Analyses that Distinguish Different Evaluation Orders
or
Unsoundness Results in Control-Flow Analysis

Mitchell Wand*
College of Computer Science
Northeastern University
Boston, MA 02115, USA
wand@ccs.neu.edu

July 17, 2002

Abstract

The standard control-flow analysis for higher-order languages, 0CFA, as defined by Sestoft, Shivers, Palsberg, *et al.*, has been shown correct for a variety of semantics. One view of this family of results is that it shows that 0CFA is insensitive to evaluation order. We present simple modifications of 0CFA for call-by-value and call-by-name that distinguish call-by-value, call-by-name, and unrestricted β-reduction.

1 Introduction

The standard control-flow analysis for higher-order languages, 0CFA, as defined by Sestoft, Shivers, Palsberg, *et al.*, has been shown correct for a variety of semantics. A sample of these results is shown in figure 1.

Our goal is to find analyses that will distinguish between call-by-name, call-by-value, and unrestricted β-reduction.

*Work supported by the National Science Foundation under grants numbered CCR-9404646 and CCR-9629801.

Author	Language or Reduction Order	Semantics	
Sestoft 89[Ses89]	CBV	Big-Step	Closure Semantics
Sestoft 91[Ses91]	CBN	Small-Step	Closure Semantics
Shivers 91[Shi91]	CPS	Denotational	Closure Semantics
Palsberg & Schwartzbach 95[PS95]	CBV	Big-Step	Closure Semantics
Palsberg & Schwartzbach 95[PS95]	CBN	Big-Step	Closure Semantics
Flanagan & Felleisen 95[FF95]	ANF	Small-Step	Closure Semantics
Palsberg 95*[Pal95]	β	Small-Step	Substitution Semantics
Nielson & Nielson 97[NN97]	CBV	Small-Step	Closure Semantics
Wand & Williamson 02[WW02]	β	Small-Step	Substitution Semantics

Figure 1: 0CFA Soundness Results

We do this in the framework of *constraint-based* analysis [PS95]. This approach is implicit in all of the work cited in figure 1. The basic idea of constraint-based analysis is that to analyze an expression e, we, generate a set $[\![e]\!]$ of constraints. Then any solution Φ to these conditional constraints is an analysis of e. We write $\Phi \models [\![e]\!]$ ("Φ is an analysis of e" or "Φ describes e"). We normally work with labelled expressions; the label of an expression e is denoted by $\mathrm{lab}(e)$.

We consider the soundness of such an analysis relative to some small-step or reduction semantics. In such a framework, a soundness theorem typically takes the following form:

Typical Soundness Theorem: *if* $\Phi \models [\![e]\!]$ *and* $e \to e'$, *then*

1. $\Phi \models [\![e']\!]$
2. $\Phi(\mathrm{lab}(e')) \subseteq \Phi(\mathrm{lab}(e))$

A general technique for proving such a theorem is presented in [WW02]. The semantics is presented as a reduction system for labelled expressions, in which each expression or value is labelled with its origin (an idea dating back to [JM82]). The proof relies on a *local structure heuristic:* $[\![e]\!]$ depends only on local structure of e. The local structure of e' is not much different from that of e. Hence the constraints in $[\![e']\!]$ are not much different from those in $[\![e]\!]$. So the proof just needs to account for the differences.

From such a soundness theorem, it is easy to deduce

Typical Prediction Theorem: *if* $\Phi \models [\![e]\!]$, *and* $e \to^* \lambda^l x^\beta.e_0$, *then* $l \in \Phi(\text{lab}(e))$.

This says that every possible result of evaluating e is predicted by any analysis of e. The proof is easy: by part 1 of the soundness theorem , $\Phi \models [\![\lambda^l x^\beta.e_0]\!]$, hence $l \in \Phi(l)$. But by part 2 of the soundness theorem, $\Phi(l) \subseteq \Phi(\text{lab}(e))$.

This approach works for all of the small-step semantics above. How then, are we to distinguish call-by-name from call-by-value? For the pure λ-calculus, the only observable difference is termination: if e reduces to an abstraction under call-by-value, then it reduces to the same abstraction under call-by-name. But in general, non-termination is not observable by 0CFA: the soundness theorems are about safety (partial correctness).[1]

We solve this problem by adding another observable effect: *errors*. In the presence of errors, CBN and CBV are incomparable. This gives us enough different behaviors to make the difference detectable.

In section 2 we present the syntax of our language and three reduction semantics for it. In section 3 we present the classic 0CFA for this language, and in section 4 we show how to modify this analysis for CBN and CBV. In sections 5 and 6, we present our main results: the soundness of each analysis for its corresponding semantics and the unsoundness of each analysis for the other semantics. We conclude with related and future work.

2 The Language

The syntax of our language is shown in figure 2. It consists of a λ-calculus with a single scalar constant $\bullet$, and in which every expression is labelled. Expressions reduce either to a scalar or an abstraction; following [Plo75] we call these *values*. The result of a computation is either a value or an error.

We assume that every expression is *well-labelled*: that every bound occurrence of a variable is bound with the same label as its binding occurrence, and its free variables are consistently labelled. We write $\Gamma \vdash e\ \mathsf{wl}$ when e is well-labelled, where $\Gamma : \mathit{Var} \to \mathit{Lab}$ describes the labels of e's free variables. Well-labelledness is preserved under all notions of reduction that we consider. A formal definition, and the key lemmas, are presented in [WW02].

[1] Occasionally a control flow analysis can predict non-termination: If $\Phi(\text{lab}(e))$ is empty, then by the prediction theorem, e cannot terminate with an abstraction.

$$
\begin{array}{rcll}
x & \in & \mathit{Var} & \text{Variables} \\
l, \beta & \in & \mathit{Lab} & \text{Labels} \\
e & ::= & x^{\beta} \mid (e_1\ e_2)^{l} \mid (\lambda^{l} x^{\beta}.e) \mid \bullet^{l} & \text{Expressions} \\
v & ::= & \bullet^{l} \mid (\lambda^{l} x^{\beta}.e) & \text{Values} \\
r & ::= & v \mid \mathit{err} & \text{Results}
\end{array}
$$

Figure 2: Syntax

Our choice of notation (taken from [FF95]) minimizes the profusion of labels (compare [WS99]); nevertheless we will omit labels when they are not important.

We write $e \in e'$ whenever e occurs as a subexpression of e', *ie.,* whenever $e' = C[e]$ for any context C, including both the empty context and binding contexts.

We express our semantics using reduction contexts, as shown in figure 3. The first reduction rule is ordinary β-reduction; the second says that if a scalar is applied to an argument in a reduction context, then the entire computation aborts with an error.

The different semantics are distinguished by different choices for the definition of reduction contexts E. For unrestricted β-reduction, a reduction can take place anywhere in a term. For call-by-value, the operator is reduced first, then the operand is reduced, so an application can be reduced only when both operator and operand have been reduced to values. For call-by-name, the operator is reduced first; the application is reduced as soon as the operator is reduced to a value.

To illustrate the differences, we list some expressions with different behaviors under these reduction strategies. In general, CBV and CBN on a given term will yield incomparable results, but any result obtainable by CBV or CBN is also reachable by β-reduction, though it may not be a final value under β.

$$E[\ ((\lambda^{l'} x^{\beta}.e_0)\ e_1)^l\ ] \to E[\ e_0[e_1/x]\ ]$$
$$E[\ (\bullet^{l'}\ e)^l\ ] \to err$$

$$\begin{array}{ll} \beta: & E ::= [\ ] \mid (E\ e)^l \mid (e\ E)^l \mid \lambda^l x^{\beta}.E \\ \mathrm{CBV}: & E ::= [\ ] \mid (E\ e)^l \mid ((\lambda^{l'} x^{\beta}.e_0)\ E)^l \\ \mathrm{CBN}: & E ::= [\ ] \mid (E\ e)^l \end{array}$$

Figure 3: Reduction Semantics

$$\begin{array}{rll} (\lambda x.(\bullet\ x)) & \to^* (\lambda x.(\bullet\ x))(\lambda x.(\bullet\ x)) & \text{in CBV, CBN, } \beta \\ & \to^* err & \text{in } \beta \\ & & \\ ((\lambda x.(\lambda^1 y.y))\ (\bullet\ (\lambda^2 z.z))) & \to^* (\lambda^1 y.y) & \text{in } \beta\text{, CBN} \\ ((\lambda x.(\lambda^1 y.y))\ (\bullet\ (\lambda^2 z.z))) & \to^* err & \text{in } \beta\text{, CBV} \\ & & \\ ((\lambda x.(\bullet\ x))\ ((\lambda x.(x\ x))\ (\lambda x.(x\ x)))) & \to^* err & \text{in CBN, } \beta \\ ((\lambda x.(\bullet\ x))\ ((\lambda x.(x\ x))\ (\lambda x.(x\ x)))) & \not\to^* v & \text{for any } v \text{ in CBV} \end{array}$$

3 Analyses

For each expression e, we generate a set $[\![e]\!]$ of constraints on a function

$$\Phi : Lab \to \mathcal{P}(Lab \cup \{err\})$$

We can think of Φ as a collection of sets $\Phi(l) \subseteq Lab \cup \{err\}$. $\Phi(l)$ is intended to be the set of labels of possible outcomes of evaluating an expression labelled l.

We do this first for classical 0CFA, which corresponds to β-reduction, and then for CBV and CBN. We present this as a set of deduction rules, shown in figure 4. The first pair of rules correspond to abstractions and scalars, which are self-evaluating. There is no rule for variables, because expressions are assumed to be well-labelled, and therefore each variable carries the same label as its binder. The second rule is the classical connecting rule for applications [PS95]. The remaining rules deal with errors: the first of these rules says that if the operator of

$$\frac{(\lambda^l x.e_0) \in e}{(l \in \Phi(l)) \in [\![e]\!]} \qquad \frac{\bullet^l \in e}{(l \in \Phi(l)) \in [\![e]\!]}$$

$$\frac{(e_1\ e_2)^l \in e \quad (\lambda^{l'} x^\beta.e_0) \in e}{\begin{array}{l}(l' \in \Phi(\mathrm{lab}(e_1)) \Rightarrow \ \Phi(\mathrm{lab}(e_2)) \subseteq \Phi(\beta) \\ \qquad\qquad\qquad\qquad \wedge\ \Phi(\mathrm{lab}(e_0)) \subseteq \Phi(l)) \in [\![e]\!]\end{array}}$$

$$\frac{(e_1\ e_2)^l \in e \quad \bullet^{l'} \in e}{l' \in (\Phi(\mathrm{lab}(e_1)) \Rightarrow err \in \Phi(l)) \in [\![e]\!]}$$

$$\frac{(e_1\ e_2)^l \in e}{(err \in \Phi(\mathrm{lab}(e_i)) \Rightarrow err \in \Phi(l)) \in [\![e]\!]} \quad i = 1,2$$

$$\frac{(\lambda^l x.e_0) \in e}{(err \in \Phi(\mathrm{lab}(e_0)) \Rightarrow err \in \Phi(l)) \in [\![e]\!]}$$

Figure 4: Rules for generating constraints (for unrestricted β)

an application can evaluate to a scalar, then evaluating the application may result in an error. The remaining rules propagate this information to any term containing such an application.

This is ordinary 0CFA and is sound for unrestricted β-reduction. The proof follows [WW02], suitably modified to account for scalars and errors. The presentation of the constraint-generation algorithm as a set of non-recursive rules (essentially a large comprehension over the source expression), rather than as an inductive system, makes the proof much easier and avoids a variety of subtle errors to which an inductive presentation is prey.

From the prediction theorem and the sample reductions above, we learn facts like

$$\Phi \models [\![(\lambda^1 x.(\bullet\ x))]\!] \Rightarrow err \in \Phi(1)$$

and

$$\Phi \models [\![((\lambda x.(\lambda y.y))\ (\bullet\ (\lambda^1 x.x)))^2]\!] \Rightarrow \{1, err\} \subseteq \Phi(2)$$

4 Modifying the analysis for CBN and CBV

In order to modify the analysis to handle call-by-name or call-by-value, we add a *reachability predicate*. The rules will guarantee that any subexpression that is evaluated will be marked as reachable. We arrange things so that constraints are generated only for subexpressions that are reachable. Reachability predicates were introduced in [GNN97], though they are implicit in a number of earlier works; see section 7.

Formally, we add a new set $R \subseteq Lab$ of *reachable* labels, and our formulas will now constrain Φ and R. Our new formulas will be Horn clauses, as before, but they will contain conjuncts about reachability, so the rules will only fire for reachable expressions.

We will have two sets of rules: one for $[\![e]\!]_{\mathrm{CBN}}$ and one for $[\![e]\!]_{\mathrm{CBV}}$. These are summarized in figure 5. These rules generate constraints for the analysis $[\![e]\!]_X$ of some expression e under some reduction strategy X, where X is either CBN or CBV. The consequences of the rules are always Horn formulas to be added to $[\![e]\!]_X$, so we omit the "$\in [\![e]\!]_X$" from each rule.

The first two rules mark abstractions and scalars as self-evaluating, as before, except that they only do so when they are reachable. The next rule is the standard rule for propagating values through applications, again restricted to reachable applications. The next pair of rules raise and propagate errors for reachable applications.

The remainder of the rules deal with reachability. The first of these marks the top-level expression e as reachable. (Without this rule there would be no reachable expressions!) The next rule says that if an application is reachable, so is its operator. The next rule deals with reachability of an operand. It says that if an application is reachable and its operator may evaluate to some abstraction, then

1. under CBV, the operand is always reachable
2. under CBN, the operand is reachable if the bound variable appears free in the body of the abstraction.

Notice that if $\Phi(\mathrm{lab}(e_1)) = \emptyset$, then e_1 is guaranteed not to terminate, and so the operand is unreachable under either CBN or CBV.

The last rule deals with reachability of the body of the abstraction that flows into the operand. Under CBN, the body is always reachable, because it is the next expression evaluated. Under CBV, the body is reachable only if the operand can

$$\frac{(\lambda^l x^\beta.e_0) \in e}{(l \in R) \Rightarrow (l \in \Phi(l))} \qquad \frac{\bullet^l \in e}{(l \in R) \Rightarrow (l \in \Phi(l))}$$

$$\frac{(e_1\ e_2)^l \in e \qquad (\lambda^{l'} x^\beta.e_0) \in e}{(l \in R) \wedge (l' \in \Phi(\mathrm{lab}(e_1))) \Rightarrow \begin{array}{l} \Phi(\mathrm{lab}(e_2)) \subseteq \Phi(\beta) \\ \wedge\ \Phi(\mathrm{lab}(e_0)) \subseteq \Phi(l) \end{array}}$$

$$\frac{(e_1\ e_2)^l \in e \qquad \bullet^{l'} \in e}{l \in R\ \wedge\ l' \in \Phi(\mathrm{lab}(e_1)) \Rightarrow err \in \Phi(l)}$$

$$\frac{(e_1\ e_2)^l \in e}{l \in R\ \wedge\ err \in \Phi(\mathrm{lab}(e_i)) \Rightarrow err \in \Phi(l)} \quad i = 1,2$$

$$\frac{}{\mathrm{lab}(e) \in R}$$

$$\frac{(e_1\ e_2)^l \in e}{(l \in R) \Rightarrow \mathrm{lab}(e_1) \in R}$$

$$\frac{(e_1\ e_2)^l \in e \qquad (\lambda^{l'} x^\beta.e_0) \in e}{(l \in R) \wedge (l' \in \Phi(\mathrm{lab}(e_1))) \Rightarrow \begin{array}{l} \mathrm{lab}(e_2) \in R \\ X = \mathrm{CBV}\ \vee\ (X = \mathrm{CBN}\ \wedge\ x \in \mathrm{fvars}(e_0)) \end{array}}$$

$$\frac{(e_1\ e_2)^l \in e \qquad (\lambda^{l'} x^\beta.e_0) \in e}{(l \in R) \wedge (l' \in \Phi(\mathrm{lab}(e_1))) \Rightarrow \begin{array}{l} \boxed{\mathrm{lab}(e_0) \in R \quad (X = \mathrm{CBN})} \\ \boxed{\Phi(\mathrm{lab}(e_2)) \neq \emptyset \Rightarrow \mathrm{lab}(e_0) \in R \quad (X = \mathrm{CBV})} \end{array}}$$

Figure 5: Rules for $[\![e]\!]_X$ ($X = \mathrm{CBN}$ or CBV)

possibly produce a value. Our presentation of this clause is not quite a Horn clause, but it is easily converted to one.

5 Soundness Results

The CBN and CBV rules are each sound for their respective notions of reduction:

Theorem 1 (Prediction for CBN, CBV**)** *If* $X = \mathrm{CBN}$ *or* CBV, *and* $\emptyset \vdash e\ \mathsf{wl}$ *and* $(R,\Phi) \models [\![e]\!]_X$, *then*

1. *if* $e \to_X^* (\lambda^l x^\beta.e_0)$, *then* $l \in \Phi(\mathrm{lab}(e))$.
2. *if* $e \to_X^* err$, *then* $err \in \Phi(\mathrm{lab}(e))$.

We prove the theorem separately for the CBN and CBV cases. We first show that in each case

$$(R,\Phi) \models [\![E[r]]\!]_X \Rightarrow \mathrm{lab}(r) \in R$$

This says that every actual redex is marked as reachable. We then proceed by a series of lemmas much like those in [WW02, Definition 4.5–4.7]; the theorem then follows using a calculation like that of [WW02, Theorem 4.8].

6 Unsoundness Results

Theorem 2 (Separation)

1. $[\![-]\!]_{\mathrm{CBV}}$ *and* $[\![-]\!]_{\mathrm{CBN}}$ *are unsound for* $\to_\beta^*$.
2. $[\![-]\!]_{\mathrm{CBN}}$ *is unsound for* $\to_{\mathrm{CBV}}^*$
3. $[\![-]\!]_{\mathrm{CBV}}$ *is unsound for* $\to_{\mathrm{CBN}}^*$

Proof: For each case, we present a counterexample to the prediction theorem.

1. Let $e_1 = (\lambda^1 x^2.(\bullet^3\ x^2)^4)$. Let $R = \{1\}$, $\Phi(1) = \{1\}$, all other $\Phi(l)$ empty. Then $(R,\Phi) \models [\![e_1]\!]_{\mathrm{CBV,CBN}}$, but $e \to_\beta^* err$. So $[\![-]\!]_{\mathrm{CBV}}$ and $[\![-]\!]_{\mathrm{CBN}}$ are unsound for $\to_\beta^*$.

2. Let $e_2 = ((\lambda^1 x.(\lambda^2 y.y))\ (\bullet\ (\lambda^3 z.z)))^4$. Then there exists (R,Φ) such that $(R,\Phi) \models [\![e_2]\!]_{\mathrm{CBN}}$ and $\Phi(4) = \{2\}$. But $e_2 \rightarrow^*_{\mathrm{CBV}} err$. So $[\![-]\!]_{\mathrm{CBN}}$ is unsound for $\rightarrow^*_{\mathrm{CBV}}$.

3. Let $e_3 = ((\lambda^1 x^2.(\bullet^3\ x^2)^4)\ ((\lambda x.(x\ x))\ (\lambda x.(x\ x)))^5)$ This loops under CBV, but produces an error under CBN. However, the smallest CBV analysis has $\Phi(5) = \emptyset$. Therefore, the body 4 of the operator is not marked as reachable, and hence no error is predicted. Therefore $[\![-]\!]_{\mathrm{CBV}}$ is unsound for $\rightarrow^*_{\mathrm{CBN}}$.

The counterexamples are summarized in Figure 6. This theorem presents the best possible separation, since any possible result of a computation under CBV or CBN is also possible under unrestricted β reduction. Therefore any analysis that is sound for β is also sound for CBV and CBN.

	$\rightarrow_\beta$	$\rightarrow_{\mathrm{CBN}}$	$\rightarrow_{\mathrm{CBV}}$
$[\![-]\!]_\beta$	–	–	–
$[\![-]\!]_{\mathrm{CBN}}$	e_1	–	e_2
$[\![-]\!]_{\mathrm{CBV}}$	e_1	e_3	–

Figure 6: Summary of unsoundness results

$$\frac{}{\mathrm{lab}(e) \in R}$$

$$\frac{(e_1\ e_2)^l \in e}{(l \in R) \Rightarrow \mathrm{lab}(e_1) \in R \wedge \mathrm{lab}(e_2) \in R}$$

$$\frac{(e_1\ e_2)^l \in e \qquad (\lambda^{l'} x^\beta.e_0) \in e}{(l \in R) \wedge (l' \in \Phi(\mathrm{lab}(e_1))) \Rightarrow \mathrm{lab}(e_0) \in R}$$

Figure 7: Reachability constraints for [NN97] and [PS95]

7 Related Work

Palsberg and Schwartzbach [PS95] introduced the idea of using 0CFA for safety analysis, as we have done here. They were concerned primarily with comparing their safety analysis to a type system; their system was shown to accept more terms than the type system because the type system had to analyze every term but their system analyzed only reachable ones.

Nielson and Nielson [NN97] introduced a class of analyses based on a co-inductive definition of satisfaction. Like [PS95], they had a reachability condition that was encoded in the inductive definition. Explicit reachability conditions were introduced in [GNN97], which also showed how to transform a coinductive definition of satisfaction into a constraint-generation system.

Both [PS95] and [NN97] used the same reachability condition, in which both the operator and operand of a reachable application are reachable, and in which the body of an abstraction is marked as reachable whenever it may appear as the value of the operator of a reachable application; see figure 7 (*cf.* the rules in figure 5). These reachability rules are sound for both CBV and CBN.

Reachability sets are used in [TP00] to keep track of reachable methods during the analysis of an object-oriented program. The memento cache in [NN98, NN99] may be regarded as a multivalued generalization of a reachability set.

The idea of validating a single analysis against multiple evaluation semantics comes from [PS95]. [Pal95] introduced the idea of validating 0CFA against unrestricted β-reduction. Unfortunately that proof was flawed (hence the asterisk in figure 1); a correct proof is found in [WW02].

8 Future Work

Our results are for small-step substitution semantics; we conjecture that the results also hold for big-step environment semantics of CBV and CBN using closures, and for small-step environment semantics using continuations or similar mechanism.

We would also like to apply the local structure heuristic to get easy soundness results for more complex reduction systems involving stores, continuations, events, threads, and the like.

References

[FF95] Cormac Flanagan and Matthias Felleisen. The semantics of future and its use in program optimization. In ACM, editor, *Proceedings 22nd Annual ACM Symposium on Principles of Programming Languages*, pages 209–220, 1995.

[GNN97] Kirsten L. Solberg Gasser, Flemming Nielson, and Hanne Riis Nielson. Systematic realisation of control flow analyses for CML. *ACM SIGPLAN Notices*, 32(8):38–51, August 1997.

[JM82] Neil D. Jones and Steven S. Muchnick. A flexible approach to interprocedural data flow analysis and progrms with recursive data structures. In *Conf. Rec. 9th ACM Symposium on Principles of Programming Languages*, pages 66–74, 1982.

[NN97] Flemming Nielson and Hanne Riis 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, January 1997.

[NN98] Hanne Riis Nielson and Flemming Nielson. Flow logics for contraint based analysis. In *Compiler Construction CC'98*, volume 1383 of *Lecture Notes in Computer Science*, pages 109–127, Berlin, Heidelberg, and New York, 1998. Springer-Verlag.

[NN99] F. Nielson and H. R. Nielson. Interprocedural control flow analysis. In *European Symposium on Programming 1999*, volume 1576 of *Lecture Notes in Computer Science*, pages 20–39, Berlin, Heidelberg, and New York, 1999. Springer-Verlag.

[Pal95] Jens Palsberg. Closure analysis in constraint form. *ACM Transactions on Programming Languages and Systems*, 17(1):47–62, January 1995.

[Plo75] Gordon D. Plotkin. Call-by-name, call-by-value and the λ-calculus. *Theoretical Computer Science*, 1:125–159, 1975.

[PS95] Jens Palsberg and Michael I. Schwartzbach. Safety analysis versus type inference. *Information and Computation*, 118(1):128–141, 1995.

[Ses89] Peter Sestoft. Replacing function parameters by global variables. Master's thesis, DIKU, University of Copenhagen, Copenhagen, 1989.

[Ses91] Peter Sestoft. *Analysis and efficient implementation of functional programs*. PhD thesis, DIKU, University of Copenhagen, Copenhagen, 1991.

[Shi91] Olin Shivers. *Control-Flow Analysis of Higher-Order Languages*. PhD thesis, Carnegie-Mellon University, May 1991.

[TP00] Frank Tip and Jens Palsberg. Scalable propagation-based call graph construction algorithms. *ACM SIGPLAN Notices*, 35(10):281–293, 2000.

[WS99] Mitchell Wand and Igor Siveroni. Constraint systems for useless variable elimination. In *Proceedings 26th Annual ACM Symposium on Principles of Programming Languages*, pages 291–302, 1999.

[WW02] Mitchell Wand and Galen B. Williamson. A modular, extensible proof method for small-step flow analyses. In Daniel Le Métayer, editor, *Programming Languages and Systems, 11th European Symposium on Programming, ESOP 2002, Proceedings*, volume 2305 of *Lecture Notes in Computer Science*, pages 213–227, Berlin, Heidelberg, and New York, 2002. Springer-Verlag.