Tracking Available Values for Lightweight Closures (Summary)

Paul Steckler and Mitchell Wand *

College of Computer Science
Northeastern University
360 Huntington Avenue, 161CN
Boston, MA 02115, USA
{steck, wand}@ccs.neu.edu

March 28, 1994

Abstract

In earlier work, we showed how the free variables of a procedure in a higher-order programming language can sometimes be omitted from the closure for that procedure. The omitted free variables were supplied as extra arguments at the procedure's call sites. In this paper, we generalize that result: We can omit from the closure any variable whose value is available at all of the call sites of the procedure. Which free variables have available values is determined by solving a system of constraints on parse tree annotations. A proof is given that any solution to the constraints is sound, and justifies the resulting transformation.

1 Introduction

In our POPL'94 paper [2], we considered the problem of *lightweight* closure conversion: if we know that some of the free variables of a procedure are available at all the call sites of that procedure, we can avoid putting those variables in the closure of that procedure. For example, in

```
(lambda (x a)
  (let ((f (lambda (y) (lambda (z) (- x y z))))
        (g (lambda (h) (h 3))))
    (g (f a))))
```

one might expect the closure for `(lambda (z) (- x y z))` to contain bindings for the `x` and `y` variables. But at the only call site for this procedure (at `(h 3)`), the value of `x` is available, so instead of keeping `x` in the closure, we can pass it at procedure call time. This is as if first we transformed the program to

```
(lambda (x a)
  (let ((f (lambda (y) (lambda (x z) (- x y z))))
        (g (lambda (h) (h x 3))))
    (g (f a))))
```


*Work supported by the National Science Foundation and ARPA under grants CCR-9002253 and CCR-9014603.

and then performed ordinary closure conversion. Our closure conversion algorithm is not actually staged in this way; the transform adds the extra arguments at the same time explicit closures are created.

We call a variable like `x`, which the procedure expects to pick up at the call site, a *dynamic* variable. A closure in which the dynamic variables are not included is called a *lightweight* closure. This analysis must ensure not only that the variable is lexically available, but also that it has the right binding; see [2] for details.

Sometimes, however, we find that a free variable of the procedure is not lexically available at its call sites, but that variable's *value* is still available at the call site, because some *other* variable has been bound to it. A new, more complex program analysis allows us to determine when we can supply such other variables as additional arguments at call sites.

In our small program, for example, the value of `y` is also available at the call site, under the name `a`. So we can transform our program to

```
(lambda (x a)
  (let ((f (lambda (y) (lambda (x y z) (- x y z))))
        (g (lambda (h) (h x a 3))))
    (g (f a))))
```

and proceed with closure conversion.

2 Revising the analysis

We peform our analysis and transformation following the paradigm set out in [2]. This methodology proceeds as follows: (1) Begin with an evaluator that takes and returns an *occurrence closure*: a node of the parse tree of the program and an environment. (2) Specify the syntax and semantics of a language of propositions on occurrence closures. (3) Formulate a set of *local constraints* ("flow equations") and show that if one annotates every node of the parse tree with a proposition in a way that satisfies all the local constraints, then all the propositions must be true. (4) Use the annotations to generate the transformed program, and then use the truth of the propositions to prove the equivalence of the transformed program to the original. Readers should refer to our POPL'94 paper [2] for required definitions and a fuller description of the methodology.

For the new analysis, we proceed by extending the invariance sets θ used in our earlier paper. There, each node i of the parse tree was annotated with a set θ of variables. The local constraints guaranteed that if $(i, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$ and $x \in \theta_i$, then (among other things) $\psi(x) = \psi'(x)$. To do the new analysis, we make θ a binary relation on variables, which we call an *invariance relation*. Then we formulate local constraints that ensure that if $(i, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$ and $(x, y) \in \theta_i$, then (again, among other things) $\psi(x) = \psi'(y)$.

2.1 Languages and Evaluators

Our input and output languages are unchanged for the new analysis. The input language, Λ_{in}, is an untyped λ-calculus with constants, boolean values, and a conditional. The output language, Λ_{out}, adds records. Therefore, the term and occurence evaluators are also unchanged. Recall, $\underset{t}{\Longrightarrow}$ is the evaluation relation for terms and their values, and $\underset{oc}{\Longrightarrow}$ is the evaluation relation on occurrence closures and their values.

2.2 A new semantics of annotations

In our earlier paper, we formalized two notions of satisfaction. The first notion specified a satisfaction relation on environments and proposition environments, the other gave a satisfaction relation on values and propositions. Only the first of these definitions needs modification for the new analysis. Of course, since these two definitions are mutually recursive, the meaning of both has in fact changed.

We now say:

1. An occurrence environment ψ satisfies a proposition environment A under a protocol assignment Π, as follows:

$$\psi \overset{\Pi}{\models} A$$

   iff $\forall x \in Dom(A)$

   (a) $x \in Dom(\psi)$ and $\psi(x) \overset{\Pi}{\models} A(x)$, and

   (b) If $A(x) = (\phi, \pi, \theta)$, $\psi(x) = (i, \psi')$, and $(y, z) \in \theta$, then $y \in Dom(\psi)$, $z \in Dom(\psi')$, and $\psi(y) = \psi'(z)$

   and, as before

2. An occurrence closure (i, ψ) satisfies a proposition (ϕ, π, θ) under a protocol assignment Π, as follows:

$$(i, \psi) \overset{\Pi}{\models} (\phi, \pi, \theta)$$

   iff

   (a) $[\![i]\!]$ is a constant, variable, **true**, or **false**, or

   (b) $Abs(i)$, $\Pi(i) = \pi$ and there exists A such that $(i, A) \in \phi$ and $\psi \overset{\Pi}{\models} A$.

A new set of local constraints is shown in Figure 1. A proposition $\mathcal{P}$ is now a triple consisting of a flow ϕ (a set of abstract closures), a protocol tag π (describing an application protocol), and an invariance relation θ. For convenience, we write $\{(x, \_)\}$ for the set of all pairs of variables with x in the first position.

The ordering on propositions is as before:

$$(\phi, \pi, \theta) \leq (\phi', \pi', \theta') \text{ iff } \phi \subseteq \phi' \wedge \pi = \pi' \wedge \theta' \subseteq \theta$$

The constraints on flows are identical to those in the earlier analysis. For protocol tags, the constraints are again the same, except for applications, where before we had the constraint

$$\pi_{i.rator} = cl_\sigma \Longrightarrow \lceil \sigma \rceil \subseteq \theta_{i.rator}$$

we now have

$$\pi_{i.rator} = cl_\sigma \Longrightarrow \forall y \in \lceil \sigma \rceil,\ \exists x \text{ such that } (x, y) \in \theta_{i.rator}$$

The old constraint said that if lightweight closures are applied at a call site, the set of supplied dynamic variables must be a subset of the invariance set associated with the call site. But now the θ's are relations, not sets. So the new constraint says that if we apply lightweight closures at a call site, each dynamic variable x must have a representative element (x, y) in θ. Of course, for the new θ's, the new constraints are significantly different.

$$Var\,(i) \Longrightarrow A_i(\llbracket i \rrbracket) = \mathcal{P}_i$$

$$Const\,(i) \Longrightarrow \theta_i = \emptyset$$

$$Abs\,(i) \Longrightarrow \begin{cases} A_{i.body} = A_i[\llbracket i.bv \rrbracket \mapsto \mathcal{P}_{i.bv}] \\ \{(i, A_i)\} \subseteq \phi_i, \text{ and} \\ (x, y) \in \theta_i \Longrightarrow x = y \wedge x \in Dom(A_i) \end{cases}$$

$$App\,(i) \text{ and } Const\,(i.rator) \Longrightarrow \begin{cases} A_{i.rand} = A_i, \text{ and} \\ \theta_i = \emptyset \end{cases}$$

$$App\,(i) \text{ and } \neg\, Const\,(i.rator) \Longrightarrow \begin{cases} A_{i.rator} = A_{i.rand} = A_i, \\ \theta_i \subseteq \theta_{i.rator}, \\ (x, y) \in \theta_{i.rator} \cap \theta_{i.rand} \Longrightarrow (x, x) \in \theta_{i.rator} \\ \pi_{i.rator} = cl_\sigma \Longrightarrow \forall y \in \lceil \sigma \rceil, \ \exists x \text{ such that } (x, y) \in \theta_{i.rator}, \text{ and} \\ \forall (j, B) \in \phi_{i.rator} \\ \quad \begin{cases} Abs\,(j), \\ \mathcal{P}_{i.rand} \leq \mathcal{P}_{j.bv}, \\ \mathcal{P}_{j.body} \leq \mathcal{P}_i, \\ \theta_{j.bv} \subseteq \theta_{i.rator}, \\ (x, y) \in \theta_{j.bv} \cup \theta_i \Longrightarrow x \neq \llbracket j.bv \rrbracket, \text{ and} \\ (x, y) \in \theta_{i.rator} \cap (\theta_{j.body} - \{(\llbracket j.bv \rrbracket, \_)\}) \Longrightarrow \\ \qquad\qquad (x, x) \in \theta_{i.rator} \end{cases} \end{cases}$$

$$Cond\,(i) \Longrightarrow \begin{cases} A_{i.test} = A_{i.then} = A_{i.else} = A_i, \\ \mathcal{P}_{i.then} \leq \mathcal{P}_i, \text{ and} \\ \mathcal{P}_{i.else} \leq \mathcal{P}_i, \end{cases}$$

Figure 1: Local Constraints

3 Soundness of the annotations

Using the definitions of monovariance and local consistency from the earlier paper, we can now give a revised version of our soundness theorem.

Theorem 1 (Soundness) *Let Γ be a locally consistent, monovariant annotation map. Let Π be the protocol assignment defined by $\Pi(i) = \pi_i$. Let i be any occurrence index, and let ψ be an occurrence environment such that $\psi \overset{\Pi}{\models} A_i$. If $(i, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$, then*

$$(1) \quad (j, \psi') \overset{\Pi}{\models} (\phi_i, \pi_i, \theta_i), \text{ and}$$

$$(2) \quad \forall (x, y) \in \theta_i, \ x \in Dom(\psi), \ y \in Dom(\psi'), \textit{ and } \psi(x) = \psi'(y)$$

Once more, the proof is an induction on the structure of the proof that $(i, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$.

For consequent (1) (value satisfaction), note that the definition for a value satisfying a proposition has not changed, and does not involve the θ-component of the proposition. Therefore, the

$$Var\,(i) \Longrightarrow \Phi(i) = [\![i]\!]$$

$$Const\,(i) \Longrightarrow \Phi(i) = [\![i]\!]$$

$$Abs\,(i) \wedge \pi_i = id \Longrightarrow \Phi(i) = \lambda[\![i.bv]\!].\Phi(i.body)$$

$$\begin{aligned} Abs\,(i) \wedge \pi_i = cl_{\langle v_1,\ldots,v_n\rangle} \Longrightarrow \Phi(i) = \; & [(\lambda e v_1 \ldots v_n[\![i.bv]\!].\,\mathbf{destr}\; e\; (\lambda \vec{u}.\Phi(i.body))), [\vec{u}]] \\ & \text{where } e \text{ fresh} \\ & \text{and } \vec{u} = FV([\![i]\!]) - \{v_1,\ldots,v_n\} \end{aligned}$$

$$App\,(i) \wedge Const\,(i.rator) \Longrightarrow \Phi(i) = \Phi(i.rator)\;\; \Phi(i.rand)$$

$$App\,(i) \wedge \neg\, Const\,(i.rator) \wedge \pi_{i.rator} = id \Longrightarrow \Phi(i) = \Phi(i.rator)\;\; \Phi(i.rand)$$

$$\begin{aligned} & App\,(i) \wedge \neg\, Const\,(i.rator) \wedge \pi_{i.rator} = cl_{\langle v_1,\ldots,v_n\rangle} \Longrightarrow \\ & \qquad \Phi(i) = \mathbf{app}\; \Phi(i.rator)\; u_1\; \ldots\; u_n\; \Phi(i.rand) \\ & \qquad\qquad \text{where } \forall k,\; 1 \le k \le n,\; (u_k, v_k) \in \theta_{i.rator} \end{aligned}$$

$$Cond\,(i) \Longrightarrow \Phi(i) = \mathbf{if}\; \Phi(i.test)\; \mathbf{then}\; \Phi(i.then)\; \mathbf{else}\; \Phi(i.else)$$

Figure 2: Definition of the transform

real changes to the soundness proof involve consequent (2) (invariance). We present the proof for the three productions of the λ-calculus in the appendix.

Note that this analysis subsumes the earlier one. If at a given call site i, we were able to include a variable x in the invariance set for the operator, the new analysis may include (x, x) in $\theta_{i.rator}$. Exactly as before, we can omit x from any closure flowing to that call site, and supply x as an extra argument.

4 Revising the transformation; correctness

What do we have to change in the transformation? The transformation does essentially two things: forming closures, and supplying dynamic variables at call sites.

If we have a procedure with a protocol tag of $cl_{\langle v_1,\ldots,v_n\rangle}$, that means we can omit $\{v_1, \ldots, v_n\}$ from the procedure's closure. That is as before. Now, for a call site which has a protocol tag of $cl_{\langle v_1,\ldots,v_n\rangle}$, for each v_i, we must choose a pair (u, v_i) from the invariance relation, and supply u as a dynamic variable. We know such a pair exists by the local consistency conditions.

We give the new transformation in Figure 2. Again, the transform is extended to occurrence closures:

$$\hat{\Phi}(i, \psi) = \Phi(i)\{\hat{\Phi} \circ \psi\}$$

Given this transformation, we can give the correctness theorem, which is stated exactly as before.

Theorem 2 (Correctness) *Let Γ be a locally consistent, monovariant annotation map. Let Π be the protocol assignment defined by $\Pi(i) = \pi_i$. Let i be any occurrence index, and let ψ be an occurrence environment such that $\psi \overset{\Pi}{\models} A_i$. If*

$$(i, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$$

then

$$\hat{\Phi}(i, \psi) \underset{t}{\Longrightarrow} \hat{\Phi}(j, \psi')$$

Proof: (Sketch) The proof is by induction on the size of the derivation that $(i, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$. Details will appear in [1]. ∎

5 Conclusion

In our POPL'94 paper, we used the problem of selective and lightweight closure conversion as an occasion to describe a method for justifying program transformations using flow-based program analyses. By the work described here, we have shown that this method is robust enough to be extended to a generalized version of the closure conversion transformation. In [1], we will show how the same general methodology can be applied to two wholly different transformations, one in which call-by-name programs are transformed to call-by-value equivalents, and another in which copies of variables are eliminated.

A Appendix: Proof of Theorem 1

Proof Induction on the structure of the proof that $(i, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$

We show only the cases for the pure λ-calculus subset of Λ_{in}.

case $Var(i)$
Then $(i, \psi) \underset{oc}{\Longrightarrow} \psi([\![i]\!])$. Since the annotation map is locally consistent, $A_i(x) = (\phi_i, \pi_i, \theta_i)$. Since $\psi \overset{\Pi}{\models} A_i$, $\psi([\![i]\!]) \overset{\Pi}{\models} (\phi_i, \pi_i, \theta_i)$, and $(x, y) \in \theta_i$ implies $x \in Dom(\psi)$, $y \in Dom(\psi')$, and $\psi(x) = \psi'(y)$.

case $Abs(i)$
So $(i, \psi) \underset{oc}{\Longrightarrow} (i, \psi)$. First we show that $(i, \psi) \overset{\Pi}{\models} (\phi_i, \pi_i, \theta_i)$. By the definition of $\overset{\Pi}{\models}$, this holds if $\Pi(i) = \pi_i$ (true by definition of Π), and if there is some A such that $(i, A) \in \phi_i$ and $\psi \overset{\Pi}{\models} A$. The annotation map is locally consistent, so $\{(i, A_i)\} \subseteq A_i$, and by assumption, $\psi \overset{\Pi}{\models} A_i$. By the local consistency condition, we have $(x, y) \in \theta_i$ implies $x = y$ and $x \in Dom(A_i)$. Since $\psi \overset{\Pi}{\models} A_i$, $Dom(A_i) \subseteq Dom(\psi)$, and trivially, $\psi(x) = \psi(x)$.

case $App(i)$
Suppose we have the evaluation proof:

$$\frac{\begin{array}{l}(i.rator, \psi) \underset{oc}{\Longrightarrow} (j, \psi'), \quad Abs(j) \\ (i.rand, \psi) \underset{oc}{\Longrightarrow} (k, \psi'') \\ (j.body, \psi'[[\![j.bv]\!] \mapsto (k, \psi'')]) \underset{oc}{\Longrightarrow} (m, \psi''')\end{array}}{(i, \psi) \underset{oc}{\Longrightarrow} (m, \psi''')}$$

Then:

(1)	$\psi \overset{\Pi}{\models} A_{i.rator}, A_{i.rand}$	$A_{i.rator} = A_{i.rand} = A_i$
(2)	$(i.rator, \psi) \underset{oc}{\Longrightarrow} (j, \psi')$	proof that $(i, \psi) \underset{oc}{\Longrightarrow} (m, \psi''')$
(3)	$(j, \psi') \overset{\Pi}{\models} (\phi_{i.rator}, \pi_{i.rator}, \theta_{i.rator})$	IH re $\overset{\Pi}{\models}$ at $i.rator$
(4)	$(x, y) \in \theta_{i.rator} \Longrightarrow x \in Dom(\psi),\ y \in Dom(\psi'),$ $\psi(x) = \psi'(y)$	IH re invariance at $i.rator$
(5)	$(j, A_j) \in \phi_{i.rator}$	defs of $\overset{\Pi}{\models}$, monovariance, (3)
(6)	$\psi' \overset{\Pi}{\models} A_j$	by def of $\overset{\Pi}{\models}$
(7)	$(i.rand, \psi) \underset{oc}{\Longrightarrow} (k, \psi'')$	proof that $(i, \psi) \underset{oc}{\Longrightarrow} (m, \psi''')$
(8)	$(k, \psi'') \overset{\Pi}{\models} (\phi_{i.rand}, \pi_{i.rand}, \theta_{i.rand})$	IH re $\overset{\Pi}{\models}$ at $i.rand$
(9)	$(x, y) \in \theta_{i.rand} \Longrightarrow x \in Dom(\psi),\ y \in Dom(\psi''),$ $\psi(x) = \psi''(y)$	IH re invariance at $i.rand$
(10)	$\phi_{i.rand} \subseteq \phi_{j.bv}$	flow constraints
(11)	$\phi_{j.body} \subseteq \phi_i$	
(12)	$\pi_{i.rand} = \pi_{j.bv}$	closure tag constraints
(13)	$\pi_{j.body} = \pi_i$	
(14)	$\theta_{j.bv} \subseteq \theta_{i.rator}$	inv. rel. constraints
(15)	$\theta_{j.bv} \subseteq \theta_{i.rand}$	
(16)	$(x, y) \in \theta_{j.bv} \Longrightarrow x \neq \llbracket j.bv \rrbracket$	
(17)	$\theta_i \subseteq \theta_{j.body}$	
(18)	$\theta_i \subseteq \theta_{i.rator}$	
(19)	$(x, y) \in \theta_i \Longrightarrow x \neq \llbracket j.bv \rrbracket$	
(20)	$(x, y) \in (\theta_{i.rator} \cap \theta_{i.rand}) \Longrightarrow (x, x) \in \theta_{i.rator}$	
(21)	$(x, y) \in (\theta_{i.rator} \cap \theta_{i.rand}) \Longrightarrow x \in Dom(\psi) \cap Dom(\psi'),$ $\psi(x) = \psi'(x)$	by (4), (20)
(22)	$(x, y) \in (\theta_{i.rator} \cap \theta_{i.rand}) \Longrightarrow x \in Dom(\psi) \cap Dom(\psi'),$ $y \in Dom(\psi') \cap Dom(\psi''),$ $\psi(x) = \psi'(x) =$ $\psi'(y) = \psi''(y)$	by (4), (9), (21)
(23)	$(x, y) \in \theta_{j.bv} \Longrightarrow x \in Dom(\psi) \cap Dom(\psi'),$ $y \in Dom(\psi') \cap Dom(\psi''),$ $\psi(x) = \psi'(x) =$ $\psi'(y) = \psi''(y)$	by (14), (15), (22)

$$
\begin{array}{lll}
(24) & (x,y) \in \theta_{j.bv} \implies x \in Dom(\psi'[[\![j.bv]\!] \mapsto (k,\psi'')]), & \text{by } (16),\ (23) \\
 & \qquad y \in Dom(\psi''), & \\
 & \qquad \psi'[[\![j.bv]\!] \mapsto (k,\psi'')](x) = \psi''(y) & \\
(25) & (k,\psi'') \overset{\Pi}{\models} (\phi_{j.bv}, \pi_{j.bv}, \theta_{j.bv}) & \text{by } (8),\ (10),\ (12) \\
(26) & \psi'[[\![j.bv]\!] \mapsto (k,\psi'')] \overset{\Pi}{\models} A_j[[\![j.bv]\!] \mapsto (\phi_{j.bv}, \pi_{j.bv}, \theta_{j.bv})] & \text{by } (6),\ (24),\ (25) \\
(27) & A_{j.body} = A_j[[\![j.bv]\!] \mapsto (\phi_{j.bv}, \pi_{j.bv}, \theta_{j.bv})] & \text{def of consistent anno} \\
(28) & \psi'[[\![j.bv]\!] \mapsto (k,\psi'')] \overset{\Pi}{\models} A_{j.body} & (26),\ (27) \\
(29) & (m,\psi''') \overset{\Pi}{\models} (\phi_{j.body}, \pi_{j.body}, \theta_{j.body}) & \text{IH re } \overset{\Pi}{\models} \text{ at } \textit{j.body} \\
(30) & (x,y) \in \theta_{j.body} \implies x \in Dom(\psi'[[\![j.bv]\!] \mapsto (k,\psi'')]), & \text{IH re invariance at } \textit{j.body} \\
 & \qquad y \in Dom(\psi'''), & \\
 & \qquad \psi'[[\![j.bv]\!] \mapsto (k,\psi'')](x) = \psi'''(y) & \\
(31) & (m,\psi''') \overset{\Pi}{\models} (\phi_i, \pi_i, \theta_i) & (11),\ (13), \text{and } (29) \\
(32) & (x,y) \in (\theta_{j.body} - \{([\![j.bv]\!], \_)\}) \implies x \in Dom(\psi'),\ y \in Dom(\psi'''), & \text{by } (30) \\
 & \qquad \psi'(x) = \psi'''(y) & \\
(33) & (x,y) \in (\theta_{i.rator} \cap (\theta_{j.body} - \{([\![j.bv]\!], \_)\})) \implies & \text{by } (4),\ (32) \\
 & \qquad x \in Dom(\psi),\ y \in Dom(\psi'''), & \\
 & \qquad \psi(x) = \psi'(y), & \\
 & \qquad \psi'(x) = \psi'''(y) & \\
(34) & (x,y) \in (\theta_{i.rator} \cap (\theta_{j.body} - \{([\![j.bv]\!], \_)\})) \implies (x,x) \in \theta_{i.rator} & \text{inv. rel. constraints} \\
(35) & (x,y) \in (\theta_{i.rator} \cap (\theta_{j.body} - \{([\![j.bv]\!], \_)\})) \implies \psi(x) = \psi'(x) & \text{by } (4),\ (34) \\
(36) & (x,y) \in (\theta_{i.rator} \cap (\theta_{j.body} - \{([\![j.bv]\!], \_)\})) \implies & \text{by } (33),\ (35) \\
 & \qquad x \in Dom(\psi),\ y \in Dom(\psi'''), & \\
 & \qquad \psi(x) = \psi'''(y) & \\
(37) & (x,y) \in \theta_i \implies y \in Dom(\psi) \cap Dom(\psi'''), & \text{by } (17),\ (18),\ (19),\ (36) \\
 & \qquad \psi(x) = \psi'''(y) &
\end{array}
$$

References

[1] Paul A. Steckler. *Correctness of Flow-Based Program Transformations for Higher-Order Programming Languages.* PhD thesis, Northeastern University, 1994 (to appear).

[2] Mitchell Wand and Paul Steckler. Selective and Lightweight Closure Conversion. In *Conf. Rec. 21st ACM Symposium on Principles of Programming Languages*, pages 435–45, 1994.