lll} E & ::= & x^l \qquad\qquad \text{(variable)} \\ & | & \lambda^l x.E \qquad \text{(abstraction)} \\ & | & E_1\ @_i\ E_2 \quad \text{(application)} \end{array}$$

The labels on variables, abstraction symbols, and application symbols have no semantic impact; they mark program points. The label on a bound variable is the same as that on the λ that binds it. Labels are drawn from the infinite set Label. The symbols l, l', i range over labels. The labels and the application symbols are not part of the concrete syntax of Λ. We identify terms that are α-congruent. The α-conversion changes only bound variables, not labels. We assume the Variable Convention of Barendregt [1981]: when a λ-term occurs in this article, all bound variables are chosen to be different from the free variables. This can be achieved by renaming bound variables. An occurrence of $(\lambda^l x.E)\ @_i\ E'$ is called a redex. The semantics is as usual given by the rewriting-rule scheme:

$$(\lambda^l x.E)\ @_i\ E' \to E[E'/x^l] \qquad \text{(beta-reduction)}\ .$$

Here, $E[E'/x^l]$ denotes the term E with E' substituted for the free occurrences of x^l. Notice that by the Variable Convention, no renaming of bound variables is necessary when doing substitution. In particular, when we write $(\lambda^l y.E)[E'/x^{l'}]$, we have that $y^l \not\equiv x^{l'}$ and that y^l is not among the free variables of E'. Thus, $(\lambda^l y.E)[E'/x^{l'}] = \lambda^l y.(E[E'/x^{l'}])$. We write $E_S \to^* E_T$ to denote that E_T has been obtained from E_S by 0 or more beta-reductions. A term without redexes is in normal form.

The abstract domain for closure analysis of a λ-term E is called $\mathsf{CMap}(E)$ and is defined as follows.

Definition 2.2. A *metavariable* is of one of the forms $[\![\nu^l]\!]$, $[\![\lambda^l]\!]$, and $[\![@_i]\!]$. The set of all metavariables is denoted Metavar. A λ-term is assigned a metavariable by the function var, which maps x^l to $[\![\nu^l]\!]$, $\lambda^l x.E$ to $[\![\lambda^l]\!]$, and $E_1\ @_i\ E_2$ to $[\![@_i]\!]$.

For a λ-term E, $\mathsf{Lab}(E)$ is the set of labels on abstractions (but not applications) occurring in E. Notice that $\mathsf{Lab}(E)$ is finite. The set $\mathsf{CSet}(E)$ is the powerset of $\mathsf{Lab}(E)$; $\mathsf{CSet}(E)$ with the inclusion ordering is a complete lattice. The set $\mathsf{CMap}(E)$ consists of the total functions from Metavar to $\mathsf{CSet}(E)$. The set $\mathsf{CEnv}(E)$ contains each function in $\mathsf{CMap}(E)$ when restricted to metavariables of the form $[\![\nu^l]\!]$. Both $\mathsf{CMap}(E)$ and $\mathsf{CEnv}(E)$ with pointwise ordering, written $\sqsubseteq$, are complete lattices where the least upper bound is written $\sqcup$. The function $\langle V \mapsto S\rangle$ maps the metavariable V to the set S and maps all other metavariables to the empty set. Finally, we define $upd\ V\ S\ L = \langle V \mapsto S\rangle \sqcup L$.

2.1 The Specification of Closure Analysis

We can then state precisely what a closure analysis is. An intuitive argument follows the formal definition.

Definition 2.1.1. For a λ-term E and for every $L \in \mathsf{CMap}(E)$, we define a binary relation T_L on λ-terms, as follows. $T_L(E_X, E_Y)$ holds if and only if the following four conditions hold:

—If E_Y equals $\lambda^l x.E$, then $\{l\} \subseteq L(\mathsf{var}(E_X))$.

—If E_Y contains $\lambda^{l'} y.(\lambda^l x.E)$, then E_X contains $\lambda^{l'} z.E'$ such that $\{l\} \subseteq L(\mathsf{var}(E'))$.

—If E_Y contains $(\lambda^l x.E)\ @_i\ E_2$, then E_X contains $E_1\ @_i\ E_2'$ such that $\{l\} \subseteq L(\mathsf{var}(E_1))$.

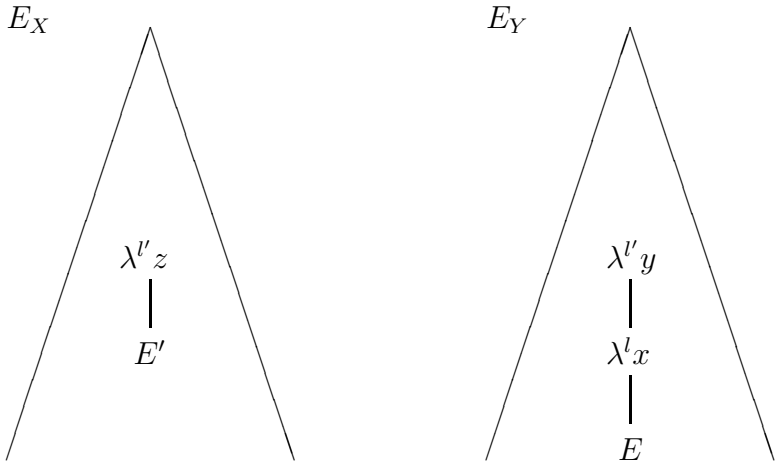
—If E_Y contains E_1 $@_i$ $(\lambda^l x.E)$, then E_X contains E_1' $@_i$ E_2 such that $\{l\} \subseteq L(\mathsf{var}(E_2))$.

A *closure analysis* of E produces $L \in \mathsf{CMap}(E)$ such that if $E \rightarrow^* E'$, then $T_L(E, E')$.

Intuitively, if $E_X \rightarrow^* E_Y$, then we can get conservative information about the abstractions in E_Y by doing closure analysis of E_X. For example, the first condition in Definition 2.1.1 can be illustrated as follows.

In this case, E_Y is an abstraction with label l. Thus, E_X can evaluate to an abstraction with label l. The first condition says that in this case the mapping L must satisfy $\{l\} \subseteq L(\mathsf{var}(E_X))$. In other words, the analysis must be aware that such an abstraction is a possible result of evaluating E_X.

The three other conditions in Definition 2.1.1 cover the cases where abstractions are proper subterms of E_Y. The second condition covers the case where an abstraction in E_Y is the body of yet another abstraction. The third and fourth conditions cover the cases where an abstraction is the operator and the operand of an application, respectively. Here, we will illustrate just the first of these three conditions; the others are similar.



In this case, E_Y contains an abstraction with label l $(\lambda^l x.E)$. This abstraction is in turn the body of an abstraction with label l' $(\lambda^{l'} y.\lambda^l x.E)$. The second condition

in Definition 2.1.1 says that in this case there must be an abstraction in E_X with label l' ($\lambda^{l'} z.E'$, the bound variable may be different) such that the mapping L satisfies $\{l\} \subseteq L(\mathsf{var}(E'))$. In other words, the analysis must be aware that some abstraction $\lambda^{l'} z.E'$ in E_X can evolve into an abstraction with a body being an abstraction with label l.

Notice the possibility that more than one abstraction in E_X has label l'. Thus, if we want closure information for "the body of the abstraction with label l'" we must compute the *union* of information for the bodies of *all* abstractions in E_X with label l'. A similar comment applies to the third and fourth condition in Definition 2.1.1. Such use of closure information is not of concern in this article, however.

2.2 Bondorf's Definition

We now recall the closure analysis of Bondorf [1991], with a few minor changes in the notation compared to his presentation. The analysis assumes that all labels are distinct. Bondorf's definition was originally given for a subset of Scheme; we have restricted it to the λ-calculus. Note that Bondorf's definition is based on Sestoft's [Sestoft 1989].

We have simplified Bondorf's definition as follows. Bondorf's original definition assigns *distinct* metavariables to different occurrences of a variable; in contrast we assign the *same* metavariable to each occurrence of a variable. The simplified definition is equivalent to Bondorf's original definition; see below.

We will use the notation that if $\lambda^l x.E$ is a subterm of the term to be analyzed, then the partial function *body* maps the label l to E.

Definition 2.2.1. We define

$$
\begin{array}{l}
B : (E : \Lambda) \to \mathsf{CMap}(E) \times \mathsf{CEnv}(E) \\
B(E) = \mathit{fix}(\lambda(\mu,\rho).b(E)\mu\rho)
\end{array}
$$

$$
\begin{array}{l}
b : (E : \Lambda) \to \mathsf{CMap}(E) \to \mathsf{CEnv}(E) \to \mathsf{CMap}(E) \times \mathsf{CEnv}(E) \\
b(x^l)\mu\rho = (\mathit{upd}\ [\![\nu^l]\!]\ \rho[\![\nu^l]\!]\ \mu, \rho) \\
b(\lambda^l x.E)\mu\rho = \text{let } (\mu',\rho') \text{ be } b(E)\mu\rho \\
\qquad\qquad \text{in } (\mathit{upd}\ [\![\lambda^l]\!]\ \{l\}\ \mu', \rho') \\
b(E_1\ @_i\ E_2)\mu\rho = \text{let } (\mu',\rho') \text{ be } (b(E_1)\mu\rho) \sqcup (b(E_2)\mu\rho) \text{ in} \\
\qquad\qquad \text{let } c \text{ be } \mu'(\mathsf{var}(E_1)) \text{ in} \\
\qquad\qquad \text{let } \mu'' \text{ be } \mathit{upd}\ [\![@_i]\!]\ (\sqcup_{l\in c}\ \mu'(\mathsf{var}(\mathit{body}(l))))\ \mu' \text{ in} \\
\qquad\qquad \text{let } \rho'' \text{ be } \rho' \sqcup (\sqcup_{l\in c}\ (\mathit{upd}\ [\![\nu^l]\!]\ \mu'(\mathsf{var}(E_2)))\ \rho') \\
\qquad\qquad \text{in } (\mu'',\rho'')\ .
\end{array}
$$

We can now do closure analysis of E by computing $\mathit{fst}(B(E))$.

If we modify the above definition such that different occurrences of a variable are assigned distinct metavariables, then we obtain Bondorf's original definition. That definition will assign the *same* set to all metavariables for the occurrences of a given variable, and moreover, the computed closure information will be the same as that computed by the stated analysis (we leave the details to the reader).

2.3 A Simpler Abstract Interpretation

Bondorf's definition can be simplified considerably. To see why, consider the second component of $\mathsf{CMap}(E) \times \mathsf{CEnv}(E)$. This component is updated only in

$b(E_1\ @_i\ E_2)\mu\rho$ and read only in $b(x^l)\mu\rho$. The key observation is that both these operations can be done on the first component instead. Thus, we can omit the use of $\mathsf{CEnv}(E)$. By rewriting Bondorf's definition according to this observation, we arrive at the following definition. As with Bondorf's definition, we assume that all labels are distinct.

Definition 2.3.1. We define

$$
\begin{aligned}
&m : (E : \Lambda) \to \mathsf{CMap}(E) \to \mathsf{CMap}(E)\\
&m(x^l)\mu = \mu\\
&m(\lambda^l x.E)\mu = (m(E)\mu) \sqcup \langle [\![\lambda^l]\!] \mapsto \{l\}\rangle\\
&m(E_1\ @_i\ E_2)\mu = (m(E_1)\mu) \sqcup (m(E_2)\mu) \sqcup\\
&\qquad\qquad \bigsqcup_{l\in\mu(\mathsf{var}(E_1))} (\langle [\![\nu^l]\!] \mapsto \mu(\mathsf{var}(E_2))\rangle \sqcup \langle [\![@_i]\!] \mapsto \mu(\mathsf{var}(\mathit{body}(l)))\rangle) \ .
\end{aligned}
$$

We can now do closure analysis of E by computing $\mathit{fix}(m(E))$.

A key question is: is the simpler abstract interpretation equivalent to Bondorf's? We might attempt to prove this using fixed-point induction, but we find it much easier to do using a particular constraint system as a "stepping stone."

2.4 A Constraint System

For a λ-term E, the constraint system is a finite set of Horn clauses over inclusions of the form $P \subseteq P'$, where P and P' are either metavariables or elements of $\mathsf{CSet}(E)$. A *solution* of such a system is an element of $\mathsf{CMap}(E)$ that satisfies all Horn clauses.

The constraint system is defined in terms of the λ-term to be analyzed. We need *not* assume that all labels are distinct.

The set $R(E_1\ @_i\ E_2, \lambda^l x.E)$ consists of the two elements

$$
\begin{aligned}
\{l\} \subseteq \mathsf{var}(E_1) &\Rightarrow \mathsf{var}(E_2) \subseteq [\![\nu^l]\!]\\
\{l\} \subseteq \mathsf{var}(E_1) &\Rightarrow \mathsf{var}(E) \ \subseteq [\![@_i]\!] \ .
\end{aligned}
$$

For a λ-term E, the constraint system $C(E)$ is the union of the following sets of constraints.

—For every $\lambda^l x.E'$ in E, the singleton constraint set consisting of $\{l\} \subseteq [\![\lambda^l]\!]$.

—For every $E_1\ @_i\ E_2$ in E and for every $\lambda^l x.E'$ in E, the set $R(E_1\ @_i\ E_2, \lambda^l x.E')$.

Each $C(E)$ has a least solution, namely, the intersection of all solutions.

We can now do closure analysis of E by computing a solution of $C(E)$. The canonical choice of solution is of course the least one.

The closure analysis of Bondorf and Jørgensen [1993] can be understood as adding two constraints to each $R(E_1\ @_i\ E_2, \lambda^l x.E')$ such that in effect the inclusions $\mathsf{var}(E_2) \subseteq [\![\nu^l]\!]$ and $\mathsf{var}(E) \subseteq [\![@_i]\!]$ are changed to equalities. Thus, their closure analysis computes more approximate information than ours. In return, their analysis can be computed in almost-linear time, using an other formulation of the problem [Bondorf and Jørgensen 1993], whereas the fastest known algorithm for computing the least solution of $C(E)$ uses transitive closure (see Palsberg and Schwartzbach [1992a; 1994b]).

3. EQUIVALENCE

We now prove that the three closure analyses defined in Section 2 are equivalent (when applied to λ-terms where all labels are distinct). We will use the standard

terminology that μ is a *prefixed point* of $m(E)$ if $m(E)\mu \sqsubseteq \mu$.

LEMMA 3.1. *If μ is a prefixed point of $m(E)$, then so is it of $m(E')$ for every subterm E' of E.*

PROOF. By induction on the structure of E. □

LEMMA 3.2. *$C(E)$ has least solution fix$(m(E))$.*

PROOF. We prove a stronger property: the solutions of $C(E)$ are exactly the prefixed points of $m(E)$. There are two inclusions to be considered.

First, we prove that every solution of $C(E)$ is a prefixed point of $m(E)$. We proceed by induction on the structure of E. In the base case, consider x^l. Clearly, every μ is a prefixed point of $m(x^l)$. In the induction step, consider first $\lambda^l x.E$. Suppose μ is a solution of $C(\lambda^l x.E)$. Then μ is also a solution of $C(E)$, so by the induction hypothesis, μ is a prefixed point of $m(E)$. Hence, we get $m(\lambda^l x.E)\mu = (m(E)\mu) \sqcup \langle [\![\lambda^l]\!] \mapsto \{l\}\rangle \sqsubseteq \mu \sqcup \langle [\![\lambda^l]\!] \mapsto \{l\}\rangle = \mu$, by using the definition of m, that μ is a prefixed point of $m(E)$, and that since $C(\lambda^l x.E)$ has solution μ, $\{l\} \subseteq \mu([\![\lambda^l]\!])$.

Consider then $E_1 \ @_i \ E_2$. Suppose μ is a solution of $C(E_1 \ @_i \ E_2)$. Then μ is also a solution of $C(E_1)$ and $C(E_2)$, so by the induction hypothesis, μ is a prefixed point of $m(E_1)$ and $m(E_2)$. Hence, we get $m(E_1 \ @_i \ E_2)\mu = \mu$, by using the definition of m, that μ is a prefixed point of $m(E_1)$ and $m(E_2)$, and that $C(E_1 \ @_i \ E_2)$ has solution μ.

Second, we prove that every prefixed point of $m(E)$ is a solution of $C(E)$. We proceed by induction on the structure of E. In the base case, consider x^l. Clearly, every μ is a solution of $C(x^l)$. In the induction step, consider first $\lambda^l x.E'$. Suppose μ is a prefixed point of $m(\lambda^l x.E')$. Then, by Lemma 3.1, μ is also a prefixed point of $m(E')$. By the induction hypothesis, μ is a solution of $C(E')$. Thus, we need to prove that μ satisfies $\{l\} \subseteq [\![\lambda^l]\!]$ and for every $E_1 \ @_i \ E_2$ in E', $R(E_1 \ @_i \ E_2, \lambda^l x.E')$. For the first of these, use that μ is a prefixed point of $m(\lambda^l x.E')$ to get $\mu \sqsupseteq m(\lambda^l x.E')\mu = (m(E')\mu) \sqcup \langle [\![\lambda^l]\!] \mapsto \{l\}\rangle \sqsupseteq \langle [\![\lambda^l]\!] \mapsto \{l\}\rangle$, from which the result follows. For the second one, consider $E_1 \ @_i \ E_2$ in E'. By Lemma 3.1, μ is also a prefixed point of $m(E_1 \ @_i \ E_2)$. Using the assumption that we get $\mu \sqsupseteq m(E_1 \ @_i \ E_2)\mu \sqsupseteq \bigsqcup_{l \in \mu(\mathsf{var}(E_1))} (\langle [\![\nu^l]\!] \mapsto \mu(\mathsf{var}(E_2))\rangle \sqcup \langle [\![@_i]\!] \mapsto \mu(\mathsf{var}(\mathit{body}(l)))\rangle)$, from which the result follows.

Consider then $E_1 \ @_i \ E_2$. Suppose μ is a prefixed point of $m(E_1 \ @_i \ E_2)$. Then, by Lemma 3.1, μ is also a prefixed point of both $m(E_1)$ and $m(E_2)$. By the induction hypothesis, μ is a solution of both $C(E_1)$ and $C(E_2)$. Thus, we need to prove that for every $\lambda^l x.E'$ in $E_1 \ @_i \ E_2$, μ satisfies $R(E_1 \ @_i \ E_2, \lambda^l x.E')$. From μ being a prefixed point of $m(E_1 \ @_i \ E_2)$, we get $\mu \sqsupseteq m(E_1 \ @_i \ E_2)\mu \sqsupseteq \bigsqcup_{l \in \mu(\mathsf{var}(E_1))} (\langle [\![\nu^l]\!] \mapsto \mu(\mathsf{var}(E_2))\rangle \sqcup \langle [\![@_i]\!] \mapsto \mu(\mathsf{var}(\mathit{body}(l)))\rangle)$, from which the result follows. □

LEMMA 3.3. *$C(E)$ has least solution fst$(B(E))$.*

PROOF. Similar to the proof of Lemma 3.2. □

THEOREM 3.4. *The three closure analyses defined in Section 2 are equivalent.*

PROOF. Combine Lemmas 3.2 and 3.3. □

4. CORRECTNESS

We now prove that the three closure analyses defined in Section 2 are correct. The key is to define an entailment relation $A \rightsquigarrow A'$ (Definition 4.1) meaning that all constraints in the constraint system A' can be logically derived from those in A. A central result (Theorem 4.10) is that if $E_X \to E_Y$, then $C(E_X) \rightsquigarrow C(E_Y)$. This theorem is proved without at all considering solutions of the involved constraint systems.

Definition 4.1. If A is a constraint system, and H is a Horn clause, then the judgment $A \vdash H$ ("A entails H") holds if it is derivable using the following five rules:

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

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

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

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

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

If A, A' are constraint systems, then $A \rightsquigarrow A'$ if and only if $\forall H \in A' : A \vdash H$.

LEMMA 4.2. *$\rightsquigarrow$ is reflexive, transitive, and solution-preserving. If $A \supseteq A'$, then $A \rightsquigarrow A'$.*

PROOF. The last property is immediate using Discharge. Reflexivity of $\rightsquigarrow$ is a consequence of the last property. For transitivity of $\rightsquigarrow$, suppose $A \rightsquigarrow A'$ and $A' \rightsquigarrow A''$. The statement "if $A' \vdash H$ then $A \vdash H$" can be proved by induction on the structure of the proof of $A' \vdash H$. To prove $A \rightsquigarrow A''$, suppose then that $H \in A''$. From $A' \rightsquigarrow A''$ we get $A' \vdash H$, and from the above statement we finally get $A \vdash H$. To prove that $\rightsquigarrow$ is solution-preserving, suppose $A \rightsquigarrow A'$ and that A has solution L. We need to prove that for every $H \in A'$, H has solution L. This can be proved by induction on the structure of the proof of $A \vdash H$. □

The following lemmas are structured such that Modus Ponens is only used in the proof of Lemma 4.3, and Weakening is only used in the proof of Lemma 4.6.

To aid intuition we can informally read $A \vdash \mathsf{var}(E) \subseteq \mathsf{var}(E')$ as "under the assumption A, the λ-term E has smaller flow information than the λ-term E'."

The next lemma states that two specific constraints can be derived from the constraint system for a redex. Informally, the first constraint says that the argument has smaller flow information than the bound variable, and the second constraint says that the body of the abstraction has smaller flow information than the whole redex.

LEMMA 4.3. *If* $A \rightsquigarrow C((\lambda^l x.E)\ \mathtt{@}_i\ E_2)$, *then* $A \vdash \mathsf{var}(E_2) \subseteq [\![\nu^l]\!]$ *and* $A \vdash \mathsf{var}(E) \subseteq [\![\mathtt{@}_i]\!]$.

PROOF. We have $A \vdash \{l\} \subseteq [\![\lambda^l]\!]$ and $A \rightsquigarrow R((\lambda^l x.E)\ \mathtt{@}_i\ E_2, \lambda^l x.E)$. The result then follows from $\mathsf{var}(\lambda^l x.E) = [\![\lambda^l]\!]$ and Modus Ponens. □

The next lemma is a substitution lemma. Informally, it states that a λ-term gets smaller flow information if a subterm gets substituted by one with smaller flow information.

LEMMA 4.4. *If* $A \vdash \mathsf{var}(U) \subseteq [\![\nu^l]\!]$, *then* $A \vdash \mathsf{var}(E[U/x^l]) \subseteq \mathsf{var}(E)$.

PROOF. By induction on the structure of E, using Reflexivity repeatedly. □

Informally, the next lemma states that beta-reduction creates λ-terms with smaller flow information.

LEMMA 4.5. *If* $A \rightsquigarrow C(E_X)$ *and* $E_X \to E_Y$, *then* $A \vdash \mathsf{var}(E_Y) \subseteq \mathsf{var}(E_X)$.

PROOF. We proceed by induction on the structure of E_X. In the base case, consider x^l. The conclusion is immediate since x^l is in normal form.

In the induction step, consider first $\lambda^l x.E$. Suppose $E \to E'$. Notice that $\mathsf{var}(\lambda^l x.E) = \mathsf{var}(\lambda^l x.E') = [\![\lambda^l]\!]$. Using Reflexivity we get $A \vdash [\![\lambda^l]\!] \subseteq [\![\lambda^l]\!]$.

Consider finally $E_1\ \mathtt{@}_i\ E_2$. There are three cases. Suppose $E_1 \to E_1'$. Notice that $\mathsf{var}(E_1\ \mathtt{@}_i\ E_2) = \mathsf{var}(E_1'\ \mathtt{@}_i\ E_2) = [\![\mathtt{@}_i]\!]$. Using Reflexivity we get $A \vdash [\![\mathtt{@}_i]\!] \subseteq [\![\mathtt{@}_i]\!]$.

Suppose then that $E_2 \to E_2'$. Notice that $\mathsf{var}(E_1\ \mathtt{@}_i\ E_2) = \mathsf{var}(E_1'\ \mathtt{@}_i\ E_2) = [\![\mathtt{@}_i]\!]$. Using Reflexivity we get $A \vdash [\![\mathtt{@}_i]\!] \subseteq [\![\mathtt{@}_i]\!]$.

Suppose then that $E_1 = \lambda^l x.E$ and that $E_1\ \mathtt{@}_i\ E_2 \to E[E_2/x^l]$. From Lemma 4.3 we get $A \vdash \mathsf{var}(E_2) \subseteq [\![\nu^l]\!]$ and $A \vdash \mathsf{var}(E) \subseteq [\![\mathtt{@}_i]\!]$. From the former of these and Lemma 4.4 we get $A \vdash \mathsf{var}(E[E_2/x^l]) \subseteq \mathsf{var}(E)$. Using Transitivity we can finally conclude that $A \vdash \mathsf{var}(E[E_2/x^l]) \subseteq [\![\mathtt{@}_i]\!]$. □

Informally, the next lemma states that entailment is robust under beta-reduction and substitution.

LEMMA 4.6. *Suppose* $A \rightsquigarrow R(E_1\ \mathtt{@}_i\ E_2, \lambda^l x.E_3) \cup C(E_1) \cup C(E_2) \cup C(E_3)$. *If* $E_j = E_j'$ *or* $E_j \to E_j'$ *or* $E_j' = E_j[U_j/x_j^{l_j}]$ *where* $A \vdash \mathsf{var}(U_j) \subseteq \mathsf{var}(x_j^{l_j})$ *for* $j \in 1..3$, *then* $A \rightsquigarrow R(E_1'\ \mathtt{@}_i\ E_2', \lambda^l x.E_3')$.

PROOF. For $j \in 1..3$, we get $A \vdash \mathsf{var}(E_j') \subseteq \mathsf{var}(E_j)$ from either Reflexivity, Lemma 4.5, or Lemma 4.4. The result then follows using Weakening. □

The following definition is needed for stating and proving Lemma 4.9.

Definition 4.7. The set $W(E, E')$ is the union of the following sets of constraints.

—$C(E) \cup C(E')$.

—For every $E_1\ \mathtt{@}_i\ E_2$ in E and for every $\lambda^l x.E_3$ in E', the set $R(E_1\ \mathtt{@}_i\ E_2, \lambda^l x.E_3)$.

—For every $E_1\ \mathtt{@}_i\ E_2$ in E' and for every $\lambda^l x.E_3$ in E, the set $R(E_1\ \mathtt{@}_i\ E_2, \lambda^l x.E_3)$.

LEMMA 4.8. $W(E_1, E_2) \subseteq C(E_1\ \mathtt{@}_i\ E_2)$. *Moreover, if* E_1' *is a subterm of* E_1, *then* $W(E_1', E_2) \subseteq W(E_1, E_2)$.

PROOF. Immediate. □

The next lemma is a substitution lemma. Like Lemma 4.6, it states that entailment is robust under substitution.

LEMMA 4.9. *If* $A \rightsquigarrow W(E,U)$ *and* $A \vdash \mathsf{var}(U) \subseteq [\![\nu^l]\!]$, *then* $A \rightsquigarrow C(E[U/x^l])$.

PROOF. Let ρ denote the substitution $[U/x^l]$. We proceed by induction on the structure of E. In the base case, consider $E = y^{l'}$. If $x^l \equiv y^{l'}$, then $E\rho = U$ so the result follows from $A \rightsquigarrow W(E,U)$ and Lemma 4.2. If $x^l \not\equiv y^{l'}$, then $E\rho = E$ so again the result follows from $A \rightsquigarrow W(E,U)$ and Lemma 4.2.

In the induction step, consider first $E = \lambda^{l'} y.E'$. If $x^l \equiv y^{l'}$, then $E\rho = E$ so also in this case the result follows from $A \rightsquigarrow W(E,U)$ and Lemma 4.2. If $x^l \not\equiv y^{l'}$, $E\rho = \lambda^{l'} y.(E'\rho)$. By the induction hypothesis, $A \rightsquigarrow C(E'\rho)$. Thus, we need to show $A \vdash \{l'\} \subseteq [\![\lambda^{l'}]\!]$ and for every $E_1 \ \mathbb{C}_i \ E_2$ in $E'\rho$, $A \rightsquigarrow R(E_1 \ \mathbb{C}_i \ E_2, \lambda^{l'} y.(E'\rho))$. The first follows from $A \rightsquigarrow C(\lambda^{l'} y.E')$. For the second, consider any $E_1 \ \mathbb{C}_i \ E_2$ in $E'\rho$. Notice that either $E_1 \ \mathbb{C}_i \ E_2$ is a subterm of E', or $E_1 \ \mathbb{C}_i \ E_2 = (E'_1 \ \mathbb{C}_i \ E'_2)\rho = (E'_1\rho) \ \mathbb{C}_i \ (E'_2\rho)$ where $E'_1 \ \mathbb{C}_i \ E'_2$ is a subterm of E', or $E_1 \ \mathbb{C}_i \ E_2$ is a subterm of U. In each case the result follows from $A \rightsquigarrow W(E,U)$ and Lemma 4.6.

Consider finally $E = E_1 \ \mathbb{C}_i \ E_2$. Notice that $(E_1 \ \mathbb{C}_i \ E_2)\rho = (E_1\rho) \ \mathbb{C}_i \ (E_2\rho)$. By the induction hypothesis, $A \rightsquigarrow C(E_1\rho) \cup C(E_2\rho)$. Thus, we need to show that for every $\lambda^{l'} y.E'$ in $(E_1 \ \mathbb{C}_i \ E_2)\rho$, $A \rightsquigarrow R((E_1 \ \mathbb{C}_i \ E_2)\rho, \lambda^{l'} y.E')$. Consider any $\lambda^{l'} y.E'$ in $(E_1 \ \mathbb{C}_i \ E_2)\rho$. Notice that either $\lambda^{l'} y.E'$ is a subterm of $E_1 \ \mathbb{C}_i \ E_2$, or $\lambda^{l'} y.E' = \lambda^{l'} y.(E'\rho)$ where $\lambda^{l'} y.E'$ is a subterm of $E_1 \ \mathbb{C}_i \ E_2$, or $\lambda^{l'} y.E'$ is a subterm of U. In each case the result follows from $A \rightsquigarrow W(E,U)$ and Lemma 4.6. □

We can now prove that if we beta-reduce E_X to E_Y, then the constraint system for E_X entails the constraint system for E_Y.

THEOREM 4.10. *If* $E_X \to E_Y$, *then* $C(E_X) \rightsquigarrow C(E_Y)$.

PROOF. We proceed by induction on the structure of E_X. In the base case of x^l, the conclusion is immediate since x^l is in normal form.

In the induction step, consider first $\lambda^l x.E$. Suppose $E \to E'$. By the induction hypothesis, $C(E) \rightsquigarrow C(E')$, so also $C(\lambda^l x.E) \rightsquigarrow C(E')$. Thus, we need to show $C(\lambda^l x.E) \vdash \{l\} \subseteq [\![\lambda^l]\!]$ and for every $E_1 \ \mathbb{C}_i \ E_2$ in $\lambda^l x.E'$, $C(\lambda^l x.E) \rightsquigarrow R(E_1 \ \mathbb{C}_i \ E_2, \lambda^l x.E')$. The first follows using Discharge. For the second, there are four cases. Notice that by Discharge we have $C(\lambda^l x.E) \rightsquigarrow R(E'_1 \ \mathbb{C}_i \ E'_2, \lambda^l x.E)$ for every $E'_1 \ \mathbb{C}_i \ E'_2$ in $\lambda^l x.E$. In the first case, suppose $E_1 \ \mathbb{C}_i \ E_2$ is also a subterm of $\lambda^l x.E$. The result then follows from Lemma 4.6. In the second case, consider a subterm $E'_1 \ \mathbb{C}_i \ E_2$ of $\lambda^l x.E$ such that $E'_1 \to E_1$. Again, the result follows from Lemma 4.6. In the third case, consider a subterm $E_1 \ \mathbb{C}_i \ E'_2$ of $\lambda^l x.E$ such that $E'_2 \to E_2$. Yet again, the result follows from Lemma 4.6. In the fourth case, consider a subterm $E'_1 \ \mathbb{C}_i \ E'_2$ of $\lambda^l x.E$ such that $E_1 \ \mathbb{C}_i \ E_2 = (E'_1 \ \mathbb{C}_i \ E'_2)[E_S/y^{l'}]$. The substitution arises because of the contraction of a redex. From Lemma 4.3 we get $C(\lambda^l x.E) \vdash \mathsf{var}(E_S) \subseteq [\![\nu^{l'}]\!]$. The result then follows from Lemma 4.6.

Consider finally $E_1 \ \mathbb{C}_i \ E_2$. For every $\lambda^l x.E$ in $E_1 \ \mathbb{C}_i \ E_2$, we have $C(E_1 \ \mathbb{C}_i \ E_2) \rightsquigarrow R(E_1 \ \mathbb{C}_i \ E_2, \lambda^l x.E)$. There are three cases.

Suppose that $E_1 \to E'_1$. By the induction hypothesis, $C(E_1) \rightsquigarrow C(E'_1)$, so also $C(E_1 \ \mathbb{C}_i \ E_2) \rightsquigarrow C(E'_1)$. Thus we need to show that for every $\lambda^l x.E'$ in $E'_1 \ \mathbb{C}_i \ E_2$, $C(E_1 \ \mathbb{C}_i \ E_2) \rightsquigarrow R(E'_1 \ \mathbb{C}_i \ E_2, \lambda^l x.E')$. There are three cases. In the first case, suppose $\lambda^l x.E'$ is a subterm of $E_1 \ \mathbb{C}_i \ E_2$. The result then follows from Lemma 4.6.

In the second case, consider a subterm $\lambda^l x.E$ of $E_1 \ @_i \ E_2$ such that $E \to E'$. Again, the result follows from Lemma 4.6. In the third case, consider a subterm $\lambda^l x.E$ of $E_1 \ @_i \ E_2$ such that $\lambda^l x.E' = \lambda^l x.(E[E_S/y^{l'}])$. The substitution arises because of the contraction of a redex. From Lemma 4.3 we see $C(E_1 \ @_i \ E_2) \vdash \mathsf{var}(E_S) \subseteq [\![\nu^{l'}]\!]$. The result then follows from Lemma 4.6.

Suppose then that $E_2 \to E_2'$. The proof in this case is similar to the case of $E_1 \to E_1'$ so we omit the details.

Suppose then that $E_1 = \lambda^l x.E$ and that $E_1 \ @_i \ E_2 \to E[E_2/x^l]$. From Lemma 4.3 we see $C(E_1 \ @_i \ E_2) \vdash \mathsf{var}(E_2) \subseteq [\![\nu^l]\!]$. From Lemma 4.8 we see that $W(E, E_2) \subseteq C(E_1 \ @_i \ E_2)$. The result then follows from Lemma 4.9. □

THEOREM 4.11. *The three closure analyses defined in Section 2 are correct.*

PROOF. From Theorem 3.4 we see that the three analyses are equivalent when applied to λ-terms where all labels are distinct. Thus, it is sufficient to prove that the one defined using a constraint system is correct. The proof has two steps.

In Step 1, use Lemmas 4.3, 4.4, and 4.5 to prove that if $A \rightsquigarrow C(E_X)$ and $E_X \to E_Y$, then both of the following properties hold:

—If E_Y contains $\lambda^l y.E$, then E_X contains $\lambda^l z.E'$ such that $A \vdash \mathsf{var}(E) \subseteq \mathsf{var}(E')$.

—If E_Y contains $E_1 \ @_i \ E_2$, then E_X contains $E_1' \ @_i \ E_2'$ such that $A \vdash \mathsf{var}(E_1) \subseteq \mathsf{var}(E_1')$ and $A \vdash \mathsf{var}(E_2) \subseteq \mathsf{var}(E_2')$.

In Step 2, suppose $C(E_X)$ has solution L, and suppose $E_X \to^* E_Y$. We will prove $T_L(E_X, E_Y)$ by induction on the length of $E_X \to^* E_Y$.

In the base case, $T_L(E_X, E_X)$ is immediate. In the induction step, suppose $E_X \to E_Z \to^n E_Y$. By Theorem 4.10, $C(E_X) \rightsquigarrow C(E_Z)$. By Lemma 4.2, $C(E_Z)$ has solution L. By the induction hypothesis, $T_L(E_Z, E_Y)$. To prove $T_L(E_X, E_Y)$, there are four cases to be considered.

First suppose $E_Y = \lambda^l x.E$. From $T_L(E_Z, E_Y)$ we get $\{l\} \subseteq L(\mathsf{var}(E_Z))$. From Lemma 4.5 we get $C(E_X) \vdash \mathsf{var}(E_Z) \subseteq \mathsf{var}(E_X)$. Finally, the result follows by using that $C(E_X)$ has solution L.

Then suppose E_Y contains $\lambda^{l'} y.(\lambda^l x.E)$. From $T_L(E_Z, E_Y)$ we get that E_Z contains $\lambda^{l'} z.E'$ such that $\{l\} \subseteq L(\mathsf{var}(E'))$. From Step 1 of this proof, we get that E_X contains $\lambda^{l'} w.E''$ such that $C(E_X) \vdash \mathsf{var}(E') \subseteq \mathsf{var}(E'')$. Finally, the result follows by using that $C(E_X)$ has solution L.

In the last two cases, suppose E_Y contains either $(\lambda^l x.E) \ @_i \ E_2$ or $E_1 \ @_i \ (\lambda^l x.E)$, respectively. Both cases are similar to the second one, so we omit the details. □

Finally, we prove our subject-reduction result.

THEOREM 4.12. *If $C(E)$ has solution L and $E \to E'$, then $C(E')$ has solution L.*

PROOF. Immediate from Theorem 4.10 and Lemma 4.2. □

ACKNOWLEDGMENTS

The author thanks Torben Amtoft, Nils Klarlund, and the anonymous referees for helpful comments on a draft of the article.

REFERENCES

Agesen, O., Palsberg, J., and Schwartzbach, M. I. 1993. Type inference of Self: Analysis of objects with dynamic and multiple inheritance. In *Proceedings of ECOOP'93, 7th European Conference on Object-Oriented Programming*. Lecture Notes in Computer Science, vol. 707. Springer-Verlag, New York, 247–267.

Ayers, A. 1992. Efficient closure analysis with reachability. In *Proceedings of WSA'92, Analyse Statique*. IRISA, Rennes, France, 126–134.

Barendregt, H. P. 1981. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, Amsterdam.

Bondorf, A. 1993. *Similix 5.0 Manual*. DIKU, University of Copenhagen, Denmark. Included in Similix 5.0 distribution.

Bondorf, A. 1991. Automatic autoprojection of higher order recursive equations. *Sci. Comput. Program. 17,* 1–3 (Dec.), 3–34.

Bondorf, A. and Danvy, O. 1991. Automatic autoprojection of recursive equations with global variables and abstract data types. *Sci. Comput. Program. 16*, 151–195.

Bondorf, A. and Jørgensen, J. 1993. Efficient analyses for realistic off-line partial evaluation. *J. Functional Program. 3*, 3, 315–346.

Consel, C. 1990. Binding time analysis for higher order untyped functional languages. In *Proceedings of the ACM Conference on Lisp and Functional Programming*. ACM, New York, 264–272.

Giannini, P. and Rocca, S. R. D. 1988. Characterization of typings in polymorphic type discipline. In *Proceedings of LICS'88, 3rd Annual Symposium on Logic in Computer Science*. IEEE, New York, 61–70.

Heintze, N. 1994. Set-based analysis of ML programs. In *Proceedings of the ACM Conference on LISP and Functional Programming*. ACM, New York, 306–317.

Heintze, N. 1992. Set based program analysis. Ph. D. thesis, CMU–CS–92–201, Carnegie Mellon University, Pittsburgh, Pa.

Jones, N. D. 1981. Flow analysis of lambda expressions. In *Proceedings of the 8th Colloquium on Automata, Languages, and Programming*. Lecture Notes in Computer Science, vol. 115. Springer-Verlag, New York, 114–128.

Palsberg, J. 1993. Correctness of binding-time analysis. *J. Functional Program. 3*, 3, 347–363.

Palsberg, J. and Schwartzbach, M. I. 1994a. Binding-time analysis: Abstract interpretation versus type inference. In *Proceedings of ICCL'94, 5th IEEE International Conference on Computer Languages*. IEEE, New York, 289–298.

Palsberg, J. and Schwartzbach, M. I. 1994b. *Object-Oriented Type Systems*. John Wiley and Sons, New York.

Palsberg, J. and Schwartzbach, M. I. 1992a. Safety analysis versus type inference. *Inf. Comput.* To be published.

Palsberg, J. and Schwartzbach, M. I. 1992b. Safety analysis versus type inference for partial types. *Inf. Process. Lett. 43*, 175–180.

Palsberg, J. and Schwartzbach, M. I. 1991. Object-oriented type inference. In *Proceedings of OOPSLA'91, ACM SIGPLAN 6th Annual Conference on Object-Oriented Programming Systems, Languages and Applications*. ACM, New York, 146–161.

Sestoft, P. 1991. Analysis and efficient implementation of functional programs. Ph. D. thesis, DIKU, University of Copenhagen.

Sestoft, P. 1989. Replacing function parameters by global variables. M.S. thesis, DIKU, University of Copenhagen.

Shivers, O. 1991a. Control-flow analysis of higher-order languages. Ph. D. thesis, CMU–CS–91–145, Carnegie Mellon University, Pittsburgh, Pa.

Shivers, O. 1991b. Data-flow analysis and type recovery in Scheme. In *Topics in Advanced Language Implementation*, P. Lee, Ed. MIT Press, Cambridge, Mass., 47–87.

Ungar, D. and Smith, R. B. 1987. SELF: The power of simplicity. In *Proceedings of OOPSLA'87, Object-Oriented Programming Systems, Languages and Applications*. ACM, New

York, 227–241. Also published in Lisp and Symbolic Computation 4(3), Kluwer Acadamic Publishers, June 1991.

Wand, M. and Steckler, P. 1994. Selective and lightweight closure conversion. In *Proceedings of POPL'94, 21st Annual Symposium on Principles of Programming Languages*. ACM, New York, 434–445.

Received May 1994; revised October 1994; accepted October 1994