Notes: Introspective Flow Analysis with Granular Tracking of Imprecision

tgilray@cs.umd.edu

These notes develop an approach for tracking the sources of imprecision in a flow analysis by accumulating, alongside each abstract value, a set of the places where that value becomes conflated with another—each abstract value remembers the join points, *in the anlaysis*, leading to its precision loss. The idea is thus to instrument static information about program values at run-time with dynamic information about the analysis' treatment of those values at analysis-time. The goal of this approach is a dynamic analysis of a static analysis, computed at constant asymptotic overhead, which provides good, inexpensive information for incrementally adding polyvariance/context-sensitivity in a targeted fashion.

This general approach for tracking imprecision should be equally applicable to a wide variety of static analyses. Discrete, finite flow analyses such as closure analysis or points-to analysis are ideal for adding instrumentations that track sources of imprecision in this manner. Perhaps many other kinds of analyses will be suitable as well. In a traditional higher-order closure analysis, abstract closures are pairs of a syntactic lambda and a binding environment (a map from free variables to polyvariant/context-sensitive addresses). Analysis of a functional language *after* bottom-up closure conversion, results in closures made from pairing a lambda with a single, flat heap context—as opposed to a mapping with per-variable context [7]. In the case of points-to analysis, abstract objects, which pair a syntactic allocation site (and so class name) with a flat heap context, are the object-oriented analogue of abstract closures used in the descriptions below.

The big picture Consider the fragment of a global-store-widened closure analysis shown in Figure 1. Seven control-flow-graph nodes $(e_1, \hat{\rho}_1), \ldots, (e_7, \hat{\rho}_7)$ (each specific to a program expression e and binding environment $\hat{\rho}$) are shown, annotated with the bindings made across edges. For example, $\hat{a}_1 \mapsto \{\widehat{clo}_1\}$ is shown between the nodes for e_1 and e_4, which illustrates that this edge only contributes closure $\widehat{clo}_1$ at address $\hat{a}_1$. Down the middle, a sequence of four configurations propagate the closure $\widehat{clo}_1$ from address to address. At each of the last three configurations, two different source configurations (callers) make a binding to the same address when stepping into the function. The first of these conflates the values $\widehat{clo}_1$ and $\widehat{clo}_2$ at address $\hat{a}_1$. The second of these joins $\widehat{clo}_1$ with itself at address $\hat{a}_2$. The third is another genuine conflation of abstract values joining $\widehat{clo}_3$ at address $\hat{a}_3$. Then, let us say that $\hat{a}_3$ is the address for the variable in call position of e_7—its imprecision being particularly impactful.

Each of the join points in an abstract transition graph is a place in the analysis which might have been more precise. For example, the second transition to $(e_4, \hat{\rho}_4)$, from $(e_2, \hat{\rho}_2)$, might instead have been made using a more precise allocator to yield greater polyvariance/context-sensitivity. Instead of producing the same environment for e_4 as before, this transition might produce a new variant of $\hat{\rho}_4$, $\hat{\rho}'_4$, an environment specifically for the values in scope in e_4 *after* the call site e_2, as opposed to e_1 or any other.

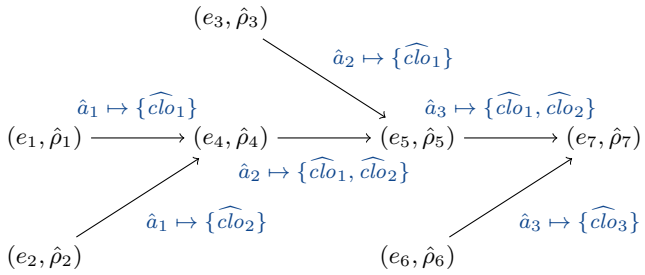

Figure 1. A fragment of an analysis showing an abstract transition graph and the flows of a closure analysis where several values become conflated.

To apply this additional polyvariance in a targeted fashion, we would like to approximate the locations at analysis-time where a lack of polyvariance led to spurious values elsewhere in the store or spurious edges elsewhere in the graph. In our example, the final address $\hat{a}_3$ is a location with potentially spurious values we especially want to refine. The analysis finds that three possible closures may reach $\hat{a}_3$, any, some, or none of which may be spurious flows (we cannot know which, if any, without a better *static* analysis).

$$\hat{a}_3 \mapsto \left\{\widehat{clo}_1, \widehat{clo}_2, \widehat{clo}_3\right\}$$

The idea is to effectively replace this flow set of closures, with a map from closures to a set of the join points which are relevant to its inclusion at this address.

$$\hat{a}_3 \mapsto \left[\begin{array}{ll} \widehat{clo}_1 \mapsto & \{(e_4, \hat{\rho}_4), (e_7, \hat{\rho}_7)\}, \\ \widehat{clo}_2 \mapsto & \{(e_4, \hat{\rho}_4), (e_5, \hat{\rho}_5), (e_7, \hat{\rho}_7)\}, \\ \widehat{clo}_3 \mapsto & \{(e_7, \hat{\rho}_7)\} \end{array}\right]$$

As this is a partial function, it still encodes the original flow set as its domain. Paired with each element in the domain is a set of the preceding join points, at which the element became conflated with another. The mapping for $\widehat{clo}_3$ indicates that the only place it becomes conflated with other closures on its way to $\hat{a}_3$ is when transitioning to $(e_7, \hat{\rho}_7)$. The mapping for $\widehat{clo}_1$ shows both $(e_4, \hat{\rho}_4)$ and $(e_7, \hat{\rho}_7)$ as places it becomes conflated with other values. The mapping for $\widehat{clo}_2$ is asymmetric with that of $\widehat{clo}_1$ because it cares that $\widehat{clo}_1$ is reinforced at two distinct points, while $\widehat{clo}_1$ does not.

An extensible closure analysis The general principle of tracking sources of imprecision this way should be broadly applicable, but it is especially simple to formalize on the CPS λ-calculus in Figure 2 in the context of a closure analysis. After applying the AAM process [6, 10], we obtain an analysis with the abstract domains in

Figure 3, the abstract atomic-expression evaluator in Figure 4, an allocator for 0-CFA in Figure 5, and a single small-step operational evaluation rule shown in Figure 6.

$$
\begin{aligned}
e \in \mathsf{Exp} &::= (\text{æ æ } \ldots) \mid \texttt{halt} \\
\text{æ} \in \mathsf{AExp} &::= lam \mid x \\
lam \in \mathsf{Lam} &::= (\lambda\ (x\ \ldots)\ e) \\
x \in \mathsf{Var} &::= \langle \textit{the set of program variables} \rangle
\end{aligned}
$$

Figure 2. A CPS λ-calculus language.

$$
\begin{aligned}
\hat{\varsigma} \in \hat{\Sigma} &\triangleq \mathsf{Exp} \times \widehat{Env} \times \widehat{Store} \times \hat{I} \\
\hat{\rho} \in \widehat{Env} &\triangleq \mathsf{Var} \rightharpoonup \widehat{Addr} \\
\hat{\sigma} \in \widehat{Store} &\triangleq \widehat{Addr} \to \widehat{Vals} \\
\hat{v} \in \widehat{Vals} &\triangleq \mathcal{P}\left(\widehat{Clo}\right) \\
\widehat{clo} \in \widehat{Clo} &\triangleq \mathsf{Lam} \times \widehat{Env} \\
\hat{a} \in \widehat{Addr} &\triangleq \langle \textit{a finite set determined by } \widehat{alloc} \rangle \\
\hat{\imath} \in \hat{I} &\triangleq \langle \textit{a finite set determined by } \rightsquigarrow_\iota \rangle
\end{aligned}
$$

Figure 3. Abstract domains for an AAM of Exp.

$$
\begin{aligned}
\hat{\mathcal{A}} &\subseteq \mathsf{AExp} \times \hat{\Sigma} \rightharpoonup \widehat{Vals} \\
\hat{\mathcal{A}}(x, (e, \hat{\rho}, \hat{\sigma})) &\triangleq \hat{\sigma}(\hat{\rho}(x)) \\
\hat{\mathcal{A}}(lam, (e, \hat{\rho}, \hat{\sigma})) &\triangleq \{(lam, \hat{\rho})\}
\end{aligned}
$$

Figure 4. An abstract atomic-expression evaluator.

The domains shown in Figure 3 include a parameter $\hat{\imath}$ for instrumentation data at each state. This follows the formulation for tuning polyvariance using allocation [5] and makes it possible to extend the analysis arbitrarily by swapping in a new instrumentation relation $\rightsquigarrow_\iota$. This relation is a total function which takes all components of a predecessor state and its successor state across a transition, except for the target state's instrumentation data, and yields a set of possible instrumentation data. A unique target results for each.

Figure 5 shows an allocator for 0-CFA, and, as this requires no special instrumentation (e.g., for tracking call histories), an empty instrumentation that always results in an unused value, ().

$$
\begin{aligned}
\widehat{alloc} &\subseteq \mathsf{Var} \times \hat{\Sigma} \to \widehat{Addr} \\
\widehat{alloc}(x, \hat{\varsigma}) &\triangleq x
\end{aligned}
$$

$$
\begin{aligned}
(\rightsquigarrow_\iota) &\subseteq \hat{\Sigma} \times \mathsf{Exp} \times \widehat{Env} \times \widehat{Store} \to \mathcal{P}\left(\hat{I}\right) \\
\hat{\varsigma} &\rightsquigarrow_\iota (e', \hat{\rho}', \hat{\sigma}', ())
\end{aligned}
$$

Figure 5. An allocator and instrumentation to instantiate 0-CFA.

We use the following syntactic sugar to emphasize that the instrumentation can be an arbitrarily sophisticated analysis itself:

$$\hat{\imath}' \in (\rightsquigarrow_\iota)(\hat{\varsigma}, e', \hat{\rho}', \hat{\sigma}') \iff \hat{\varsigma} \rightsquigarrow_\iota (e', \hat{\rho}', \hat{\sigma}', \hat{\imath}')$$

In Figure 6, this notation is used to defer to $(\rightsquigarrow_\iota)$ on the possible instrumentation data to reach in succeeding states.

$$
\begin{aligned}
&\overbrace{((\text{æ}_f\ \text{æ}_0\ \ldots\ \text{æ}_j), \hat{\rho}, \hat{\sigma}, \hat{\imath})}^{\hat{\varsigma}} \rightsquigarrow (e', \hat{\rho}', \hat{\sigma}', \hat{\imath}'), \text{where} \\
&\quad ((\lambda\ (x_0\ \ldots x_j)\ e'),\ \hat{\rho}_\lambda) \in \hat{\mathcal{A}}(\text{æ}_f, \hat{\varsigma}) \\
&\qquad\qquad \hat{\rho}' = \hat{\rho}_\lambda[x_i \mapsto \hat{a}_i] \\
&\qquad\qquad \hat{\sigma}' = \hat{\sigma} \sqcup [\hat{a}_i \mapsto \hat{\mathcal{A}}(\text{æ}_i, \hat{\varsigma})] \\
&\qquad\qquad \hat{a}_i = \widehat{alloc}(x_i, \hat{\varsigma}) \\
&\qquad\qquad \hat{\varsigma} \rightsquigarrow_\iota (e', \hat{\rho}', \hat{\sigma}', \hat{\imath}')
\end{aligned}
$$

Figure 6. Small-step operational semantics.

This analysis could also be reformulated with a global store or any other form of store widening without that changing the expressiveness of $\widehat{alloc}$ to tune analysis polyvariance [5]. Likewise, where the instrumentation data is something expensive, like a map, it can be widened or structurally made global for a less expensive analysis. We may also instantiate a sub-0-CFA with $O(n \log n)$ complexity by using value lattices of $O(1)$ height [1]. This could be formalized as a widening operator [2–4]; normally used to speed convergence within infinite abstract domains and guarantee termination, widening operators can also be used to improve efficiency for finite abstract domains. The operator $(\triangledown^k_{\text{sub}})$ can be used in place of a normal join, it jumps to $\top$ upon exceeding some constant k.

$$\hat{v}\ \triangledown^k_{\text{sub}}\ \hat{v}' \triangleq \begin{cases} \hat{v} \sqcup \hat{v}' & |\hat{v} \sqcup \hat{v}'| \leq k \\ \top & \textit{otherwise} \end{cases}$$

As the least-upper-bound of two stores distributes point-wise, so does lifting this to a widening operator for stores.

$$
\begin{aligned}
\hat{\sigma} \sqcup \hat{\sigma}' &\triangleq \lambda\hat{a}.\hat{\sigma}(\hat{a}) \sqcup \hat{\sigma}'(\hat{a}) \\
\hat{\sigma}\ \triangledown^k_{\text{sub}}\ \hat{\sigma}' &\triangleq \lambda\hat{a}.\hat{\sigma}(\hat{a})\ \triangledown^k_{\text{sub}}\ \hat{\sigma}'(\hat{a}))
\end{aligned}
$$

Effectively, this cuts off the top of the powerset lattice of values $\hat{v}$, decreasing complexity, but also precision. This technique may not be desirable for a particular static analysis, but the same essential idea will be important for keeping the instrumentation we add next inexpensive.

An instrumentation for tracking imprecision Now consider defining an instrumentation, not for empowering a more precise abstract allocator directly, but for tracking sources of imprecision so the allocator may be made more precise as a following step. This instrumentation recapitulates the value store with additional information. At an address $\hat{a}$, it maintains two maps, each from closures (those which flow to $\hat{a}$) to sets $\hat{s}$ of configurations $(e, \hat{\rho})$.

$$
\begin{aligned}
\hat{\imath} \in \hat{I} &\triangleq \widehat{Addr} \rightharpoonup \widehat{IVals} \times \widehat{IVals} \\
\widehat{iv} \in \widehat{IVals} &\triangleq \widehat{Clo} \rightharpoonup \widehat{CSet} \\
\hat{s} \in \widehat{CSet} &\triangleq \mathcal{P}\left(\mathsf{Exp} \times \widehat{Env}\right)
\end{aligned}
$$

The first of these maps contains an approximation of the join points where each $\widehat{clo}$ becomes conflated with other values on the way to reaching $\hat{a}$. The second of these maps tracks the set of configurations that locally contribute a closure. The first map contains what the analysis is really after, but the second map helps to formalize the analysis. For example, in Figure 1, the configuration $(e_5, \hat{\rho}_5)$ has two configurations which contribute values to $\hat{a}_2$. The second map which tracks local contributors would map $\widehat{clo}_1$ to both $(e_3, \hat{\rho}_3)$

and $(e_4, \hat{\rho}_4)$, while it maps $\widehat{clo}_2$ only to $(e_4, \hat{\rho}_4)$. This means that $(e_5, \hat{\rho}_5)$ is a join point for $\widehat{clo}_2$, but not for $\widehat{clo}_1$.

Figure 7 shows an atomic-expression evaluator for instrumentations. Just as the normal atomic evaluator $\hat{\mathcal{A}}$, this takes an æ and a current state $\hat{\varsigma}$; however, instead of returning a set of closures, A_ι will return two maps of closures to their respective instrumentation data. This $\hat{\mathcal{A}}_\iota$ is responsible for maps of local contributors in the analysis; these in turn enable dection of join points by the instrumentation function (in this formulation of imprecision tracking).

$$\hat{\mathcal{A}}_\iota \subseteq \mathsf{AExp} \times \hat{\Sigma} \rightharpoonup \widehat{IVals} \times \widehat{IVals}$$
$$\hat{\mathcal{A}}_\iota(x, (e, \hat{\rho}, \hat{\sigma}, \hat{\iota})) \triangleq ([\ldots, \widehat{clo}_i \mapsto \hat{s}_i, \ldots], [\ldots, \widehat{clo}_i \mapsto \{(e, \hat{\rho})\}, \ldots]), \text{where } \hat{\iota}(\hat{\rho}(x)) = ([\ldots, \widehat{clo}_i \mapsto \hat{s}_i, \ldots], \_)$$
$$\hat{\mathcal{A}}_\iota(lam, (e, \hat{\rho}, \hat{\sigma}, \hat{\iota})) \triangleq ([(lam, \hat{\rho}) \mapsto \varnothing], [(lam, \hat{\rho}) \mapsto \{(e, \hat{\rho})\}])$$

Figure 7. An atomic-evaluator for instrumentation data.

In the case of closure creation, a pair of two maps with a single point each is returned where the set of long-term join points is empty and the set of local contributors contains only the current expression and environment $(e, \hat{\rho})$. In the case of variable reference, the existing maps are extracted from the store at the address in $\hat{\rho}$. The first map is returned unmodified, the second is updated so that every set of local contributors is replaced with a singleton set where only the current configuration is a local contributor. Thus, as instrumentation data is retrieved and propagated, local contributor sets (the second maps) are updated to reflect the local configurations coming together at each point. This enables join points, where different values become conflated, to be identified; the long-term sets of transitive join points (the first maps) are then faithfully moved and extended.

Figure 8 shows the instrumentation relation $(\rightsquigarrow_\iota)$. Just as in the core semantics, a propositional statement fixes the closure being applied across the transition; the successor components e', $\hat{\rho}'$ and $\hat{\sigma}'$ are provided as inputs to $(\rightsquigarrow_\iota)$, constrained by the relation $(\rightsquigarrow)$ in its evaluation context. The function $\hat{\mathcal{A}}_\iota$ is used to retrieve and propagate instrumentation data in much the same way as for values in the core semantics of Figure 6. Joins distribute point-wise for functions as before; joins distribute element-wise for tuples.

A helper function $J_{(e,\hat{\rho})}$ is used to identify, from the map of local contributors, for which closures is $(e, \hat{\rho})$ a join point. The left-hand side of the set-builder notation is a point for each satisfying $\hat{a}$ and $\widehat{clo}$ associated with a singleton set containing the current join point $(e, \hat{\rho})$ to be added. The right-hand side of the set-builder notation states that $\hat{a}$ is an address in $\hat{\iota}$, names the two maps at this address, and states that $\widehat{clo}$ flows to both. It then states this is a join point for $\widehat{clo}$: there must exist some other closure $\widehat{clo}'$ with a contributor $(e', \hat{\rho}')$ that does not contribute $\widehat{clo}$.

For example, in Figure 1, the closure $\widehat{clo}_1$, at the address $\hat{a}_2$, has two local contributors while the closure $\widehat{clo}_2$ has only one. The former set of configurations contains the latter, meaning the invocation of $J_{(e_5,\hat{\rho}_5)}$ will determine that when $\widehat{clo} = \widehat{clo}_2$ and $\widehat{clo}' = \widehat{clo}_1$, the contributor $(e', \hat{\rho}') = (e_3, \hat{\rho}_3)$ is a contributor for $\widehat{clo}_1$ but not for $\widehat{clo}_2$ (last few lines in Figure 8), and so $(e_5, \hat{\rho}_5)$ is a join point for $\widehat{clo}_2$, but not for $\widehat{clo}_1$.

The *crux* of this technique is that although these sets of join points may become large, as they do, they also become more expensive to fix and lose relevance. The set of local contributors for some closure is bounded by the number of incoming edges which bind that abstract address. The set of global join-points may grow to include a number of configurations proportional to the total number in the analysis. To avoid this expense, we may apply the same essential technique embodied by the sub-0-CFA widening operator as shown in Figure 9. Although the set of join points causing imprecision at a closure and address is not soundly conservative in any way (neither is fixing every join point guarenteed to be sufficient, nor is fixing any single join point guarenteed not to be), the hypothesis is that this set will tend to approximate how many of which program points must be fixed and therefore especially large sets may be widened to $\top$.

$$\overbrace{((\text{æ}_f\ \text{æ}_0\ \ldots \text{æ}_j), \hat{\rho}, \hat{\sigma}, \hat{\iota})}^{\hat{\varsigma}} \rightsquigarrow_\iota (e', \hat{\rho}', \hat{\sigma}', \hat{\iota}'), \text{where}$$
$$((\lambda\ (x_0 \ldots x_j)\ e'),\ \hat{\rho}_\lambda) \in \hat{\mathcal{A}}(\text{æ}_f, \hat{\varsigma})$$
$$\hat{\iota}' = J_{(e',\hat{\rho}')}(\hat{\iota} \sqcup [\hat{a}_i \mapsto \hat{\mathcal{A}}_\iota(\text{æ}_i, \hat{\varsigma})])$$
$$\hat{a}_i = \widehat{alloc}(x_i, \hat{\varsigma})$$

$$J_{(e,\hat{\rho})}(\hat{\iota}) \triangleq \hat{\iota} \sqcup \bigsqcup \left\{ [\hat{a} \mapsto ([\widehat{clo} \mapsto \{(e, \hat{\rho})\}], \bot)] \;\middle|\; \begin{array}{l} [\hat{a} \mapsto (\widehat{iv}_{\text{jns}}, \widehat{iv}_{\text{lcl}})] \in \hat{\iota} \\ \wedge\ \widehat{clo} \in dom(\widehat{iv}_{\text{jns}}) \wedge \widehat{clo} \in dom(\widehat{iv}_{\text{lcl}}) \wedge \\ \exists \widehat{clo}', e', \hat{\rho}', \hat{s}, \hat{s}'. (\widehat{clo} \neq \widehat{clo}' \\ \quad \wedge\ [\widehat{clo} \mapsto \hat{s}] \in \widehat{iv}_{\text{lcl}} \wedge (e', \hat{\rho}') \notin \hat{s} \\ \quad \wedge\ [\widehat{clo}' \mapsto \hat{s}'] \in \widehat{iv}_{\text{lcl}} \wedge (e', \hat{\rho}') \in \hat{s}') \end{array} \right\}$$

Figure 8. Imprecision-tracking instrumentation semantics. $J_{(e,\hat{\rho})}$ is a helper function that adds join points in the instrumentation when it observes the conditions are met by looking at local contributors.

$$\hat{\iota} \triangledown_\iota^k \hat{\iota}' \triangleq \hat{\iota} \sqcup \hat{\iota}' \sqcup \bigsqcup \left\{ [\hat{a} \mapsto ([\widehat{clo} \mapsto \top], \bot)] \;\middle|\; \begin{array}{l} [a \mapsto (\widehat{iv}_{\text{jns}}, \_)] \in (\hat{\iota} \sqcup \hat{\iota}') \wedge \\ [\widehat{clo} \mapsto \hat{s}] \in \widehat{iv}_{\text{jns}} \wedge |\hat{s}| > k \end{array} \right\}$$

Figure 9. A widening operator that limits complexity to k join points per closure.

Related work The closest related work seems to be the introspective context-sensitivity of Smaragdakis, et al. [9] and the impact pre-analysis of Oh, et al. [8]. Both are approaches to performing a multi-phase analysis where an initial phase is used to guide the tuning of polyvariance for real analysis. Smaragdakis et al. targets potentially expensive points in an analysis to be run context insensitively while the rest of the analysis uses a modest amount of context. Oh, et al. targets the places in a program where even a very high degree of context sensitivity will yield some degree of precision (and is thus also likely be inexpensive), leaving the remaining analysis context insensitive.

Smaragdakis, et al., performs a context-insensitive points-to analysis and then computes two different sets of aggregate statistics (e.g., the number of local variables pointing to each abstract object,

the total points-to size of an object, across fields, or the total points-to size of a method, across local variables). Each of these two ensemble metrics are then used to classify every allocation site and call site into either a context-sensitive or context-insensitive point in the code and recomputes the analysis. The goal of this adaptation is not to track the sources of imprecision in a granular way, but to identify the places where large points-to sets make it possible that the context-sensitive analysis will be particularly expensive. These hotspots with large numbers of flows are left context-insensitive, while the rest of the analysis is improved to some fixed degree of call or object sensitivity in a uniform way.

Oh, et al., describes a different approach where the client analysis is first coarsly approximated at a much higher degree of context-sensitivity, to see where the client analysis must be precise. The clients they use are an interval analysis and octagon analysis of C; in the former case, the impact pre-analysis distinguishes only two intervals: the infinite interval $\top$ and any finite interval $\star$. The analysis is then run with what the authors term "full" context sensitivity; this means they run a k-CFA with an unbounded k except where no call site may appear twice (to avoid cycles and thus unboundedness). In this way, they take an approach opposite to that of Smaragdakis, et al. where the analysis attempts to add a high degree of context sensitivity at the particular places where a pre-analysis shows even this very high degree of context sensitivity will yield some precision as opposed to none. This analysis is performed on a body of C code, and finds particular portions of code where the context sensitivity may be dialed up considerably without yielding imprecision. As imprecision and complexity explosions tend to go hand in hand, the authors find experimentally that analysis times only increase modestly in proportion with precision.

Neither of these approaches tracks where imprecision occurs in the analysis at hand in a fine-grained way. Smaragdakis, et al., uses statistics from a context-insensitive analysis to avoid places where complexity is potentially high when performing the context-sensitive pass. Oh, et al., performs a very inexpensive analysis at a higher context sensitivity, supposing that where polyvariance is successful for such a coarse analysis, so too will it be inexpensive on the actual analysis.

Future work We are also able to exploit the fact that allocation can be tuned arbitrarily to break the mold of k-call sensitivity and k-obj sensitivity and other standard approaches to polyvariance. It seems likely that both the instrumentation proposed, and especially how it is utilized by the allocator on a second pass, will change as experiments start to guide this research. There are some situations where fixing the polyvariance at one point, only pushes a join back a single transition—it may be helpful to combine this imprecision tracking with some simple live heuristics in alloc.

References

[1] M. D. Adams. *Flow-Sensitive Control-Flow Analysis in Linear-Log Time*. PhD thesis, Indiana University, 2011.

[2] A. Cortesi. Widening operators for abstract interpretation. In *2008 Sixth IEEE International Conference on Software Engineering and Formal Methods*, pages 31–40. IEEE, 2008.

[3] A. Cortesi and M. Zanioli. Widening and narrowing operators for abstract interpretation. *Computer Languages, Systems & Structures*, 37(1):24–42, 2011.

[4] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the Second International Symposium on Programming*, pages 106–130. Paris, France, 1976.

[5] T. Gilray, M. D. Adams, and M. Might. Allocation characterizes polyvariance: A unified methodology for polyvariant control-flow analysis. In *Proceedings of the International Conference on Functional Programming*, September 2016.

[6] M. Might. Abstract interpreters for free. In *Static Analysis Symposium*, pages 407–421, September 2010.

[7] M. Might, Y. Smaragdakis, and D. Van Horn. Resolving and exploiting the k-CFA paradox: Illuminating functional vs. object-oriented program analysis. In *Proceedings of the International Conference on Programming Language Design and Implementation*, pages 305–315, June 2010.

[8] H. Oh, W. Lee, K. Heo, H. Yang, and K. Yi. Selective context-sensitivity guided by impact pre-analysis. *ACM SIGPLAN Notices*, 49(6):475–484, 2014.

[9] Y. Smaragdakis, G. Kastrinis, and G. Balatsouras. Introspective analysis: context-sensitivity, across the board. In *ACM SIGPLAN Notices*, volume 49, pages 485–495. ACM, 2014.

[10] D. Van Horn and M. Might. Abstracting abstract machines. In *International Conference on Functional Programming*, page 51, Sep 2010.