

Comparing Flow-based Binding-time Analyses

Jens Palsberg

Computer Science Department, Aarhus University
Ny Munkegade, DK-8000 Aarhus C, Denmark
`palsberg@daimi.aau.dk`

Abstract. Binding-time analyses based on flow analysis have been presented by Bondorf, Consel, Bondorf and Jørgensen, and Schwartzbach and the present author. The analyses are formulated in radically different ways, making comparison non-trivial.

In this paper we demonstrate how to compare such analyses. We prove that the first and the fourth analyses can be specified by constraint systems of a particular form, enabling direct comparison. As corollaries, we get that Bondorf's analysis is more conservative than ours, that both analyses can be performed in cubic time, and that the core of Bondorf's analysis is correct. Our comparison is of analyses that apply to the pure λ -calculus.

1 Our Results

We present a constraint-based technique for comparing flow-based binding-time analyses. Binding-time analysis is used in most partial evaluators. The analysis divides the computations in a source program into “static” computations (to be performed by the partial evaluator) and “dynamic” computations (to be performed in the partially evaluated program).

Several binding-time analyses of untyped higher-order languages such as Scheme use a flow analysis to obtain information about higher-order control flow. Such analyses have been presented by Bondorf [2], Consel [4], Bondorf and Jørgensen [3], and Schwartzbach and the present author [10]. The analyses are formulated in radically different ways, as follows:

- **Bondorf.** Flow analysis is performed first, by abstract interpretation; and binding-time analysis is then performed by another abstract interpretation, using the computed flow information.
- **Consel.** Flow and binding-time information are computed by a single abstract interpretation.
- **Bondorf and Jørgensen.** Flow analysis is performed first, by solving a constraint system; binding-time analysis is then performed by solving another constraint system.

- **Palsberg and Schwartzbach (PS).** Binding-time analysis is performed by searching for an output which satisfies a certain predicate. The predicate is formulated using constraint systems where variables range over a combination of flow and binding-time information.

Comparison of these analyses is non-trivial and until now an open problem.

In this paper we demonstrate how to compare such analyses. We concentrate on comparing the analysis of Bondorf with the analysis of PS. To enable comparison, we restrict Bondorf’s analysis to the pure λ -calculus. We prove that both analyses can be specified by so-called SF-systems (SF = “Separate Flow”). This leads to the direct comparison:

$$\text{Bondorf} \iff \text{Bondorf as SF-system} \xrightarrow{\text{direct compar.}} \text{PS as SF-system} \iff \text{PS}$$

An SF-system is a form of constraint system that uses both variables ranging over flow information and variables ranging over binding-time information. The two SF-systems for the analyses of Bondorf and PS are both derived from the program to be analyzed and both contain a subsystem that specifies flow analysis. As corollaries of the two equivalence proofs, we get:

- **Comparison.** Bondorf’s analysis is more conservative than that of Schwartzbach and me, because for all programs the latter SF-system is a subset of the former;
- **Efficiency.** Both analyses can be performed in cubic time, because every SF-system can be solved in cubic time; and
- **Correctness.** The core of Bondorf’s analysis is correct, because the SF-system can be connected to a known correctness result for binding-time analysis.

These results demonstrate the benefits of formulating flow-based binding-time analyses as SF-systems.

2 Example

We now illustrate the idea of flow-based binding-time analysis, the notion of an SF-system, and the similarities and differences between the binding-time analysis of Bondorf and that of PS.

Consider the λ -term $(\lambda x.xx)(y)$. Throughout we assume that free variables, in this case y , correspond to dynamic information. The task of a binding-time analysis is to assign either **Stat** (static) or **Dyn** (dynamic) to each subterm. This information can then be used to annotate the λ -term.

Following Bondorf [2] and others, we label all abstractions and applications. Variables will also 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.

$$(\lambda^1 x.x^1 @_2 x^1) @_3 y^4$$

For this particular λ -term, there are two possible ways of annotating consistently:

$$(\lambda^1 x.x^1 \underline{\mathbb{O}}_2 x^1) \mathbb{O}_3 y^4 \quad (1)$$

$$(\underline{\lambda^1} x.x^1 \underline{\mathbb{O}}_2 x^1) \underline{\mathbb{O}}_3 y^4 \quad (2)$$

Here, underlining means “dynamic” and no underlining means “static”. Annotated λ -terms are called 2-level λ -terms. Consistency means that no static computation can depend on the result of a dynamic computation [7].

Notice that only abstraction and application symbols can be annotated. We do not need to annotate variables because a free variable is dynamic and the binding-time of a bound variable is the same as that of the λ that binds it.

Bondorf’s binding-time analysis yields information which when used for annotation leads to (2). The binding-time analysis of PS leads to (1). Thus, in this particular case, Bondorf’s analysis is more conservative. One of our theorems says that Bondorf’s analysis always leads to the same or more underlinings compared the analysis of PS.

An SF-system is a form of constraint system that uses both variables ranging over flow information and variables ranging over binding-time information. Our equivalence theorems say that both of the analyses of Bondorf and PS can be formulated as SF-systems.

The two SF-systems for the analyses of Bondorf and PS are both derived from the program to be analyzed and both contain a subsystem that specifies flow analysis. One of our theorems says that for every λ -term, the SF-systems for the analyses of Bondorf and PS have identical subsystems for flow analysis. In the case of the above λ -term, this subsystem is as follows.

$$\begin{array}{l} \text{From } \lambda^1 \quad \{1\} \subseteq \llbracket \lambda^1 \rrbracket \\ \text{From } \mathbb{O}_2 \text{ and } \lambda^1 \quad \begin{cases} \{1\} \subseteq \llbracket \nu^1 \rrbracket \Rightarrow \llbracket \nu^1 \rrbracket \subseteq \llbracket \nu^1 \rrbracket \\ \{1\} \subseteq \llbracket \nu^1 \rrbracket \Rightarrow \llbracket \mathbb{O}_2 \rrbracket \subseteq \llbracket \mathbb{O}_2 \rrbracket \end{cases} \\ \text{From } \mathbb{O}_3 \text{ and } \lambda^1 \quad \begin{cases} \{1\} \subseteq \llbracket \lambda^1 \rrbracket \Rightarrow \llbracket \nu^4 \rrbracket \subseteq \llbracket \nu^1 \rrbracket \\ \{1\} \subseteq \llbracket \lambda^1 \rrbracket \Rightarrow \llbracket \mathbb{O}_2 \rrbracket \subseteq \llbracket \mathbb{O}_3 \rrbracket \end{cases} \end{array}$$

Symbols of the forms $\llbracket \nu^l \rrbracket$, $\llbracket \lambda^l \rrbracket$, and $\llbracket \mathbb{O}_i \rrbracket$ are meta-variables ranging over flow information, that is, sets of labels. They relate to variables with label l , abstractions with label l , and applications with label i , respectively.

To the left of the constraints, we have indicated from where they arise. The first constraint says that an abstraction may evaluate to an abstraction with the same label. The rest of the constraints comes in pairs. For each application point \mathbb{O}_i and each abstraction with label l there are two constraints of the form:

$$\begin{array}{l} \{l\} \subseteq \text{“meta-var. for operator of } \mathbb{O}_i \text{”} \Rightarrow \text{“meta-var. for operand of } \mathbb{O}_i \text{”} \subseteq \llbracket \nu^l \rrbracket \\ \{l\} \subseteq \text{“meta-var. for operator of } \mathbb{O}_i \text{”} \Rightarrow \text{“meta-var. for body of abst.”} \subseteq \llbracket \mathbb{O}_i \rrbracket \end{array}$$

Such pairs of constraints can be read as:

- **The first constraint.** If the function part of \mathcal{C}_i evaluates to an abstraction with label l , then the bound variable of that abstraction may be substituted with anything to which the argument part of \mathcal{C}_i can evaluate.
- **The second constraint.** If the function part of \mathcal{C}_i evaluates to an abstraction with label l , then anything to which the body of the abstraction may evaluate is also a possible result of evaluating the whole application \mathcal{C}_i .

In a solution of the constraint system, meta-variables are assigned flow information. The minimal solution of the above constraint system is the mapping L where:

$$\begin{aligned} L[\lambda^1] &= \{1\} \\ L[\nu^1] &= L[\nu^4] = L[\mathcal{C}_2] = L[\mathcal{C}_3] = \emptyset \end{aligned}$$

This L says that the only subterm that can evaluate to an abstraction is the abstraction with label 1.

Although the SF-systems for the analyses of Bondorf and PS have the same subsystem for specifying flow analysis, they are not the same. One of our theorems says that for all λ -terms, the SF-system for the analysis of Bondorf is a superset of the SF-system for the analysis of PS. The SF-system for the PS analysis of the above λ -term contains the following constraints in addition to those already presented.

$$\begin{array}{ll} \text{From } \mathcal{C}_3 & \llbracket \mathcal{C}_3 \rrbracket_b = \text{Dyn} \\ \text{From } y^4 & \llbracket \nu^4 \rrbracket_b = \text{Dyn} \\ \text{From } \lambda^1 & \llbracket \lambda^1 \rrbracket_b = \text{Dyn} \Rightarrow \llbracket \nu^1 \rrbracket_b = \llbracket \mathcal{C}_2 \rrbracket_b = \text{Dyn} \\ \text{From } \mathcal{C}_2 & \llbracket \nu^1 \rrbracket_b = \text{Dyn} \Rightarrow \llbracket \nu^1 \rrbracket_b = \llbracket \mathcal{C}_2 \rrbracket_b = \text{Dyn} \\ \text{From } \mathcal{C}_3 & \llbracket \lambda^1 \rrbracket_b = \text{Dyn} \Rightarrow \llbracket \nu^4 \rrbracket_b = \llbracket \mathcal{C}_3 \rrbracket_b = \text{Dyn} \\ \text{From } \mathcal{C}_2 \text{ and } \lambda^1 & \left\{ \begin{array}{l} \{1\} \subseteq \llbracket \nu^1 \rrbracket \Rightarrow \llbracket \nu^1 \rrbracket_b = \llbracket \nu^1 \rrbracket_b \\ \{1\} \subseteq \llbracket \nu^1 \rrbracket \Rightarrow \llbracket \mathcal{C}_2 \rrbracket_b = \llbracket \mathcal{C}_2 \rrbracket_b \end{array} \right. \\ \text{From } \mathcal{C}_3 \text{ and } \lambda^1 & \left\{ \begin{array}{l} \{1\} \subseteq \llbracket \lambda^1 \rrbracket \Rightarrow \llbracket \nu^4 \rrbracket_b = \llbracket \nu^1 \rrbracket_b \\ \{1\} \subseteq \llbracket \lambda^1 \rrbracket \Rightarrow \llbracket \mathcal{C}_2 \rrbracket_b = \llbracket \mathcal{C}_3 \rrbracket_b \end{array} \right. \end{array}$$

Meta-variables with b as subscript range over binding-time information, that is, the set $\{\text{Stat}, \text{Dyn}\}$, where $\text{Stat} \leq \text{Dyn}$. To the left of the constraints, we have indicated from where they arise. The constraints can be informally read as follows.

- The first constraint says that the partial evaluator must produce a program.
- The second constraint says that the variable y^4 corresponds to dynamic information.
- The third constraint says that if an abstraction gets classified as dynamic, then so should its bound variable and its body.
- The fourth and fifth constraints say that if the function part of an application gets classified as dynamic, then so should the argument part and the whole application.

- The rest of the constraints come in pairs. They involve both variables ranging over flow information and variables ranging over binding-time information. These constraints are similar to the ones used in the subsystem for flow analysis. The key difference is that the binding-times of the actual and the formal parameter should be equal, and so should the binding-times of the body of the abstraction and of the application.

The minimal solution of this SF-system is a pair of mappings (L, M) , where L was presented above and M is as follows.

$$\begin{aligned} M[[\lambda^1]]_b &= \text{Stat} \\ M[[\nu^1]]_b &= M[[\nu^4]]_b = M[[\mathfrak{C}_2]]_b = M[[\mathfrak{C}_3]]_b = \text{Dyn} \end{aligned}$$

The SF-system for Bondorf’s analysis of the above λ -term contains in addition the following two constraints.

$$\begin{aligned} \text{From } \mathfrak{C}_2 \quad [[\nu^1]]_b = \text{Dyn} &\Rightarrow [[\nu^1]]_b = \text{Dyn} \\ \text{From } \mathfrak{C}_3 \quad [[\nu^4]]_b = \text{Dyn} &\Rightarrow [[\lambda^1]]_b = \text{Dyn} \end{aligned}$$

The constraints can be informally read as follows.

- If the argument part of an application gets classified as dynamic, then so should the function part.

The minimal solution of this SF-system is a pair of mappings (L, M') , where L was presented above and M' is as follows.

$$M'[[\lambda^1]]_b = M'[[\nu^1]]_b = M'[[\nu^4]]_b = M'[[\mathfrak{C}_2]]_b = M'[[\mathfrak{C}_3]]_b = \text{Dyn}$$

The minimal solution of an SF-system can be computed in cubic time. Previously, no complexity analysis has been given for the analysis of Bondorf, and the best-known algorithm for the analysis of PS has so far been one with worst-case exponential running time [10].

Our equivalence proofs makes it possible to relate Bondorf’s analysis to a known correctness result for binding-time analysis [7]. We thereby obtain the first proof of correctness for the core of Bondorf’s analysis.

In the following section we recall from [8] a constraint system that specifies flow analysis. In Section 4 we define SF-systems and we present the two SF-systems that are equivalent to the analyses of Bondorf and PS. Finally in Section 5 we recall the original definitions of Bondorf’s and PS’s analyses and we give equivalence proofs that relate them to the SF-systems.

3 Flow Analysis

We now present the flow analysis which is a subsystem of both the SF-system for Bondorf’s analysis and the SF-system for the analysis of PS. Recall the λ -calculus [1].

Definition 1. *The language of λ -terms has an abstract syntax which is defined by the grammar:*

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

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 labels and the application symbols are not part of the concrete syntax.

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

Definition 2. *A meta-variable is of one of the forms $[\nu^l]$, $[\lambda^l]$, and $[@_i]$. The set of all meta-variables is denoted Metavar . A λ -term is assigned a meta-variable 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 , $\text{Lab}(E)$ is the set of labels on abstractions (but not applications) occurring in E . Notice that $\text{Lab}(E)$ is finite. The set $\text{CSet}(E)$ is the powerset of $\text{Lab}(E)$; $\text{CSet}(E)$ with the inclusion ordering is a complete lattice. The set $\text{CMap}(E)$ consists of the total functions from Metavar to $\text{CSet}(E)$. The set $\text{CMap}(E)$ with point-wise ordering, written \leq , is a complete lattice where least upper bound is written \sqcup .

The following flow analysis uses a constraint system. It has been used by Schwartzbach and the present author in [12, 9], and in [8] it was proved equivalent to the flow analysis of Bondorf [2], which in turn is based on Sestoft's [13]. Flow analysis is called closure analysis in some papers, including [2, 8].

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

The constraint system is defined in terms of the program 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

$$\{l\} \subseteq \text{var}(E_1) \Rightarrow \text{var}(E_2) \subseteq [\nu^l]$$

$$\{l\} \subseteq \text{var}(E_1) \Rightarrow \text{var}(E) \subseteq [@_i]$$

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

- For every $\lambda^l x.F$ in E , the singleton constraint set consisting of $\{l\} \subseteq [\lambda^l]$.
- For every $E_1 @_i E_2$ in E and for every $\lambda^l x.F$ in E , the set $R(E_1 @_i E_2, \lambda^l x.F)$.

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

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

4 Flow-based Binding-time Analysis

The output of a binding-time analysis can be presented as an *annotated* version of the analyzed term. In the annotated term, all dynamic abstractions and applications are underlined. The language of annotated terms is usually called a 2-level λ -calculus [6] and is defined as follows.

Definition 3. *The language of 2-level λ -terms is defined by the grammar:*

$$\begin{array}{l}
 W ::= x^l \quad (\text{variable}) \\
 \quad | \lambda^l x.W \quad (\text{static abstraction}) \\
 \quad | W_1 \circledast_i W_2 \quad (\text{static application}) \\
 \quad | \underline{\lambda^l} x.W \quad (\text{dynamic abstraction}) \\
 \quad | W_1 \underline{\circledast}_i W_2 \quad (\text{dynamic application})
 \end{array}$$

The language of 2-level λ -terms is partially ordered by \sqsubseteq as follows. Given 2-level λ -terms W and W' , $W \sqsubseteq W'$ if and only if they are equal except for underlinings and W' has the same and possibly more underlinings than W . For example, $(\lambda^1 x.x^1 \underline{\circledast}_2 x^1) \circledast_3 y^4 \sqsubseteq (\underline{\lambda^1} x.x^1 \underline{\circledast}_2 x^1) \underline{\circledast}_3 y^4$. Notice that \sqsubseteq admits greatest lower bounds for terms that are equal except for underlinings.

The abstract domain for the binding-time analysis of a λ -term E is called $\text{BMap}(E)$ and is defined as follows.

Definition 4. *Let $\text{BVal} = \{\text{Stat}, \text{Dyn}\}$. The set BVal is totally ordered by \leq so that $\text{Stat} \leq \text{Dyn}$. For clarity, let Metavar_b be a copy of Metavar where elements are written with b as subscript. The function var_b maps λ -terms to meta-variables with subscript b . The set BMap consists of the total functions from Metavar_b to BVal . The set BEnv contains each function in BMap when restricted to meta-variables of the form $\llbracket \nu^l \rrbracket_b$. Both BMap and BEnv with point-wise ordering, written \leq , are complete lattices where least upper bound is written \sqcup . The function $\langle V \mapsto S \rangle$ maps the meta-variable V to the value S and maps all other meta-variables to Stat . Finally, we define $\text{upd } V \ S \ L = \langle V \mapsto S \rangle \sqcup L$.*

Given a λ -term E and $M \in \text{BMap}$, we can annotate E by the following function T_M .

$$T_M(x) = x$$

$$T_M(\lambda^l x.E) = \begin{cases} \lambda^l x.T_M(E) & \text{if } M[\llbracket \lambda^l \rrbracket_b] = \text{Stat} \\ \underline{\lambda^l} x.T_M(E) & \text{if } M[\llbracket \lambda^l \rrbracket_b] = \text{Dyn} \end{cases}$$

$$T_M(E_1 \circledast_i E_2) = \begin{cases} T_M(E_1) \circledast_i T_M(E_2) & \text{if } M(\text{var}_b(E_1)) = \text{Stat} \\ T_M(E_1) \underline{\circledast}_i T_M(E_2) & \text{if } M(\text{var}_b(E_1)) = \text{Dyn} \end{cases}$$

Lemma 1. *Let E be a λ -term and let $M, M' \in \text{BMap}$. If $M \leq M'$, then $T_M(E) \sqsubseteq T_{M'}(E)$.*

Proof. Immediate.

We can now define the notion of an SF-system.

Definition 5. An SF-system is a finite set of constraints over two disjoint copies of *Metavar*. The first copy of *Metavar* is denoted Metavar_c and the second copy is denoted Metavar_b . Elements of Metavar_c are written without subscript and elements of Metavar_b are written with b as subscript. A constraint is a conditional clause of the following form:

- The hypotheses are either of the form $\{l\} \subseteq V$ where $V \in \text{Metavar}_c$, or of the form $V_b = \text{Dyn}$ where $V_b \in \text{Metavar}_b$.
- The conclusion is either of the form $P \subseteq P'$ where $P, P' \in \text{CSet}(E) \cup \text{Metavar}_c$ for some E , or of the form $P_b = P'_b$ where $P_b, P'_b \in \text{BVal} \cup \text{Metavar}_b$.

A solution for an SF-system is a pair of mappings (L, M) such that all constraints are satisfied when elements of Metavar_c are mapped to a value by L and elements of Metavar_b are mapped to a value by M . The desired binding-time information is then the mapping M .

Each SF-system has a least solution namely the component-wise greatest lower bound of all solutions. The least solution of an SF-system can be computed in cubic time using a straightforward modification of the algorithm in [12] (see also [11, Chapter 5]).

Given a λ -term E , the following SF-system yields a binding-time analysis of E .

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

- The singleton set consisting of $\text{var}_b(E) = \text{Dyn}$.
- For every free variable x^l of E , the singleton set consisting of $\llbracket \nu^l \rrbracket_b = \text{Dyn}$.
- For every $\lambda^l x.F$ in E , the set consisting of $\llbracket \lambda^l \rrbracket_b = \text{Dyn} \Rightarrow \llbracket \nu^l \rrbracket_b = \text{var}_b(F) = \text{Dyn}$.
- For every $E_1 @_i E_2$ in E , the set consisting of $\text{var}_b(E_1) = \text{Dyn} \Rightarrow \text{var}_b(E_2) = \llbracket @_i \rrbracket_b = \text{Dyn}$.
- For every $E_1 @_i E_2$ in E and for every $\lambda^l x.F$ in E , the set consisting of
 - $\{l\} \subseteq \text{var}(E_1) \Rightarrow \text{var}_b(E_2) = \llbracket \nu^l \rrbracket_b$
 - $\{l\} \subseteq \text{var}(E_1) \Rightarrow \text{var}_b(F) = \llbracket @_i \rrbracket_b$

We can now do binding-time analysis of E by computing a solution of $B(E)$. The canonical choice of solution is of course the least one.

In the next section we will prove that this binding-time analysis is equivalent to that of PS. We will also prove that the analysis of Bondorf is equivalent to the following modified analysis.

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

- For every $E_1 @_i E_2$ in E , the singleton set consisting of $\text{var}_b(E_2) = \text{Dyn} \Rightarrow \text{var}_b(E_1) = \text{Dyn}$.

Fact 1 *Bondorf's analysis is more conservative than the analysis of PS.*

Proof. Clearly, if $B'(E)$ is solvable, then so is $B(E)$. So if (L, M) is the least solution of $B(E)$ and (L', M') is the least solution of $B'(E)$, then $M \leq M'$, and by Lemma 1, $T_M(E) \sqsubseteq T_{M'}(E)$.

5 Equivalence Proofs

5.1 Bondorf's Analysis

The Original Formulation We recall the binding-time analysis of Bondorf [2], 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.

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 . We define $\mu_E^{input} = \langle \text{var}(E) \mapsto \text{Dyn} \rangle$ and we define $\rho_E^{input} = \langle x_1 \mapsto \text{Dyn} \rangle \sqcup \dots \sqcup \langle x_n \mapsto \text{Dyn} \rangle$, where $x_1 \dots x_n$ are the free variables of E .

Bondorf's analysis proceeds by first computing flow information by an abstract interpretation. In a previous paper [8] we proved that Bondorf's flow analysis is equivalent to computing the least solution of the constraint system $C(E)$. So for a λ -term E , suppose that $C(E)$ has least solution L .

We follow Bondorf in using an auxiliary function *raise*, defined as follows.

$$\begin{aligned} \text{raise} &: \text{Metavar} \rightarrow \text{BMap} \rightarrow \text{BEnv} \rightarrow \text{BMap} \times \text{BEnv} \\ \text{raise } k\mu\rho &= (\mu, \rho) \sqcup (\bigsqcup_{l \in L(k)} (\text{upd } \llbracket \lambda^l \rrbracket \text{ Dyn } \mu, \text{upd } \llbracket \nu^l \rrbracket \text{ Dyn } \rho)) \end{aligned}$$

Here follows Bondorf's binding-time analysis of E .

$$\begin{aligned} \text{Bt} &: (E : A) \rightarrow \text{BMap} \times \text{BEnv} \\ \text{Bt}(E) &= \text{fix}(\lambda(\mu, \rho). (\mu_E^{input}, \rho_E^{input})) \sqcup \text{bt}(E)\mu\rho) \end{aligned}$$

$$b, \text{bt} : (E : A) \rightarrow \text{BMap} \rightarrow \text{BEnv} \rightarrow \text{BMap} \times \text{BEnv}$$

$$\begin{aligned} \text{bt}(E)\mu\rho &= \text{let } (\mu', \rho') \text{ be } b(E)\mu\rho \text{ in} \\ &\quad \text{let } k \text{ be } \text{var}(E) \text{ in} \\ &\quad \text{if } \mu'(k) = \text{Dyn} \\ &\quad \text{then raise } k \mu' \rho' \\ &\quad \text{else } (\mu', \rho') \end{aligned}$$

$$b(x^l)\mu\rho = (\text{upd } \llbracket \nu^l \rrbracket \rho \llbracket \nu^l \rrbracket \mu, \rho)$$

$$\begin{aligned} b(\lambda^l x.E)\mu\rho &= \text{let } (\mu', \rho') \text{ be } \text{bt}(E)\mu\rho \text{ in} \\ &\quad \text{if } \mu'(\llbracket \lambda^l \rrbracket) = \text{Dyn} \\ &\quad \text{then raise var}(E) \mu' \rho' \\ &\quad \text{else } (\mu', \rho') \end{aligned}$$

$$\begin{aligned} b(E_1 @_i E_2)\mu\rho &= \\ &\quad \text{let } (\mu', \rho') \text{ be } (\text{bt}(E_1)\mu\rho) \sqcup (\text{bt}(E_2)\mu\rho) \text{ in} \end{aligned}$$

let c be $L(\text{var}(E_1))$ in
 let μ'' be
 $\text{upd } \llbracket \mathcal{Q}_i \rrbracket (\mu'(\text{var}(E_1)) \sqcup \mu'(\text{var}(E_2)) \sqcup \bigsqcup_{l \in c} \mu'(\text{var}(\text{body}(l)))) \mu'$ in
 let ρ'' be $\rho' \sqcup (\bigsqcup_{l \in c} (\text{upd } \llbracket \nu^l \rrbracket \mu'(\text{var}(E_2))) \rho')$ in
 if $\mu''(\text{var}(E_1)) = \text{Dyn}$
 then raise $\text{var}(E_2) \mu'' \rho''$
 else (μ'', ρ'')

We can now do binding-time analysis of E by computing $\text{fst}(\text{Bt}(E))$.

A Simpler Definition Bondorf's definition can be simplified considerably. To see why, consider the second component of $\text{BMap} \times \text{BEnv}$. This component is updated only in $b(E_1 \mathcal{Q}_i E_2)\mu\rho$ 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 BEnv . By rewriting Bondorf's definition according to this observation, we arrive at the definition below. We use the auxiliary function newraise which is defined as follows.

$\text{newraise} : \text{Metavar} \rightarrow \text{Metavar} \rightarrow \text{BMap} \rightarrow \text{BMap}$
 $\text{newraise } kk' \mu = \text{if } \mu(k) = \text{Dyn}$
 then $\mu \sqcup (\bigsqcup_{l \in L(k')} (\langle \llbracket \lambda^l \rrbracket \mapsto \text{Dyn} \rrbracket \sqcup \langle \llbracket \nu^l \rrbracket \mapsto \text{Dyn} \rrbracket))$
 else μ

As with Bondorf's definition, we assume that all labels are distinct.

$\text{bta}, m : (E : A) \rightarrow \text{BMap} \rightarrow \text{BMap}$
 $\text{bta}(E)\mu = \text{newraise } \text{var}(E) \text{ var}(E) (m(E)\mu)$
 $m(x^l)\mu = \mu$
 $m(\lambda^l x.E)\mu = (\text{bta}(E)\mu) \sqcup (\text{newraise } \llbracket \lambda^l \rrbracket \text{ var}(E) \mu)$
 $m(E_1 \mathcal{Q}_i E_2)\mu =$
 $(\text{bta}(E_1)\mu) \sqcup (\text{bta}(E_2)\mu) \sqcup$
 $\bigsqcup_{l \in L(\text{var}(E_1))} (\langle \llbracket \nu^l \rrbracket \mapsto \mu(\text{var}(E_2)) \rrbracket \sqcup \langle \llbracket \mathcal{Q}_i \rrbracket \mapsto \mu(\text{var}(\text{body}(l))) \rrbracket) \sqcup$
 $(\text{newraise } \text{var}(E_1) \text{ var}(E_2) \mu) \sqcup$
 $\langle \llbracket \mathcal{Q}_i \rrbracket \mapsto \mu(\text{var}(E_1)) \sqcup \mu(\text{var}(E_2)) \rrbracket$

We can now do binding-time analysis of E by computing

$$\text{fix}(\lambda \mu. \mu^{\text{input}} \sqcup \rho^{\text{input}} \sqcup \text{bta}(E)\mu) .$$

A key question is: is the simpler definition equivalent to Bondorf's? We might attempt to prove this using fixed point induction, but we find it much easier to prove that both of them are equivalent to the SF-system presented in the previous section.

Equivalence For every λ -term E where all labels are distinct, we now prove the equivalence of the binding-time analysis of Bondorf, the simplified definition of his analysis, and the analysis specified by the SF-system $B'(E)$. We will use the standard terminology that μ is a *postfixed point* of $m(E)$ if $m(E)\mu \leq \mu$.

Lemma 2. *For every λ -term E , the following properties hold:*

- *If μ is a postfix point of $\mathbf{bta}(E)$, then so is it of $m(E)$.*
- *If μ is a postfix point of $\mathbf{bta}(E)$, then so is it of $\mathbf{bta}(F)$ for every subterm F of E .*
- *If μ is a postfix point of $m(E)$, then so is it of $m(F)$ for every subterm F of E .*

Proof. By induction on the structure of E .

Lemma 3. *$B'(E)$ has least solution $(L, \text{fix}(\lambda\mu.\mu^{\text{input}} \sqcup \rho^{\text{input}} \sqcup \mathbf{bta}(E)\mu))$.*

Proof. We prove a stronger property: the solutions of $B'(E)$ that are of the form (L, M) are exactly the postfix points of $(L, \lambda\mu.\mu^{\text{input}} \sqcup \rho^{\text{input}} \sqcup \mathbf{bta}(E)\mu)$. The proof of this involves repeated use of Lemma 2 and is analogous to the proof of [8, Lemma 5]; we omit the details.

Lemma 4. *$B'(E)$ has least solution $(L, \text{fst}(\mathbf{Bt}(E)))$.*

Proof. Similar to the proof of Lemma 3.

Theorem 2. *For every λ -term E , the three binding-time analyses defined in Section 5.1.1, 5.1.2, and by the SF-system $B'(E)$ are equivalent.*

Proof. Combine Lemmas 3 and 4.

5.2 The Well-annotatedness Predicate

The Original Formulation We recall the binding-time analysis of Schwartzbach and me [10, 7], with a few minor changes in the notation compared to the previous presentations.

First, we introduce two new forms of meta-variables. A meta-variable is of one of the forms $[\nu^l]$, $[\lambda^l]$, $[\mathcal{Q}_i]$, $[\Delta^l]$, and $[\underline{\mathcal{Q}}_i]$. The set of all such meta-variables is denoted Metavar2 . A 2-level λ -term is assigned a meta-variable by the function var , which maps x^l to $[\nu^l]$, $\lambda^l x.W$ to $[\lambda^l]$, $W_1 \mathcal{Q}_i W_2$ to $[\mathcal{Q}_i]$, $\Delta^l x.W$ to $[\Delta^l]$, and $W_1 \underline{\mathcal{Q}}_i W_2$ to $[\underline{\mathcal{Q}}_i]$. (Notice that var is an extension of the previously defined function var that operates on λ -terms.)

To define if a 2-level λ -term W is *well-annotated*, we use an abstract domain $\text{DMap}(W)$ which is defined as follows.

Definition 6. *For a 2-level λ -term W , $\text{Lab}(W)$ is the set of labels on static abstractions occurring in W . Notice that $\text{Lab}(W)$ is finite. The set $\text{CSet}(W)$ is the powerset of $\text{Lab}(W)$; $\text{CSet}(W)$ with the inclusion ordering is a complete lattice. Notice that these definitions of $\text{Lab}(W)$ and $\text{CSet}(W)$ extend the previously given*

ones. We then define $D(W) = \text{CSet}(W) \cup \{\text{Dyn}\}$. The set $D(W)$ is partially ordered by \leq such that if d and d' are sets and $d \subseteq d'$, then $d \leq d'$. Notice that $D(W)$ is not a lattice, since Dyn is incomparable to all other elements of $D(W)$. The set $\text{DMap}(W)$ consists of the total functions from Metavar2 to $D(W)$.

A 2-level λ -term W is said to be well-annotated if the constraint system $WA(W)$ below is solvable. The constraint system is a finite set of conditional clauses over inequalities of the form $P \leq P'$, where P and P' are either meta-variables or elements of $D(W)$. A *solution* of such a system is an element of $\text{DMap}(W)$ that satisfies all constraints.

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

For a 2-level λ -term W , the constraint system $WA(W)$ is the union of the following sets of constraints.

- The singleton set consisting of $\text{var}(W) = \text{Dyn}$.
- For every free variable x^l of W , the singleton set consisting of $\llbracket \nu^l \rrbracket = \text{Dyn}$.
- For every $\lambda^l x.W'$ in W , the singleton set consisting of $\{l\} \leq \llbracket \lambda^l \rrbracket$.
- For every $W_1 @_i W_2$ in W , the singleton set consisting of $\emptyset \leq \text{var}(W_1)$.
- For every $W_1 @_i W_2$ in W and for every $\lambda^l.W'$ in W , the set consisting of the two constraints

$$\{l\} \leq \text{var}(W_1) \Rightarrow \text{var}(W_2) \leq \llbracket \nu^l \rrbracket$$

$$\{l\} \leq \text{var}(W_1) \Rightarrow \text{var}(W') \leq \llbracket @_i \rrbracket$$
- For every $\lambda^l x.W'$ in W , the set consisting of $\llbracket \lambda^l \rrbracket = \llbracket \nu^l \rrbracket = \text{var}(W') = \text{Dyn}$.
- For every $W_1 @_i W_2$ in W , the set consisting of $\text{var}(W_1) = \text{var}(W_2) = \llbracket @_i \rrbracket = \text{Dyn}$.

Fact 3 For all λ -terms, there is a \sqsubseteq -least well-annotated version.

Proof. See [10].

We can now do binding-time analysis of a λ -term by computing the \sqsubseteq -least well-annotated version.

Equivalence For every λ -term E , we now prove the equivalence of the binding-time analysis of PS, and the analysis specified by the SF-system $B(E)$

We will use the notation that if W is a 2-level λ -term, then \hat{W} is the λ -term which is equal to W except that all underlinings have been removed. Moreover, for a 2-level λ -term W , we will write $\text{DLab}(W)$ for the set of labels on dynamic abstractions occurring in W .

Theorem 4. Let E be a λ -term. Let W be an annotated version of E . $WA(W)$ is solvable if and only if $B(E)$ has a solution (L, M) and $T_M(E) = W$.

Proof. Let E be λ -term. Suppose first $B(E)$ has solution (L, M) . We will prove that $WA(T_M(E))$ is solvable. Construct $S \in \text{DMap}(T_M(E))$ as follows. For each subterm F in E , define

$$S(\text{var}(T_M(F))) = \begin{cases} L(\text{var}(F)) & \text{if } M(\text{var}_b(F)) = \text{Stat} \\ \text{Dyn} & \text{if } M(\text{var}_b(F)) = \text{Dyn} \end{cases}$$

On the remainder of its domain, S yields **Dyn**. It is straightforward to check that $WA(T_M(E))$ has solution S .

Suppose then that W is an annotated version of E and suppose that $WA(W)$ has solution S . Construct L as follows. For each subterm W' of W , define

$$L(\text{var}(\hat{W}')) = \begin{cases} S(\text{var}(W')) & \text{if } S(\text{var}(W')) \geq \emptyset \\ \text{DLab}(W') & \text{if } S(\text{var}(W')) = \text{Dyn} \end{cases}$$

On the remainder of its domain, L yields \emptyset .

Construct then M as follows. For each subterm W' of W , define

$$M(\text{var}_b(\hat{W}')) = \begin{cases} \text{Stat} & \text{if } S(\text{var}(W')) \geq \emptyset \\ \text{Dyn} & \text{if } S(\text{var}(W')) = \text{Dyn} \end{cases}$$

On the remainder of its domain, M yields **Stat**. Clearly, $T_M(\hat{W}) = W$. It is straightforward to check that $B(\hat{W})$ has solution (L, M) .

Theorem 5. *Let E be a λ -term and suppose $B(E)$ has least solution (L, M) . Then $T_M(E)$ is the least well-annotated version of E .*

Proof. By Theorem 4, $T_M(E)$ is well-annotated. Let W_l be the least well-annotated version of E . By Theorem 4, choose (L', M') so that $B(E)$ has solution (L', M') and $T_{M'}(E) = W_l$. We have $M \leq M'$ so by Lemma 1, we then get that $T_M(E) \sqsubseteq W_l$. From W_l and $T_M(E)$ being well-annotated and from W_l being the least of the well-annotated versions of E , we get that $W_l \sqsubseteq T_M(E)$. In conclusion, $T_M(E) = W_l$.

From Theorem 5 follows the desired result:

Theorem 6. *For every λ -term E , the binding-time analyses defined in Section 5.2.1 and by the SF-system $B(E)$ are equivalent.*

In previous work [7], we proved that any binding-time analysis that always produces well-annotated 2-level terms is correct. Since for every λ -term E , we have $B(E) \subseteq B'(E)$, the least solution (L, M) of $B'(E)$ is also a solution of $B(E)$. By Theorem 4, $T_M(E)$ is well-annotated. We thus get that Bondorf's analysis, when restricted to the pure λ -calculus, is correct.

6 Concluding Remarks

In a previous paper [10], we proved that the type inference based binding-time analysis of Gomard [5] is more conservative than the analysis of PS. In the papers [7, 10] we emphasized that the analysis of PS was originally intended to capture the outputs of the binding-time analyses of Bondorf and Consel. With the result of this paper, it is now clarified that the analyses of Bondorf is more conservative than the analysis of PS. In future work, we would like to make comparisons with also the analyses of Consel and of Bondorf and Jørgensen.

Acknowledgement. The author thanks Paul Steckler, Mitchell Wand, and the anonymous referees for helpful comments on a draft of the paper. The results of this paper were obtained at Northeastern University, Boston. The author is currently hosted by BRICS, Basic Research in Computer Science, Centre of the Danish National Research Foundation.

References

1. Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. North-Holland, 1981.
2. Anders Bondorf. Automatic autoprojection of higher order recursive equations. *Science of Computer Programming*, 17(1-3):3-34, December 1991.
3. Anders Bondorf and Jesper Jørgensen. Efficient analyses for realistic off-line partial evaluation. *Journal of Functional Programming*, 3(3):315-346, 1993.
4. Charles Consel. Binding time analysis for higher order untyped functional languages. In *Proc. ACM Conference on Lisp and Functional Programming*, pages 264-272, 1990.
5. Carsten K. Gomard. Partial type inference for untyped functional programs. In *Proc. ACM Conference on Lisp and Functional Programming*, pages 282-287, 1990.
6. Hanne R. Nielson and Flemming Nielson. Automatic binding time analysis for a typed λ -calculus. *Science of Computer Programming*, 10:139-176, 1988.
7. Jens Palsberg. Correctness of binding-time analysis. *Journal of Functional Programming*, 3(3):347-363, 1993.
8. Jens Palsberg. Closure analysis in constraint form. *ACM Transactions on Programming Languages and Systems*, 1995. To appear. Also in Proc. CAAP'94, Colloquium on Trees in Algebra and Programming, Springer-Verlag (LNCS 787), pages 276-290, Edinburgh, Scotland, April 1994.
9. Jens Palsberg and Michael I. Schwartzbach. Safety analysis versus type inference for partial types. *Information Processing Letters*, 43:175-180, 1992.
10. Jens Palsberg and Michael I. Schwartzbach. Binding-time analysis: Abstract interpretation versus type inference. In *Proc. ICCL'94, Fifth IEEE International Conference on Computer Languages*, pages 289-298, Toulouse, France, May 1994.
11. Jens Palsberg and Michael I. Schwartzbach. *Object-Oriented Type Systems*. John Wiley & Sons, 1994.
12. Jens Palsberg and Michael I. Schwartzbach. Safety analysis versus type inference. *Information and Computation*, 118(1):128-141, 1995.
13. Peter Sestoft. Replacing function parameters by global variables. Master's thesis, DIKU, University of Copenhagen, September 1989.