Purity Analysis: An Abstract Interpretation Formulation

Ravichandhran Madhavan, G. Ramalingam, and Kapil Vaswani

Microsoft Research, India.
t-rakand,grama,kapilv@microsoft.com

Abstract. Salcianu and Rinard present a compositional purity analysis that computes a summary for every procedure describing its side-effects. In this paper, we formalize a generalization of this analysis as an abstract interpretation, present several optimizations and an empirical evaluation showing the value of these optimizations. The Salcianu-Rinard analysis makes use of abstract heap graphs, similar to various heap analyses and computes a shape graph at every program point of an analyzed procedure. The key to our formalization is to view the shape graphs of the analysis as an *abstract state transformer* rather than as a set of abstract states: the concretization of a shape graph is a function that maps a concrete state to a set of concrete states. The abstract interpretation formulation leads to a better understanding of the algorithm. More importantly, it makes it easier to change and extend the basic algorithm, while guaranteeing correctness, as illustrated by our optimizations.

1 Introduction

Compositional or modular analysis [6] is a key technique for scaling static analysis to large programs. Our interest is in techniques that analyze a procedure in isolation, using pre-computed summaries for called procedures, computing a summary for the analyzed procedure. Such analyses are widely used and have been found to scale well. In this paper we consider an analysis presented by Salcianu and Rinard [17], based on a pointer analysis due to Whaley and Rinard [19], which we will refer to the WSR analysis. Though referred to as a purity analysis, it is a more general-purpose analysis that computes a summary for every procedure, in the presence of dynamic memory allocation, describing its side-effects. This is one of the few heap analyses that is capable of treating procedures in a compositional fashion.

WSR analysis is interesting for several reasons. Salcianu and Rinard present an application of the analysis to classify a procedure as *pure* or *impure*, where a procedure is impure if its execution can potentially modify pre-existing state. Increasingly, new language constructs (such as iterators, parallel looping constructs and SQL-like query operators) are realized as higher-order library procedures with procedural parameters that are expected to be side-effect free. Purity checkers can serve as verification/bug-finding tools to check usage of these constructs. Our interest in this analysis stems from our use of an extension of

this analysis to statically verify the correctness of the use of speculative parallelism [13]. WSR analysis can also help more sophisticated verification tools, such as [8], which use simpler analyses to identify procedure calls that do not affect properties of interest to the verifier and can be abstracted away.

However, we felt the need for various extensions of the WSR analysis. A key motivation was efficiency. Real-world applications make use of large libraries such as the base class libraries in .NET. While the WSR analysis is reasonably efficient, we find that it still does not scale to such libraries. Another motivation is increased functionality: our checker for speculative parallelism [13] needs some extra information (must-write sets) beyond that computed by the analysis. A final motivating factor is better precision: the WSR analysis declares "pure" procedures that use idioms like lazy initialization and caching as impure.

The desire for these extensions leads us to formulate, in this paper, the WSR analysis as an abstract interpretation, to simplify reasoning about the soundness of these extensions. The formulation of the WSR analysis as an abstract interpretation is, in fact, mentioned as an open problem by Salcianu ([16], page 128).

The WSR analysis makes use of abstract heap graphs, similar to various heap analyses and computes a shape graph g_u at every program point u of an analyzed procedure. The key to our abstract interpretation formulation, however, is to view a shape graph utilized by the analysis as an *abstract state transformer* rather than as a set of abstract states: thus, the concretization of a shape graph is a function that maps a concrete state to a set of concrete states. Specifically, if the graph computed at program point u is g_u, then for any concrete state σ, $\gamma(g_u)(\sigma)$ conservatively approximates the set of states that can arise at program point u in the execution of the procedure on an initial state σ. In our formalization, we present a concrete semantics in the style of the functional approach to interprocedural analysis presented by Sharir and Pnueli. The WSR analysis can then be seen as a natural abstract interpretation of this concrete semantics.

We then present three optimizations viz. duplicate node merging, summary merging, and safe node elimination, that improve the efficiency of WSR analysis. We use the abstract interpretation formulation to show that these optimizations are sound. Our experiments show that these optimizations significantly reduce both analysis time (sometimes by two orders of magnitude or more) and memory consumption, allowing the analysis to scale to large programs.

2 The Language, Concrete Semantics, And The Problem

Syntax A program consists of a set of procedures. A procedure P consists of a control-flow graph, with an entry vertex $entry(P)$ and an exit vertex $exit(P)$. The entry vertex has no predecessor and the exit vertex has no successor. Every edge of the control-flow graph is labelled by a primitive statement. The set of primitive statements are shown in Fig. 1. We use $u \xrightarrow{S} v$ to indicate an edge in the control-flow graph from vertex u to vertex v labelled by statement S.

Concrete Semantics Domain Let *Vars* denote the set of variable names used in the program, partitioned into the following disjoint sets: the set of global

Statement S	**Concrete semantics** $[\![S]\!]_c(\mathsf{V},\mathsf{E},\sigma)$
$v_1 = v_2$	$\{(\mathsf{V},\mathsf{E},\sigma[v_1 \mapsto \sigma(v_2)]\}$
$v = new\ C$	$\{(\mathsf{V} \cup \{n\}, \mathsf{E} \cup \{n\} \times \mathit{Fields} \times \{\mathit{null}\}, \sigma[v \mapsto n]) \mid n \in N_c \setminus \mathsf{V}\}$
$v_1.f = v_2$	$\{(\mathsf{V}, \{\langle u,l,v\rangle \in \mathsf{E} \mid u \neq \sigma(v_1) \vee l \neq f\} \cup \{\langle \sigma(v_1), f, \sigma(v_2)\rangle\}, \sigma)\}$
$v_1 = v_2.f$	$\{(\mathsf{V},\mathsf{E},\sigma[v_1 \mapsto n]) \mid \langle \sigma(v_2), f, n\rangle \in \mathsf{E}\}$
$\mathit{Call}\ P(v_1,\cdots,v_k)$	Semantics defined below

Fig. 1. Primitive statements and their concrete semantics

variables *Globals*, the set of local variables *Locals* (assumed to be the same for every procedure), and the set of formal parameter variables *Params* (assumed to be the same for every procedure). Let *Fields* denote the set of field names used in the program. We use a simple language in which all variables and fields are of pointer type. We use a fairly common representation of the concrete state as a concrete (points-to or shape) graph.

Let N_c be an unbounded set of locations used for dynamically allocated objects. A concrete state or points-to graph $g \in \mathbb{G}_c$ is a triple $(\mathsf{V},\mathsf{E},\sigma)$, where $\mathsf{V} \subseteq N_c$ represents the set of objects in the heap, $\mathsf{E} \subseteq \mathsf{V} \times \mathit{Fields} \times \mathsf{V}$ (a set of labelled edges) represents values of pointer fields in heap objects, and $\sigma \in \Sigma_c = \mathit{Vars} \mapsto \mathsf{V}$ represents the values of program variables. In particular, $(u,f,v) \in \mathsf{E}$ iff the f field of the object u points to object v. We assume N_c includes a special element *null*. Variables and fields of new objects are initialized to *null*.

Let $\mathcal{F}_c = \mathbb{G}_c \mapsto 2^{\mathbb{G}_c}$ be the set of functions that map a concrete state to a set of concrete states. We define a partial order $\sqsubseteq_c$ on $\mathcal{F}_c$ as follows: $f_a \sqsubseteq_c f_b$ iff $\forall g \in \mathbb{G}_c . f_a(g) \subseteq f_b(g)$. Let $\sqcup_c$ denote the corresponding least upper bound (join) operation defined by: $f_a \sqcup_c f_b = \lambda g. f_a(g) \cup f_b(g)$. For any $f \in \mathcal{F}_c$, we define $\overline{f} : 2^{\mathbb{G}_c} \mapsto 2^{\mathbb{G}_c}$ by: $\overline{f}(G) = \cup_{g \in G} f(g)$. We define the "composition" of two functions in $\mathcal{F}_c$ as follows: $f_a \circ f_b = \lambda g. \overline{f_b}(f_a(g))$.

Concrete Semantics Every primitive statement S has a semantics $[\![S]\!]_c \in \mathcal{F}_c$, as shown in Fig. 1. Every primitive statement has a label ℓ which is not used in the concrete semantics and is, hence, omitted from the figure. The execution of most statements transforms a concrete state to another concrete state, but the signature allows us to model non-determinism (e.g., dynamic memory allocation can return any unallocated object). The signature also allows us to model execution errors such as null-pointer dereference, though the semantics presented simplifies error handling by treating *null* as just a special object.

We now define a concrete summary semantics $[\![P]\!]_c \in \mathcal{F}_c$ for every procedure P. The semantic function $[\![P]\!]_c$ maps every concrete state g_c to the set of concrete states that the execution of P with initial state g_c can produce.

We introduce a new variable φ_u for every vertex in the control-flow graph (of any procedure) and a new variable $\varphi_{u,v}$ for every edge $u \to v$ in the control-flow graph. The semantics is defined as the least fixed point of the following set of equations. The value of φ_u in the least fixed point is a function that maps any concrete state g to the set of concrete states that arise at program point u when the procedure containing u is executed with an initial state g. Similarly, $\varphi_{u,v}$

captures the states after the execution of the statement labelling edge $u \to v$.

$$\varphi_v = \lambda g.\{g\} \qquad v \text{ is an entry vertex} \tag{1}$$

$$\varphi_v = \bigsqcup_c \{\varphi_{u,v} \mid u \to v\} \qquad v \text{ is not an entry vertex} \tag{2}$$

$$\varphi_{u,v} = \varphi_u \circ [\![S]\!]_c \qquad \text{where } u \xrightarrow{S} v \text{ and S is not a call-stmt} \tag{3}$$

$$\varphi_{u,v} = \varphi_u \circ \mathit{CallReturn}_S(\varphi_{\mathit{exit}(Q)}) \qquad \text{where } u \xrightarrow{S} v,\ \text{S is a call to proc Q} \tag{4}$$

The first three equations are straightforward. Consider Eq. 4, corresponding to a call to a procedure Q. The value of $\varphi_{\mathit{exit}(Q)}$ summarizes the effect of the execution of the whole procedure Q. In the absence of local variables and parameters, we can define the right-hand-side of the equation to be simply $\varphi_u \circ \varphi_{\mathit{exit}(Q)}$.

The function $\mathit{CallReturn}_S(f)$, defined below, first initializes values of all local variables (to *null*) and formal parameters (to the values of corresponding actual parameters), using an auxiliary function push_S. It then applies f, capturing the procedure call's effect. Finally, the original values of local variables and parameters (of the calling procedure) are restored from the state preceding the call, using a function pop_S. For simplicity, we omit return values from our language.

Let $\mathit{Param}(i)$ denote the i-the formal parameter. Let S be a procedure call statement "`Call Q(`a_1`,...,`a_k`)`". We define the functions $\mathit{push}_S \in \Sigma_c \mapsto \Sigma_c$, $\mathit{pop}_S \in \Sigma_c \times \Sigma_c \mapsto \Sigma_c$, and $\mathit{CallReturn}_S$ as follows:

$$\mathit{push}_S(\sigma) = \lambda v.\ v \in \mathit{Globals} \to \sigma(v) \mid v \in \mathit{Locals} \to \mathit{null} \mid v = \mathit{Param}(i) \to \sigma(a_i)$$

$$\mathit{pop}_S(\sigma, \sigma') = \lambda v.\ v \in \mathit{Globals} \to \sigma'(v) \mid v \in \mathit{Locals} \cup \mathit{Params} \to \sigma(v)$$

$$\mathit{CallReturn}_S(f) = \lambda(\mathsf{V}, \mathsf{E}, \sigma).\{(\mathsf{V}', \mathsf{E}', \mathit{pop}_S(\sigma, \sigma')) \mid (\mathsf{V}', \mathsf{E}', \sigma') \in f(\mathsf{V}, \mathsf{E}, \mathit{push}_S(\sigma))\}$$

We define $[\![P]\!]_c$ to be the value of $\varphi_{\mathit{exit}(P)}$ in the least fixed point of equations (1)-(4), which exists by Tarski's fixed point theorem. Specifically, let VE denote the set of vertices and edges in the control flow graph of a program. The above equations can be expressed as a single equation $\varphi = F^\natural(\varphi)$, where $F^\natural$ is a monotonic function from the complete lattice $VE \mapsto \mathcal{F}_c$ to itself. Hence, $F^\natural$ has a least fixed point.

We note that the above collection of equations is similar to those used in Sharir and Pnueli's functional approach to interprocedural analysis [18] (extended by Knoop and Steffen [11]), with the difference that we are defining a concrete semantics here, while [18] is focused on abstract analyses. The equations are a simple functional version of the standard equations for defining a collecting semantics, with the difference that we are simultaneously computing a collecting semantics for every possible initial states of the procedure's execution.

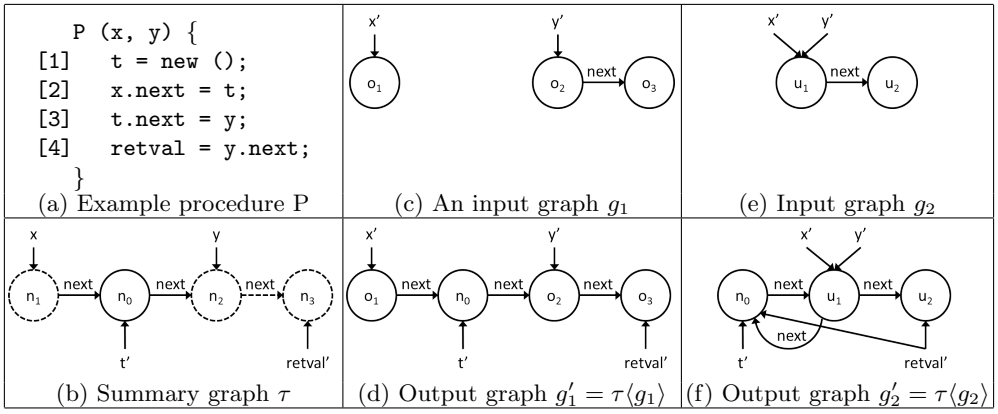
The goal of the analysis is to compute an approximation of the set of quantities $[\![P]\!]_c$ using abstract interpretation.

3 The WSR Analysis As An Abstract Interpretation

3.1 Transformer Graphs: An Informal Overview

The WSR analysis uses a single abstract graph to represent a set of concrete states, similar to several shape and pointer analyses. The distinguishing aspect

of the WSR analysis, however, is its extension of the graph based representation to represent (abstractions of) elements belonging to the functional domain $\mathcal{F}_c$. We now illustrate, using an example, how the graph representation is extended to represent an element of $\mathcal{F}_c = \mathbb{G}_c \mapsto 2^{\mathbb{G}_c}$. Consider the example procedure P shown in Fig. 2(a).



(a) Example procedure P (c) An input graph g_1 (e) Input graph g_2

(b) Summary graph τ (d) Output graph $g_1' = \tau\langle g_1 \rangle$ (f) Output graph $g_2' = \tau\langle g_2 \rangle$

Fig. 2. Illustration of transformer graphs.

The summary graph τ computed for this procedure is shown in Fig. 2(b). (We omit the *null* node from the figures to keep them simple.) Vertices in a summary graph are of two types: *internal* (shown as circles with a solid outline) and *external* nodes (shown as circles with a dashed outline). Internal nodes represent new heap objects created during the execution of the procedure. E.g., vertex n_0 is an internal node and represents the object allocated in line 1. External nodes, in many cases, represent objects that exist in the heap when the procedure is invoked. In our example, n_1, n_2, and n_3 are external nodes.

Edges in the graph are also classified into *internal* and *external* edges, shown as solid and dashed edges respectively. The edges $n_1 \rightarrow n_0$ and $n_0 \rightarrow n_2$ are internal edges. They represent updates performed by the procedure (i.e., new points-to edges added by the procedure's execution) in lines 2 and 3. Edge $n_2 \rightarrow n_3$ is an external edge created by the dereference "`y.next`" in line 4. This edge helps identify the node(s) that the external node n_3 represents: namely, the objects obtained by dereferencing the `next` field of objects represented by n_2.

The summary graph τ indicates how the execution of procedure P transforms an initial concrete state. Specifically, consider an invocation of procedure P in an initial state given by graph g_1 shown in Fig. 2(c). The summary graph helps construct a transformed graph $g_1' = \tau\langle g_1 \rangle$, corresponding to the state after the procedure's execution (shown in Fig. 2(d)) by identifying a set of new nodes and edges that must be added to g_1. (The underlying analysis performs no strong updates on the heap and, hence, never removes nodes or edges from the graph). We add a new vertex to g_1 for every internal node n in the summary graph.

Every external node n in the summary graph represents a set of vertices $\eta(n)$ in g'_1. (We will explain later how the function η is determined by τ.) Every internal edge $u \xrightarrow{h} v$ in the summary graph identifies a set of edges $\{u' \xrightarrow{h} v' \mid u' \in \eta(u), v' \in \eta(v)\}$ that must be added to the graph g'_1. In our example, n_1, n_2 and n_3 represent, respectively, $\{o_1\}$, $\{o_2\}$ and $\{o_3\}$. This produces the graph shown in Fig. 2(d), which is an *abstract* graph representing a set of concrete states. The primed variables in the summary graph represent the (final) values of variables, and are used to determine the values of variables in the output graph.

An important aspect of the summary computed by the WSR analysis is that it can be used even in the presence of potential aliases in the input (or cut-points [14]). Consider the input state g_2 shown in Fig. 2(e), in which parameters `x` and `y` point to the same object u_1. Our earlier description of how to construct the output graph still applies in this context. The main tricky aspect here is in correctly dealing with aliasing in the input. In the concrete execution, the update to `x.next` in line 2 updates the `next` field of object u_1. The aliasing between `x` and `y` means that `y.next` will evaluate to n_0 in line 4. Thus, in the concrete execution `retval` will point to the newly created object n_0 at the end of procedure execution, rather than u_2. This complication is dealt with in the definition of the mapping function η. For the example input g_2, the external node n_3 of the summary graph represents the set of nodes $\{u_2, n_0\}$. (This is an imprecise, but sound, treatment of the aliasing situation.) The rest of the construction applies just as before. This yields the abstract graph shown in Fig. 2(f).

More generally, an external node in the summary graph acts as a proxy for a set of *vertices in the final output graph to be constructed*, which may include nodes that exist in the input graph as well as new nodes added to the input graph (which themselves correspond to internal nodes of the summary graph).

We now define the transformer graph domain formally.

3.2 The Abstract Domain

The Abstract Graph Domain We utilize a fairly standard abstract shape (or points-to) graph to represent a set of concrete states. Our formulation is parameterized by a given set N_a, the universal set of all abstract graph nodes. An abstract shape graph $g \in \mathbb{G}_a$ is a triple $(\mathsf{V}, \mathsf{E}, \sigma)$, where $\mathsf{V} \subseteq N_a$ represents the set of abstract heap objects, $\mathsf{E} \subseteq \mathsf{V} \times \mathit{Fields} \times \mathsf{V}$ (a set of labelled edges) represents possible values of pointer fields in the abstract heap objects, and $\sigma \in \mathit{Vars} \mapsto 2^{\mathsf{V}}$ is a map representing the possible values of program variables.

Given a concrete graph $g_1 = \langle \mathsf{V}_1, \mathsf{E}_1, \sigma_1 \rangle$ and an abstract graph $g_2 = \langle \mathsf{V}_2, \mathsf{E}_2, \sigma_2 \rangle$ we say that g_1 can be embedded into g_2, denoted $g_1 \preceq g_2$, if there exists a function $h : \mathsf{V}_1 \mapsto \mathsf{V}_2$ such that

$$\langle x, f, y \rangle \in \mathsf{E}_1 \Rightarrow \langle h(x), f, h(y) \rangle \in \mathsf{E}_2 \tag{5}$$

$$\forall v \in \mathit{Vars}.\ \sigma_2(v) \supseteq \{h(\sigma_1(v))\} \tag{6}$$

The concretization $\gamma_G(g_a)$ of an abstract graph g_a is defined to be the set of all concrete graphs that can be embedded into g_a:

$$\gamma_G(g_a) = \{g_c \in \mathbb{G}_c \mid g_c \preceq g_a\}$$

The Abstract Functional Domain. We now define the domain of graphs used to represent summary functions. A *transformer graph* $\tau \in \mathcal{F}_a$ is a tuple $(\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$, where $\mathsf{EV} \subseteq N_a$ is the set of external vertices, $\mathsf{IV} \subseteq N_a$ is the set of internal vertices, $\mathsf{EE} \subseteq V \times \mathit{Fields} \times V$ is the set of external edges, where $V = \mathsf{EV} \cup \mathsf{IV}$, $\mathsf{IE} \subseteq V \times \mathit{Fields} \times V$ is the set of internal edges, $\pi \in (\mathit{Params} \cup \mathit{Globals}) \mapsto 2^V$ is a map representing the values of parameters and global variables in the *initial* state, and $\sigma \in \mathit{Vars} \mapsto 2^V$ is a map representing the possible values of program variables in the *transformed* state. Furthermore, a transformer graph τ is required to satisfy the following constraints:

$$\begin{aligned} \langle x, f, y\rangle \in \mathsf{EE} &\implies \exists u \in \mathit{range}(\pi).\text{x is reachable from u via } (\mathsf{IE} \cup \mathsf{EE}) \text{ edges} \\ y \in \mathsf{EV} &\implies y \in \mathit{range}(\pi) \vee \exists \langle x, f, y\rangle \in \mathsf{EE} \end{aligned}$$

Given a transformer graph $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$, a node u is said to be a parameter node if $u \in \mathit{range}(\pi)$. A node u is said to be an escaping node if it is reachable from some parameter node via a path of zero or more edges (either internal or external). Let $\mathit{Escaping}(\tau)$ denote the set of escaping nodes in τ. (Note that an abstract graph can be treated as a transformer graph in which all vertices and edges are internal with π being $\lambda x.\{\}$.)

We now define the concretization function $\gamma_T : \mathcal{F}_a \to \mathcal{F}_c$. Given a transformer graph $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$ and a concrete graph $g_c = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c)$, we need to construct a graph representing the transformation of g_c by τ. As explained earlier, every external node $n \in \mathsf{EV}$ in the transformer graph represents a set of vertices in the transformed graph. We now define a function $\eta[\![\tau, g_c]\!] : (\mathsf{IV} \cup \mathsf{EV}) \mapsto 2^{(\mathsf{IV} \cup \mathsf{V}_c)}$ that maps each node in the transformer graph τ to a set of concrete nodes (in g_c) as well as internal nodes (in τ) as the least solution to the following set of constraints over variable μ.

$$v \in \mathsf{IV} \Rightarrow v \in \mu(v) \tag{7}$$

$$v \in \pi(\mathtt{X}) \Rightarrow \sigma_c(\mathtt{X}) \in \mu(v) \tag{8}$$

$$\langle u, f, v\rangle \in \mathsf{EE}, u' \in \mu(u), \langle u', f, v'\rangle \in \mathsf{E}_c \Rightarrow v' \in \mu(v) \tag{9}$$

$$\langle u, f, v\rangle \in \mathsf{EE}, \mu(u) \cap \mu(u') \neq \emptyset, \langle u', f, v'\rangle \in \mathsf{IE} \Rightarrow \mu(v') \subseteq \mu(v) \tag{10}$$

Explanation of the constraints: An internal node represents itself (Eq. 7). An external node labelled by a parameter `X` represents the node pointed to by `X` in the input state g_c (Eq. 8). An external edge $\langle u, f, v\rangle$ indicates that v represents any f-successor v' of any node u' represented by u in the input state (Eq. 9). However, with an external edge $\langle u, f, v\rangle$, we must also account for updates to the f field of the objects represented by u during the procedure execution, ie, the transformation represented by τ, via aliases (as illustrated by the example in Fig. 2(e)). Eq. 10 handles this. The precondition identifies u' as a potential

alias for u (for the given input graph), and identifies updates performed on the f field of (nodes represented by) u'.

Given mapping function η, we define the transformed abstract graph $\tau\langle g_c\rangle$ as $\langle \mathsf{V}', \mathsf{E}', \sigma'\rangle$, where

$$\mathsf{V}' = \mathsf{V}_c \cup \mathsf{IV} \tag{11}$$

$$\mathsf{E}' = \mathsf{E}_c \cup \{\langle v_1, f, v_2\rangle \mid \langle u, f, v\rangle \in \mathsf{IE}, v_1 \in \eta(u), v_2 \in \eta(v)\} \tag{12}$$

$$\sigma' = \lambda x. \bigcup_{u \in \sigma(x)} \eta(u) \tag{13}$$

The transformed graph is an *abstract* graph that represents all concrete graphs that can be embedded in the abstract graph. Thus, we define the concretization function as below:

$$\gamma_T(\tau_a) = \lambda g_c.\gamma_G(\tau_a\langle g_c\rangle).$$

Our abstract interpretation formulation uses only a concretization function. There is no abstraction function α_T. While this form is less common, it is sufficient to establish the soundness of the analysis, as explained in [5]. Specifically, a concrete value $f \in \mathcal{F}_c$ is *correctly represented* by an abstract value $\tau \in \mathcal{F}_a$, denoted $f \sim \tau$, iff $f \sqsubseteq_c \gamma_T(\tau)$. We seek to compute an abstract value that correctly represents the least fixed point of the concrete semantic equations.

Containment Ordering. A natural "precision ordering" exists on $\mathcal{F}_a$, where τ_1 is said to be more precise than τ_2 iff $\gamma_T(\tau_1) \sqsubseteq_c \gamma_T(\tau_2)$. However, this ordering is not of immediate interest to us. (It is not even a partial order, and is hard to work with computationally.) We utilize a stricter ordering in our abstract fixed point computation. We define a relation $\sqsubseteq_{co}$ on $\mathcal{F}_a$ by: $(\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1) \sqsubseteq_{co} (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2)$ iff $\mathsf{EV}_1 \subseteq \mathsf{EV}_2$, $\mathsf{EE}_1 \subseteq \mathsf{EE}_2$, $\forall x.\pi_1(x) \subseteq \pi_2(x)$, $\mathsf{IV}_1 \subseteq \mathsf{IV}_2$, $\mathsf{IE}_1 \subseteq \mathsf{IE}_2$, and $\forall x.\sigma_1(x) \subseteq \sigma_2(x)$.

Lemma 1. *$\sqsubseteq_{co}$ is a partial-order on $\mathcal{F}_a$ with a join operation, denoted $\sqcup_{co}$. Further, γ_T is monotonic with respect to $\sqsubseteq_{co}$: $\tau_1 \sqsubseteq_{co} \tau_2 \Rightarrow \gamma_T(\tau_1) \sqsubseteq_c \gamma_T(\tau_2)$.*

3.3 The Abstract Semantics

Our goal is to approximate the least fixed point computation of the concrete semantics equations 1-4. We do this by utilizing an analogous set of abstract semantics equations shown below. First, we fix the set N_a of abstract nodes. Recall that the domain $\mathcal{F}_a$ defined earlier is parameterized by this set. The WSR algorithm relies on an "allocation site" based merging strategy for bounding the size of the transformer graphs. We utilize the labels attached to statements as allocation-site identifiers. Let *Labels* denote the set of statement labels in the given program. We define N_a to be $\{n_x \mid x \in \mathit{Labels} \cup \mathit{Params} \cup \mathit{Globals}\}$.

We first introduce a variable ϑ_u for every vertex u in the control-flow graph (denoting the abstract value at a program point u), and a variable $\vartheta_{u,v}$ for every

Statement S	**Abstract semantics** $[\![S]\!]_a\tau$ where $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$
$v_1 = v_2$	$(\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma[v_1 \mapsto \sigma(v_2)])$
$\ell : v = new\ C$	$(\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV} \cup \{n_\ell\}, \mathsf{IE} \cup \{n_\ell\} \times \mathit{Fields} \times \{\mathit{null}\}, \sigma[v \mapsto \{n_\ell\}])$
$v_1.f = v_2$	$(\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE} \cup \sigma(v_1) \times \{f\} \times \sigma(v_2), \sigma)$
$\ell : v_1 = v_2.f$	$\mathit{let}\ A = \{n \mid \exists n_1 \in \sigma(v_2), \langle n_1, f, n\rangle \in \mathsf{IE}\}\ \mathit{in}$ $\mathit{let}\ B = \sigma(v_2) \cap \mathit{Escaping}(\tau)\ \mathit{in}$ $\mathit{if}\ (B = \emptyset)$ $\mathit{then}\ (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma[v_1 \mapsto A])$ $\mathit{else}\ (\mathsf{EV} \cup \{n_\ell\}, \mathsf{EE} \cup B \times \{f\} \times \{n_\ell\}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma[v \mapsto A \cup \{n_\ell\}])$

Fig. 3. Abstract semantics of primitive instructions.

edge $u \to v$ in the control-flow graph (denoting the abstract value after the execution of the statement in edge $u \to v$).

$$\vartheta_v = \mathsf{ID} \qquad v \text{ is an entry vertex} \tag{14}$$

$$\vartheta_v = \sqcup_{co}\{\vartheta_{u,v} \mid u \xrightarrow{S} v\} \qquad v \text{ is not an entry vertex} \tag{15}$$

$$\vartheta_{u,v} = [\![S]\!]_a(\vartheta_u) \qquad \text{where } u \xrightarrow{S} v, \text{S is not a call-stmt} \tag{16}$$

$$\vartheta_{u,v} = \vartheta_{exit(Q)}\langle\!\langle\vartheta_u\rangle\!\rangle_a^S \qquad \text{where } u \xrightarrow{S} v, \text{S is a call to Q} \tag{17}$$

Here, ID is a transformer graph consisting of a external vertex for each global variable and each parameter (representing the identity function). Formally, $\mathsf{ID} = (\mathsf{EV}, \emptyset, \pi, \emptyset, \emptyset, \pi)$, where $\mathsf{EV} = \{n_x \mid x \in \mathit{Params} \cup \mathit{Globals}\}$ and $\pi = \lambda v.\ v \in \mathit{Params} \cup \mathit{Globals} \to n_v \mid v \in \mathit{Locals} \to \mathit{null}$. The abstract semantics $[\![S]\!]_a$ of any primitive statement S, other than a procedure call, is shown in Figure 3. The abstract semantics of a procedure call is captured by an operator $\tau_1\langle\!\langle\tau_2\rangle\!\rangle_a^S$, which we will define soon.

The abstract semantics of the first three statements are straightforward. The treatment of the dereference $v_2.f$ in the last statement is more involved. Here, the simpler case is where the dereferenced object is a non-escaping object: in this case, we can directly determine the possible values of $v_2.f$ from the information computed by the local analysis of the procedure. This is handled by the true branch of the conditional statement. The case of escaping objects is handled by the false branch. In this case, in addition to the possible values of $v_2.f$ identified by the local analysis, we must account for two sources of values unknown to the local analysis. The first possibility is that the dereferenced object is a pre-existing object (in the input state) with a pre-existing value for the f field. The second possibility is that the dereferenced object may have aliases unknown to the local analysis via which its f field may have been updated during the procedure's execution. We create an appropriate external node (with a corresponding incoming external edge) that serves as a proxy for these unknown values.

We now consider the abstract semantics of a procedure call statement. Let $\tau_r = (\mathsf{EV}_r, \mathsf{EE}_r, \pi_r, \mathsf{IV}_r, \mathsf{IE}_r, \sigma_r)$ be the transformer graph in the caller before a call statement S to Q and let $\tau_e = (\mathsf{EV}_e, \mathsf{EE}_e, \pi_e, \mathsf{IV}_e, \mathsf{IE}_e, \sigma_e)$ be the abstract summary of Q. We now show how to construct the graph $\tau_e\langle\!\langle\tau_r\rangle\!\rangle_a^S$ representing

the abstract graph at the point after the method call. This operation is an extension of the operation $\tau\langle g_c\rangle$ used earlier to show how τ transforms a concrete state g_c into one of several concrete states.

We first utilize an auxiliary transformer $\tau_e\langle\langle\tau_r, \eta\rangle\rangle$ that takes an extra parameter η that maps nodes of τ_e to a set of nodes in τ_e and τ_r. (As explained above, a node u in τ_e acts as a proxy for a set of vertices in a particular callsite and $\eta(u)$ identifies this set.) Given η, define $\hat{\eta}$ as $\lambda X. \bigcup_{u\in X} \eta(u)$. We then define $\tau_e\langle\langle\tau_r, \eta\rangle\rangle$ to be $(\mathsf{EV}', \mathsf{EE}', \pi', \mathsf{IV}', \mathsf{IE}', \sigma')$ where

$$\mathsf{V}' = (\mathsf{IV}_r \cup \mathsf{EV}_r) \cup \hat{\eta}(\mathsf{IV}_e \cup \mathsf{EV}_e) \tag{18}$$

$$\mathsf{IV}' = \mathsf{V}' \cap (\mathsf{IV}_r \cup \mathsf{IV}_e) \tag{19}$$

$$\mathsf{EV}' = \mathsf{V}' \cap (\mathsf{EV}_r \cup \mathsf{EV}_e) \tag{20}$$

$$\mathsf{IE}' = \mathsf{IE}_r \cup \{\langle v_1, f, v_2\rangle \mid \langle u, f, v\rangle \in \mathsf{IE}_e, v_1 \in \eta(u), v_2 \in \eta(v)\} \tag{21}$$

$$\mathsf{EE}' = \mathsf{EE}_r \cup \{\langle u', f, v\rangle \mid \langle u, f, v\rangle \in \mathsf{EE}_e, u' \in \eta(u), escapes(u')\} \tag{22}$$

$$\pi' = \pi_r \tag{23}$$

$$\sigma' = \lambda x.\ x \in \mathit{Globals} \rightarrow \hat{\eta}(\sigma_e(x)) \mid x \in \mathit{Locals} \cup \mathit{Params} \rightarrow \sigma_r(x) \tag{24}$$

$$escapes(v) \equiv \exists u \in range(\pi').v \text{ is reachable from } u \text{ via } \mathsf{IE}' \cup \mathsf{EE}' \text{ edges} \tag{25}$$

The predicate "escapes(u')" used in the above definition is recursively dependent on the graph τ' being constructed: it checks if u' is reachable from any of the parameter nodes in the graph being constructed. Thus, this leads to an iterative process for adding edges to the graph being constructed, as more escaping nodes are identified.

We now show how the node mapping function $\eta[\![\tau_e, \tau_r,]\!]$ is determined, given the transformers τ_e and τ_r. The function η is defined to be the least fixed point of the set of following constraints over the variable μ. (Here, μ_1 is said to be less than μ_2 iff $\mu_1(u) \subseteq \mu_2(u)$ for all u.) Let a_i denote the actual argument corresponding to the formal argument $\mathit{Param}(i)$.

$$x \in \mathsf{IV}_e \Rightarrow x \in \mu(x) \tag{26}$$

$$x \in \pi_e(\mathit{Param}(i)) \Rightarrow \sigma_r(a_i) \subseteq \mu(x) \tag{27}$$

$$x \in \pi_e(v) \wedge v \in \mathit{Globals} \Rightarrow \sigma_r(v) \subseteq \mu(x) \tag{28}$$

$$\langle u, f, v\rangle \in \mathsf{EE}_e, u' \in \mu(u), \langle u', f, v'\rangle \in \mathsf{IE}_r \Rightarrow v' \in \mu(v) \tag{29}$$

$$\langle u, f, v\rangle \in \mathsf{EE}_e, \mu(u) \cap \mu(u') \neq \emptyset, \langle u', f, v'\rangle \in \mathsf{IE}_e \Rightarrow \mu(v') \subseteq \mu(v) \tag{30}$$

$$\langle u, f, v\rangle \in \mathsf{EE}_e, \mu(u) \cap \mathit{Escaping}(\tau_e\langle\langle\tau_r, \mu\rangle\rangle) \neq \emptyset \Rightarrow v \in \mu(v) \tag{31}$$

In WSR analysis, rule (30) has one more pre-condition, namely $(u \neq u' \vee u \in \mathsf{EV}_e)$. This extra condition may result in a more precise node mapping function but requires a similar change to the definition of the concretization function γ_T.

Abstract Fixed Point Computation. The collection of equations 14-17 can be viewed as a single equation $\vartheta = F^\sharp(\vartheta)$, where $F^\sharp$ is a function from $VE \mapsto \mathcal{F}_a$ to itself. Let $\perp$ denote $\lambda x.(\{\}, \{\}, \lambda v.\{\}, \{\}, \{\}, \lambda v.\{\})$. The analysis iteratively computes the sequence of values ${F^\sharp}^i(\perp)$ and terminates when ${F^\sharp}^i(\perp)$

$= F^{\sharp^{i+1}}(\bot)$. We define $[\![P]\!]_a$ (the summary for a procedure P) to be the value of $\varphi_{exit(P)}$ in the final solution.

Correctness and Termination. With this formulation, correctness and termination of the analysis follow in the standard way. Correctness follows by establishing that $F^\sharp$ is a sound approximation of $F^\natural$, which follows from the following lemma that the corresponding components of $F^\sharp$ are sound approximations of the corresponding components of $F^\natural$. As usual, we say that a concrete value $f \in \mathcal{F}_c$ is *correctly represented* by an abstract value $\tau \in \mathcal{F}_a$, denoted $f \sim \tau$, iff $f \sqsubseteq_c \gamma_T(\tau)$.

Lemma 2. *(a)* $\lambda g.\{g\} \sim \mathsf{ID}$
(b) For every primitive statement S (other than a procedure call), $[\![S]\!]_a$ *is a sound approximation of* $[\![S]\!]_c$*: if* $f \sim \tau$*, then* $f \circ [\![S]\!]_c \sim [\![S]\!]_a(\tau)$.
(c) $\sqcup_{co}$ *is a sound approximation of* $\sqcup_c$*: if* $f_1 \sim \tau_1$ *and* $f_2 \sim \tau_2$*, then* $(f_1 \sqcup_c f_2) \sim (\tau_1 \sqcup_{co} \tau_2)$.
(d) if $f_1 \sim \tau_1$ *and* $f_2 \sim \tau_2$*, then* $f_2 \circ \mathit{CallReturn}_S(f_1) \sim \tau_1 \langle\!\langle \tau_2 \rangle\!\rangle_a^S$.

Lemma 2 implies the following soundness theorem in the standard way (e.g., see Proposition 4.3 of [5]).

Theorem 1. *The computed procedure summaries are correct. (For every procedure P,* $[\![P]\!]_c \sim [\![P]\!]_a$*.)*

Termination follows by establishing that $F^\sharp$ is monotonic with respect to $\sqsubseteq_{co}^*$, since $\mathcal{F}_a$ has only finite height $\sqsubseteq_{co}$-chains.

4 Extensions for Node Merging

In this section we describe an extension to the WSR analysis that allows nodes in the transformer graph to be merged together at arbitrary points during the analysis, as an efficiency heuristic, without sacrificing correctness or termination.

4.1 The Basic Idea

Informally, node merging is an operation that replaces a set of nodes $\{n_1, n_2 \ldots n_m\}$ by a single node n_{rep} such that any predecessor or successor of the nodes $n_1, n_2, \ldots, n_m$ becomes, respectively, a predecessor or successor of n_{rep}. While merging nodes seems like a natural heuristic for improving efficiency, it does introduce some subtle issues and challenges. The intuition for merging nodes arises from their use in the context of heap analyses where graphs represent sets of concrete states. However, in our context, graphs represent state transformers. We now present some results that help establish the correctness of this optimization.

Transformer Graph Embedding. We now extend the notion of graph embedding to transformer graphs. Given $\tau_1 = (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1)$ and

$\tau_2 = (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2)$, we say that $\tau_1 \preceq \tau_2$ (τ_1 can be embedded in τ_2) iff there exists a function $h : (\mathsf{IV}_1 \cup \mathsf{EV}_1) \mapsto (\mathsf{IV}_2 \cup \mathsf{EV}_2)$ such that:

$$x \in \mathsf{IV}_1 \Rightarrow h(x) \in \mathsf{IV}_2 \tag{32}$$

$$x \in \mathsf{EV}_1 \Rightarrow h(x) \in \mathsf{EV}_2 \tag{33}$$

$$\langle x, f, y\rangle \in \mathsf{IE}_1 \Rightarrow \langle h(x), f, h(y)\rangle \in \mathsf{IE}_2 \tag{34}$$

$$\langle x, f, y\rangle \in \mathsf{EE}_1 \Rightarrow \langle h(x), f, h(y)\rangle \in \mathsf{EE}_2 \tag{35}$$

$$\forall v \in \mathit{Vars}.\ \{h(x) \mid x \in \sigma_1(v)\} \subseteq \sigma_2(v) \tag{36}$$

$$\forall v \in \mathit{Vars}.\ \{h(x) \mid x \in \pi_1(v)\} \subseteq \pi_2(v) \tag{37}$$

We use $\tau_1 \preceq_h \tau_2$ to denote that the function h induces an embedding from τ_1 to τ_2. (Note that while this notion of embedding is similar to the one used in TVLA, the one significant difference is that h is not required to be surjective.)

Node merging produces an embedding. Assume that we are given an equivalence relation $\simeq$ on the nodes of a transformer graph τ (such that no internal nodes are equivalent to external nodes). We define the transformer graph $\tau/\simeq$ to be the transformer graph obtained by replacing every node u by a unique representative of its $\simeq$-equivalence class in every component of τ.

Lemma 3. *(a) $\preceq$ is a pre-order. (i.e., it is reflexive and transitive). (b) γ_T is monotonic with respect to $\preceq$: i.e., $\forall \tau_a, \tau_b \in \mathcal{F}_a . \tau_a \preceq \tau_b \Rightarrow \gamma_T(\tau_a) \sqsubseteq_c \gamma_T(\tau_b)$. (c) $\tau \preceq (\tau/\simeq)$.*

Assume that we wish to replace a transformer graph τ by a graph $\tau/\simeq$ at some point during the analysis (perhaps by incorporating this into one of the abstract operations). Our earlier correctness argument still remains valid (since if $f \sim \tau_1 \preceq \tau_2$, then $f \sim \tau_2$).

However, this optimization impacts the termination argument because we do not have $\tau \sqsubseteq_{co} (\tau/\simeq)$. Indeed, our initial implementation of the optimization did not terminate for one program because the computation ended up with a cycle of equivalent, but different, transformers (in the sense of having the same concretization).

Informally, we get around this problem by refining the implementation to ensure that once two nodes are chosen to be merged together, they are always merged together in all subsequent steps. This approach guarantees termination. However, formalizing this is non-trivial. In the sequel, we show how to enhance the underlying domain to include an equivalence relation on nodes (representing the nodes currently merged together) and update the transformers accordingly. We then show how we can ensure termination.

4.2 Enhancing The Abstract Domain

Before we present a formal definition of the enhanced abstract domain, we briefly discuss a few details pertaining to the representation of an equivalence relation in our abstract domain. We represent an equivalence relation on N_a (the set

of abstract nodes) using a function in $N_a \mapsto N_a$ that maps every node in an equivalence class to the (unique) representative of the equivalence class. Say there exists a total ordering ($\leq_v$) on the set of nodes in N_a (for instance, if $n \in N_a$ has an unique identifier $id(n) \in \mathbb{N}$ then the total ordering could be defined as $\forall n_1, n_2 \in N_a, n_1 \leq_v n_2 \iff id(n_1) \leq id(n_2)$). Let $\max_v(N)$ denote the largest element in $N \subseteq N_a$ w.r.t the total order $\leq_v$. Given an equivalence relation $\equiv$ on N_a we define the function $\xi : N_a \mapsto N_a$ representing the equivalence relation as

$$\xi(x) = \max_v(\{y \mid y \equiv x\}) \tag{38}$$

Note that, by the above definition, for every equivalence relation $\equiv$ on N_a there exists a unique function $\xi \in N_a \mapsto N_a$ representing the equivalence relation. Let $\mathcal{P} \subseteq N_a \mapsto N_a$ be the set of functions representing the set of all equivalence relations on N_a. Given a function $\xi \in \mathcal{P}$ the equivalence relation it represents (denote as $\equiv_\xi$) is defined as $\forall x, y \in N_a, x \equiv_\xi y \iff \xi(x) = \xi(y)$. Let $\leq_p$ be a binary relation on $\mathcal{P}$ defined by: $\xi_1 \leq_p \xi_2$ iff the equivalence relation $\equiv_{\xi_1}$ is a refinement of (or subset of) the equivalence relation $\equiv_{\xi_2}$. For all $\xi_1, \xi_2 \in \mathcal{P}$, let $\xi_1 \sqcup_p \xi_2$ be the function representing the equivalence relation on N_a that is a superset of $\equiv_{\xi_1}$ and $\equiv_{\xi_2}$.

Lemma 4. *$\leq_p$ is a partial order on $\mathcal{P}$ with the join operator $\sqcup_p$ (i.e., $(\mathcal{P}, \leq_p, \sqcup_p)$ is a join semi-lattice). The least element of $(\mathcal{P}, \leq_p)$ is $\bot_{\mathcal{P}} = \lambda x.x$.*

Enhanced Abstract Domain. We define the enhanced abstract domain $\mathcal{F}_m$ as the set of pairs $(\tau, \xi) \in \mathcal{F}_a \times \mathcal{P}$ satisfying the following condition:

$$\forall x \in N_a, \quad x \in (\mathsf{EV} \cup \mathsf{IV}) \implies \xi(x) = x \tag{39}$$

The above condition ensures that in every $(\tau, \xi) \in \mathcal{F}_m$, the vertices in τ are the representatives of the equivalence classes of $\equiv_\xi$. Define a binary relation $\leq_m$ on $\mathcal{F}_m$ as follows: $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2)$ iff $\xi_1 \leq_p \xi_2$ and $\tau_1 \preceq_{\xi_2} \tau_2$.

We define the concretization function $\gamma_M : \mathcal{F}_m \mapsto \mathcal{F}_c$ as $\gamma_M((\tau, \xi)) = \gamma_T(\tau)$.

Lemma 5. *(a) $\leq_m$ is a partial order on $\mathcal{F}_m$ with the join operator $\sqcup_m$ defined as follows: $(\tau_1, \xi_1) \sqcup_m (\tau_2, \xi_2) = (apply(\xi_1 \sqcup_p \xi_2, \tau_1 \sqcup_{co} \tau_2), \xi_1 \sqcup_p \xi_2)$. In other words, $(\mathcal{F}_m, \leq_m, \sqcup_m)$ is a join semi-lattice*
(b) The least element of $(\mathcal{F}_m, \leq_m)$ is $\bot_{\mathcal{F}_m} = (\bot_{\mathcal{F}_a}, \bot_{\mathcal{P}})$
(c) γ_M is monotonic w.r.t $\leq_m$.

Let $apply : \mathcal{P} \times \mathcal{F}_a \mapsto \mathcal{F}_a$ be the function that, given a transformer graph $\tau \in \mathcal{F}_a$ and a partition function $\xi \in \mathcal{P}$, applies the partition function ξ on every component of the transformer graph τ. Formally, if $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$

then, $apply(\xi, \tau) = (\mathsf{EV}_m, \mathsf{EE}_m, \pi_m, \mathsf{IV}_m, \mathsf{IE}_m, \sigma_m)$ where,

$$\begin{aligned}
\mathsf{IV}_m &= \{\xi(x) \mid x \in \mathsf{IV}\} \\
\mathsf{EV}_m &= \{\xi(x) \mid x \in \mathsf{EV}\} \\
\mathsf{EE}_m &= \{\langle \xi(x), f, \xi(y)\rangle \mid \langle x, f, y\rangle \in \mathsf{EE}\} \\
\mathsf{IE}_m &= \{\langle \xi(x), f, \xi(y)\rangle \mid \langle x, f, y\rangle \in \mathsf{IE}\} \\
\pi_m(var) &= \{\xi(x) \mid x \in \pi(var)\} \\
\sigma_m(var) &= \{\xi(x) \mid x \in \sigma(var)\}
\end{aligned}$$

Enhanced Abstract Semantics We now discuss the enhanced abstract semantics $[\![S]\!]_m$ for all the statements in our language. For every primitive statement S, $[\![S]\!]_m$ is defined as follows:

$$[\![S]\!]_m((\tau, \xi)) = (apply(\xi, [\![S]\!]_a(\tau)), \xi)$$

The abstract semantics of a method call statement S of the form $Call\ P(v_1, v_2, \ldots, v_k)$ is captured by $\langle\!\langle\rangle\!\rangle_m^S$ defined as follows. Let (τ_r, ξ_r) be the abstract state in the caller at the program point before the call instruction S. Let (τ_e, ξ_e) be the summary of callee.

$$\begin{aligned}
(\tau_e, \xi_e)\langle\!\langle(\tau_r, \xi_r)\rangle\!\rangle_m^S = (\tau', \xi_r \sqcup_p \xi_e) \text{ where,} \\
\tau' = apply(\xi_r \sqcup_p \xi_e, \tau_e\langle\!\langle\tau_r\rangle\!\rangle_a^S)
\end{aligned}$$

Given $(\tau, \xi) \in \mathcal{F}_m$ and $f \in \mathcal{F}_c$, let $f \sim (\tau, \xi)$ denote that f is correctly abstracted by (τ, ξ) i.e, $f \sqsubseteq_c \gamma_M((\tau, \xi))$.

Lemma 6. *(a)* $\lambda g.\{g\} \sim \mathsf{ID}$
(b) For every primitive statement S (other than a procedure call), $[\![S]\!]_m$ is a sound approximation of $[\![S]\!]_c$: if $f \sim (\tau, \xi)$, then $f \circ [\![S]\!]_c \sim [\![S]\!]_m((\tau, \xi))$.
(c) $\sqcup_m$ is a sound approximation of $\sqcup_c$: if $f_1 \sim (\tau_1, \xi_1)$ and $f_2 \sim (\tau_2, \xi_2)$, then $(f_1 \sqcup_c f_2) \sim ((\tau_1, \xi_1) \sqcup_m (\tau_2, \xi_2))$.
(d) if $f_1 \sim (\tau_1, \xi_1)$ and $f_2 \sim (\tau_2, \xi_2)$, then $f_2 \circ CallReturn_S(f_1) \sim (\tau_1, \xi_1)\langle\!\langle(\tau_2, \xi_2)\rangle\!\rangle_m^S$.

Consider the abstract semantics equations given by equations (40-43). We use these equations to compute an over-approximation of the least fixed point of the concrete semantics equations 1-2 by computing the least fixed point of the abstract semantics equations. In the following equations, ϑ_u denotes the abstract value at the vertex (or program point) u of the control flow graph and $\vartheta_{u,v}$ denotes the abstract value after the execution of the statement along the edge $u \to v$ in the control-flow graph.

$$\vartheta_v = \mathsf{ID} \qquad v \text{ is an entry vertex} \tag{40}$$

$$\vartheta_v = \sqcup_m\{\vartheta_{u,v} \mid u \xrightarrow{S} v\} \qquad v \text{ is not an entry vertex} \tag{41}$$

$$\vartheta_{u,v} = [\![S]\!]_m(\vartheta_u) \qquad \text{where } u \xrightarrow{S} v, \text{S is not a call-stmt} \tag{42}$$

$$\vartheta_{u,v} = \vartheta_{exit(Q)}\langle\!\langle\vartheta_u\rangle\!\rangle_m^S \qquad \text{where } u \xrightarrow{S} v, \text{S is a call to Q} \tag{43}$$

As explained before, the collection of equations 40-43 can be viewed as a single equation $\vartheta = F^\sharp(\vartheta)$, where $F^\sharp$ is a function from $VE \mapsto \mathcal{F}_m$ to itself. The abstract analysis iteratively computes the sequence of values ${F^\sharp}^i(\bot)$ and terminates when ${F^\sharp}^i(\bot) = {F^\sharp}^{i+1}(\bot)$. The abstract summary of a procedure P is defined to be the value of $\varphi_{exit(P)}$ in the final solution.

The correctness of the analysis follows from Lemma 6 in the standard way. The termination of the analysis follows from the following lemma which guarantees that every abstract transformer is monotonic w.r.t to the partial order $\leq_m$.

Lemma 7. *(a) For every primitive statement S (other than a procedure call), $[\![S]\!]_m$ is monotonic: if $(\tau, \xi) \leq_m (\tau', \xi')$, then $[\![S]\!]_m((\tau, \xi))) \leq_m [\![S]\!]_m((\tau', \xi'))$. (b) if $(\tau_r, \xi_r) \leq_m (\tau'_r, \xi'_r)$ and $(\tau_e, \xi_e) \leq_m (\tau'_e, \xi'_e)$ then $(\tau_e, \xi_e)\langle\!\langle(\tau_r, \xi_r)\rangle\!\rangle^S_m \leq_m (\tau'_e, \xi'_e)\langle\!\langle(\tau'_r, \xi'_r)\rangle\!\rangle^S_m$.*

Further Approximations In The Fixed Point Computation. As mentioned in the beginning of this section, our goal is to introduce a node merging operation at selected program points in the control flow graph and still guarantee correctness (of the computed summaries) and termination of the analysis. Node merging can be formalized as introducing further approximations in the fixed point computation. Assume that for every control-flow graph vertex v we have an isotonic function $NM_v : \mathcal{F}_m \mapsto \mathcal{F}_m$. (A function $NM_v : \mathcal{F}_m \mapsto \mathcal{F}_m$ is said to be isotonic iff $x \leq_m NM_v(x)$ for all $x \in \mathcal{F}_m$.) In our usage, each NM_v represents a node merging operation, as we will formalize soon. Consider the following set of abstract semantics equations.

$$\vartheta_v = \mathsf{ID} \qquad v \text{ is an entry vertex} \tag{44}$$

$$\vartheta_v = NM_v((\sqcup_m\{\vartheta_{u,v} \mid u \xrightarrow{S} v\})) \qquad v \text{ is not an entry vertex} \tag{45}$$

$$\vartheta_{u,v} = [\![S]\!]_m(\vartheta_u) \qquad \text{where } u \xrightarrow{S} v, \text{S is not a call-stmt} \tag{46}$$

$$\vartheta_{u,v} = \vartheta_{exit(Q)}\langle\!\langle\vartheta_u\rangle\!\rangle^S_m \qquad \text{where } u \xrightarrow{S} v, \text{S is a call to Q} \tag{47}$$

The only difference between the above equations and the equations 40-43 is that, in the above equations, at every program point v, we additionally perform a node merging operation NM_v on the abstract value that results after the join. The above set of equations can be viewed as single equation $\vartheta = (F^\sharp \circ NM^\sharp)(\vartheta)$, where $F^\sharp$ is a function from $VE \mapsto \mathcal{F}_m$ to itself defined by the equations 40-43; $NM^\sharp$ is a function from $VE \mapsto \mathcal{F}_m$ to itself which applies the node merging operation NM_v on the abstract value resulting at the program point v after the application of $F^\sharp$. (The abstract value that results at the edges of the control flow graph after the application of $F^\sharp$ would be left as such).

Lemma 8. $lfp^{\sqsubseteq^*_c} F^\natural \sim lfp^{\leq^*_m}(F^\sharp \circ NM^\sharp)$

Proof. By Lemma 6, $lfp^{\sqsubseteq^*_c} F^\natural \sim lfp^{\leq^*_m} F^\sharp$. Lemma 10 implies that NM_v is isotonic w.r.t $\leq_m$ for all program point v. Hence, by the definition of $NM^\sharp$, $NM^\sharp$ is isotonic w.r.t. $\leq^*_m$. Therefore, $lfp^{\leq^*_m} F^\sharp \leq^*_m lfp^{\leq^*_m}(F^\sharp \circ NM^\sharp)$. Since γ_M is monotonic w.r.t $\leq_m$ (Lemma 5), $lfp^{\sqsubseteq^*_c} F^\natural \sim lfp^{\leq^*_m} F^\sharp$ implies that $lfp^{\sqsubseteq^*_c} F^\natural \sim lfp^{\leq^*_m}(F^\sharp \circ NM^\sharp)$

The correctness of the procedure summaries computed using the equations 44-47 follows from Lemma 8. Since $\mathcal{F}_m$ has only finite length $\leq_m$-chains, the termination of the fixed-point computation of the equations 44-47 follows from the following lemma.

Lemma 9. *Given a monotonic function $F^\sharp$ on a poset $(\mathcal{F}_m^*, \leq_m^*)$ with least element $\perp$ and an isotonic function $NM^\sharp$ on $(\mathcal{F}_m^*, \leq_m^*)$,*

$$\{(F^\sharp \circ NM^\sharp)^i(\perp) \mid i \geq 0\} \text{ is an ascending chain in } (\mathcal{F}_m^*, \leq_m^*)$$

Node Merging. We now show how we define the node merging function NM_u. Consider a function $NM : \mathcal{F}_m \mapsto \mathcal{F}_m$ defined as follows:

$$\begin{aligned} NM((\tau, \xi)) =^{def} & \\ & \text{let } \xi' = \mathit{ChooseNodesToMerge}(\tau) \\ & \text{let } \xi'' = \xi \sqcup_p \xi' \\ & (\mathit{apply}(\xi'', \tau), \xi'') \end{aligned}$$

The above function is parametrized by an function *ChooseNodesToMerge* that is used to identify the set of nodes (all of which are either internal or external) in τ to be merged. Specifically, *ChooseNodesToMerge* returns an element of $\mathcal{P}$, representing an equivalence relation that identifies nodes to be merged tgether. The following results hold irrespective of how this function is defined.

Lemma 10. *NM is isotonic w.r.t $\leq_m$ i.e, $(\tau, \xi) \leq_m NM(\tau, \xi)$.*

The main advantage of the node merging optimization is that it reduces the size of the transformer graph while every other transfer function increases the size of the transformer graphs. However, when used injudiciously, node merging can result in loss of precision. In our implementation we use a couple of heuristics to identify the set of nodes to be merged.

Given $\tau \in \mathcal{F}_a$ and $v_1, v_2 \in \mathsf{V}(\tau)$, we merge v_1, v_2 iff one of the two conditions hold (a) $v_1, v_2 \in \mathsf{EV}(\tau)$ and $\exists u \in \mathsf{V}(\tau)$ s.t. $\langle u, f, v_1\rangle \in \mathsf{EE}(\tau)$ and $\langle u, f, v_2\rangle \in \mathsf{EE}(\tau)$ for some field f or (b) $v_1, v_2 \in \mathsf{IV}(\tau)$ and $\exists u \in \mathsf{V}(\tau)$ s.t. $\langle u, f, v_1\rangle \in \mathsf{IE}(\tau)$ and $\langle u, f, v_2\rangle \in \mathsf{IE}(\tau)$ for some field f.

In the WSR analysis, an external edge $\langle u, f, v\rangle$ on an escaping node u is often used to identify objects that $u.f$ may point-to in the state before the call to the method (i.e, pre-state). However, having two external edges with the same source and same field serves no additional purpose. Our first heuristic eliminates such duplicate external edges, which may be produced, e.g., by multiple reads "`x.f`", where `x` is a formal parameter, of the same field of a pre-state object inside a method or its transitive callees. Our second heuristic addresses a similar problem that might arise due to multiple writes to the same field of an internal object inside a method or its transitive callees. Although, theoretically, the above two heuristics can result in loss of precision, it was not the case on most of the programs on which we ran our analysis (see experimental results section). We

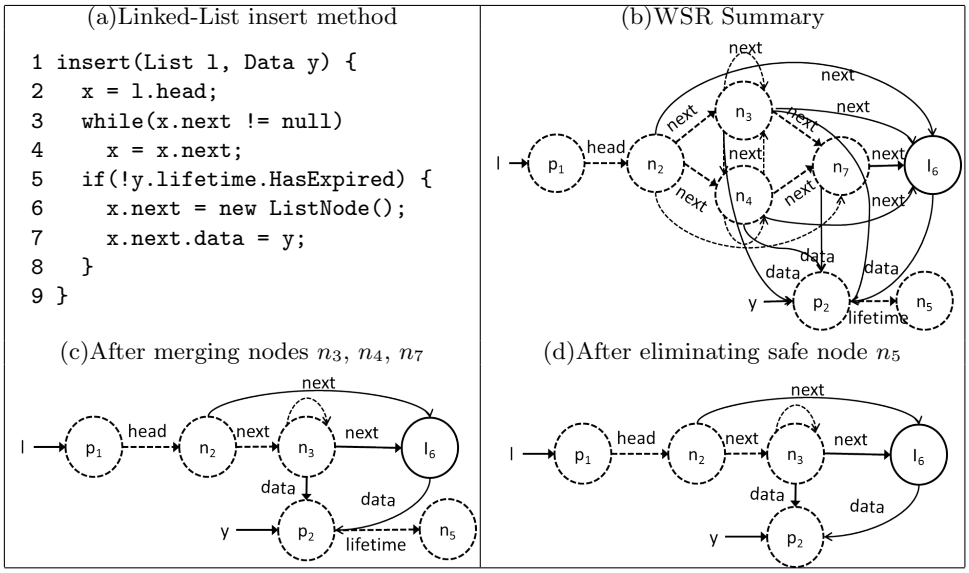


Fig. 4. Illustrative example for the optimizations

apply this node-merging optimization only at procedure exit (to the summary graph produced for the procedure).

Figure 4 shows an illustration of this optimization. Figure 4(a) shows a simple procedure that appends an element to a linked list. Figure 4(b) shows the WSR summary graph that would result by the straight forward application of the transfer functions presented in the paper. Figure 4(c) shows the impact of applying the node-merging optimization on the WSR summary shown in Figure 4(b). In the WSR summary, it can be seen that the external node n_2 has three outgoing external edges on the field *next* that end at nodes n_3, n_4 and n_7. This is due to the reads of the field *next* in the line numbers 3, 4 and 7. As shown in Figure 4(b) the blow-up due to these redundant edges is substantial (even in this small example). Figure 4(c) shows the transformer graph that results after merging the nodes n_3, n_4 and n_7 that are identified as equivalent by our heuristics. Let the transformer graphs shown in Figure 4(b) and Figure 4(c) be τ_a and τ_b respectively. It can be verified that $\gamma(\tau_a) = \gamma(\tau_b)$.

We present an empirical evaluation of the node-merging optimization in the experimental evaluation section.

5 Other Optimizations

In this section we describe a couple of more optimizations (in addition to node merging described before) for the WSR analysis that were motivated by our implementation experience. We do not describe optimizations already discussed by WSR in [19] and [17]. We present an empirical evaluation of the impact of

these optimizations on the scalability and the precision of the purity analysis in the experimental evaluation section.

Optimization 1: Summary Merging. Though the analysis described earlier does not consider virtual method calls, our implementation does handle them (explained in A). Briefly, a virtual method call is modelled as a conditional call to one of the various possible implementation methods. Let the transformer graph before and after the virtual method call statement be τ_{in} and τ_{out} respectively. Let the summaries of the possible targets of the call be $\tau_1, \tau_2, \ldots \tau_n$. In the unoptimized approach, $\tau_{out} = \tau_1\langle\langle\tau_{in}\rangle\rangle \sqcup_{co} \ldots \sqcup_{co} \tau_n\langle\langle\tau_{in}\rangle\rangle$. This optimization constructs a single summary that over-approximates all the callee summaries, as $\tau_{merge} = \tau_1 \sqcup_{co} \ldots \sqcup_{co} \tau_n$ and computes τ_{out} as $\tau_{merge}\langle\langle\tau_{in}\rangle\rangle$. Since each $\tau_i \preceq \tau_{merge}$ (in fact, $\tau_i \sqsubseteq_{co} \tau_{merge}$), τ_{merge} is a safe over-approximation of the summaries of all callees. Once the graph τ_{merge} is constructed it is cached and reused when the virtual method call instruction is re-encountered during the fix-point computation (provided the targets of the virtual method call do not change across iterations and their summaries do not change). We further apply node merging to τ_{merge} to obtain τ_{mo} which is used instead of τ_{merge}.

Optimization 2: Safe Node Elimination. This optimization identifies certain external nodes that can be discarded from a method's summary without affecting correctness. As motivation, consider a method *Set::Contains*. This method does not mutate the caller's state, but its summary includes several external nodes that capture the "reads" of the method. These extraneous nodes make subsequent operations more expensive. Let m be a method with a summary τ. An external vertex ev is safe in τ iff it satisfies the following conditions for every vertex v transitively reachable from ev: (a) v is not modified by the procedure, and (b) No internal edge in τ ends at v and there exists no variable t such that $v \in \sigma(t)$. (We track modifications of nodes with an extra boolean attached to nodes.) Let $removeSafeNodes(\tau)$ denote transformer obtained by deleting all safe nodes in τ. We can show that $\gamma_T(removeSafeNodes(\tau)) = \gamma_T(\tau)$. Like node merging we perform this optimization only at method exits. Figure 4(d) shows the transformer graph that would result after eliminating safe nodes from the transformer graph shown in Figure 4(c).

6 Empirical Evaluation

We implemented the purity analysis along with the optimizations using *Phoenix* analysis framework for .NET binaries [12]. In our implementation, summary computation is performed using an intra-procedural *flow-insensitive* analysis using the transfer functions described in Figure 3. We chose a flow-insensitive analysis due to the prohibitively large memory requirements of a flow-sensitive analysis when run on large libraries. We believe that the optimizations that we propose will have a bigger impact on the scalability of a flow-sensitive analysis.

Fig. 5 shows the benchmarks used in our evaluation. All benchmarks (except *mscorlib.dll* and *System.dll*) are open source C# libraries[4]. We carried out our

Benchmark	LOC	Description
DocX (*dx*)	10K	library for manipulating Word 2007 files
Facebook APIs (*fb*)	21K	library for integrating with Facebook.
Dynamic data display (*ddd*)	25K	real-time data visualization tool
SharpMap (*sm*)	26K	Geospatial application framework
Quickgraph (*qg*)	34K	Graph Data structures and Algorithms
PDfsharp (*pdf*)	96K	library for processing PDF documents
DotSpatial (*ds*)	220K	libraries for manipulating Geospatial data
mscorlib (*ms*)	Unknown	Core C# library
System (*sys*)	Unknown	Core C# library

Fig. 5. benchmark programs

experiments on a 2.83 GHz, 4 core, 64 bit Intel Xeon CPU running Windows Server 2008 with 16GB RAM.

We ran our implementation on all benchmarks in six different configurations (except *QuickGraph* which was run on three configurations only) to evaluate our optimizations: (a) base WSR analysis without any optimizations (*base*) (b) base analysis with summary merging (*base+sm*) (c) base analysis with node merging (*base+nm*) (d) base analysis with summary and node merging (*base+nsm*) (e) base analysis with safe node elimination (*base+sf*) (f) base analysis with all optimizations (*base+all*). We impose a time limit of 3 hours for the analysis of each program (except *QuickGraph* where we used a time limit of 8 hours).

Fig. 6 shows the execution time and memory consumption of our implementation. Runs that exceed the time limit were terminated and their times are listed as ∞. The number of methods classified as pure were same for all configurations (that terminated) for all benchmarks.

The results show that for several benchmarks, node merging drastically reduces analysis time. The other optimizations also reduce the analysis time, though not as dramatically as node merging. Fig. 7 provides insights into the reasons for this improvement by illustrating the correlation between analysis time and number of duplicate edges in the summary. A point (x, y) in the graph indicates that y percentage of analysis time was spent on procedures whose summaries had, on average, at least x outgoing edges per vertex that are labelled by the same field. The benchmarks that benefited from the node merging optimization (viz. SharpMap, PDFSharp, Dynamic Data Display, DotSpatial) spend a large fraction of the analysis time (approx. 90% of the time) on summaries that have average number of duplicate edges per vertex above 4. The graph on the right hand side plots the same metrics when node merging is enabled. It can be seen that node merging is quite effective in reducing the duplicate edges and hence also reduces analysis time.

Benchmarks	*dx*	*fb*	*ddd*	*pdf*	*sm*	*ds*	*ms*	*sys*	*qq*
# of methods	612	4112	2266	3883	1466	10810	2963	698	3380
Pure methods	340	1924	1370	1515	934	5699	1979	411	2152
time(s)									
base	21	52	4696	5088	∞	∞	108	17	∞
base+sf	19	46	3972	2914	∞	∞	56	16	–
base+sm	6	14	3244	4637	7009	∞	54	5	∞
base+nm	20	46	58	125	615	963	21	16	–
base+nsm	5	9	26	79	181	251	13	4	–
base+all	5	8	23	76	179	232	12	4	21718
memory(MB)									
base	313	478	1937	1502	∞	∞	608	387	∞
base+sf	313	460	1836	1136	∞	∞	545	390	–
base+sm	313	478	1937	1508	369	∞	589	390	∞
base+nm	296	460	427	535	356	568	515	387	–
base+nsm	296	461	411	569	369	568	514	390	–
base+all	296	446	410	550	356	568	497	390	703

Fig. 6. Results of analysing the benchmarks in six configurations

7 Related Work

Modular Pointer Analyses. The Whaley-Rinard analysis [19], which is the core of Salcianu-Rinard's purity analysis [17], is one of several modular pointer analyses that have been proposed, such as [2] and [3]. Modular pointer analyses offer the promise of scalability to large applications, but are quite complex to understand and implement. We believe that an abstract interpretation formulation of such modular analyses are valuable as they make them accessible to a larger audience and simplify reasoning about variations and modifications of the algorithm. We are not aware of any previous abstract interpretation formulation of a modular pointer analysis. Our formulation also connects the WSR approach to Sharir-Pnueli's functional approach to interprocedural analysis [18].

Compositional Shape Analyses. Calcagno *et al.* [1] and Gulavani *et al.* [7] present separation-logic based compositional approaches to shape analysis. They perform more precise analysis but compute Hoare triples, which correspond to conditional summaries: summaries which are valid only in states that satisfy the precondition of the Hoare triple. These summaries typically incorporate significant "non-aliasing" conditions in the precondition. Modular pointer analyses such as WSR have somewhat different goals. They are less precise, but more scalable and produce summaries that can be used in any input state.

Parametric Shape Analyses. TVLA [15] is a parametric abstract interpretation that has been used to formalize a number of heap and shape analyses. The WSR analysis and our formalization seem closely related to the relational approach to interprocedural shape analysis presented by Jeannet *et al.* [9]. The Jeannet *et al.*approach shows how the abstract shape graphs of TVLA can be used to represent abstract graph transformers (using a double vocabulary),

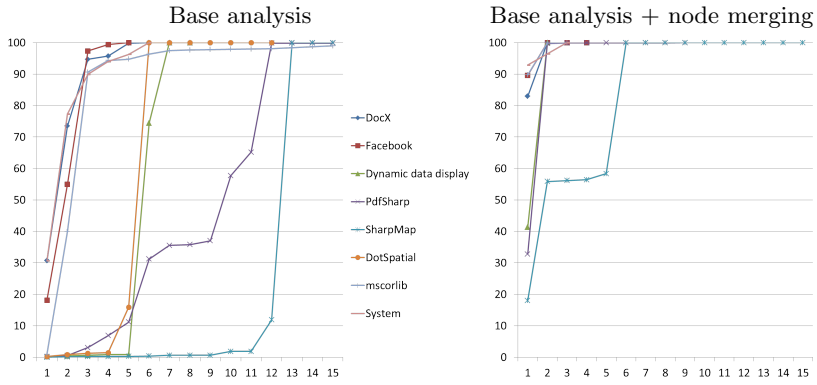


Fig. 7. Number duplicate edges in the summary graph Vs percentage time taken to compute the summary

which is used for modular interprocedural analysis. Rinetzky *et al.* [14] present a tabulation-based approach to interprocedural heap analysis of cutpoint-free programs (which imposes certain restrictions on aliasing). (While the WSR analysis computes a procedure summary that can be reused at any callsite, the tabulation approach may analyze a procedure multiple times, but reuses analysis results at different callsites if the "input heap" is the same.) However, there are interesting similarities and connections between the WSR approach and the Rinetzky *et al.* approach to merging "graphs" from the callee and the caller.

Modularity In Interprocedural Analysis. While the WSR analysis is modular in the absence of recursion, recursive procedures must be analyzed together. Our experience has shown that large strongly connected components of procedures in the call-graph can be a bottleneck in analyzing large libraries. An interesting direction for future work is to explore techniques that can be used to achieve modularity even in the presence of recursion, e.g., see [6].

8 Proofs For The Basic Abstract Interpretation

In this section we discuss the proofs of the lemmas presented in the previous sections. The proofs make use of several interesting results (stated and proved in this section) that provides more insights into the WSR analysis.

8.1 Notations and Definitions.

Given a function $h : S_1 \mapsto \mathcal{P}(S_2)$, we define $\hat{h} : \mathcal{P}(S_1) \mapsto \mathcal{P}(S_2)$ as $\hat{h}(X) = \bigcup_{x \in X} h(x)$. A function $g : S_1 \mapsto S_2$ can be naturally lifted to a function $g' : S_1 \mapsto \mathcal{P}(S_2)$, we use $\hat{g}$ to denote $\hat{g'}$.

Definition 1. ***(Embedding)*** *Let $\zeta \subseteq Vars$. Let $\tau_1 = (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1)$ and $\tau_2 = (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2)$, be two transformer graphs. We say that $\tau_1 \preceq_h^\zeta \tau_2$, $h : (\mathsf{EV}_1 \cup \mathsf{IV}_1) \mapsto (\mathsf{EV}_2 \cup \mathsf{IV}_2)$, iff*

$$x \in \mathsf{IV}_1 \Rightarrow h(x) \in \mathsf{IV}_2 \tag{48}$$

$$x \in \mathsf{EV}_1 \Rightarrow h(x) \in \mathsf{EV}_2 \tag{49}$$

$$\langle x, f, y\rangle \in \mathsf{IE}_1 \Rightarrow \langle h(x), f, h(y)\rangle \in \mathsf{IE}_2 \tag{50}$$

$$\langle x, f, y\rangle \in \mathsf{EE}_1 \Rightarrow \langle h(x), f, h(y)\rangle \in \mathsf{EE}_2 \tag{51}$$

$$\forall \boldsymbol{v} \in Vars.\ \hat{h}(\pi_1(\boldsymbol{v})) \subseteq \pi_2(\boldsymbol{v}) \tag{52}$$

$$\forall \boldsymbol{v} \in Vars \setminus \zeta.\ \hat{h}(\sigma_1(\boldsymbol{v})) \subseteq \sigma_2(\boldsymbol{v}) \tag{53}$$

Additionally, the function h is required to map *null* to *null*. The above definition is straightforward, with one unusual aspect worth noting: ζ identifies a set of variables whose final values need not be preserved by the embedding.

An abstract graph in $\mathbb{G}_a$ can be treated as a transformer graph in which all vertices and edges are internal with π being $\lambda x.\{\}$. Hence, the above notion of embedding also applies to the abstract graphs in $\mathbb{G}_a$.

In general, we omit the subscript h of $\preceq$ if the embedding function is not relevant for the context and omit the superscript ζ if it is $\emptyset$. For all $\tau_1, \tau_2, \tau_3 \in \mathcal{F}_a$ and for all $\zeta \subseteq Vars$, we have the following properties of the embedding ($\preceq$):

Property 1. $\tau_1 \preceq_h \tau_2 \implies \tau_1 \preceq_h^\zeta \tau_2$

Property 2. $\tau_1 \sqsubseteq_{co} \tau_2$ iff $\tau_1 \preceq_{id} \tau_2$, where id is the identity function.

Property 3. (Transitivity) $(\tau_1 \preceq_g \tau_2) \wedge (\tau_2 \preceq_h \tau_3) \implies (\tau_1 \preceq_{g \circ h} \tau_3)$

8.2 Monotonicity of γ_T (Lemma 1 and Lemma 3)

Lemma 11. $\forall g_1, g_2 \in \mathbb{G}_a.\ g_1 \preceq g_2 \Rightarrow \gamma_G(g_1) \subseteq \gamma_G(g_2)$

Proof. Directly follows from the definition of γ_G and $\preceq$.

Let $g_c \in \mathbb{G}_c$ be a concrete graph and $\tau \in \mathcal{F}_a$ be an abstract transformer graph. As described in Section 3, $\eta[\![\tau, g_c]\!]$ is the function that maps all the internal and external vertices in a transformer graph τ to the concrete vertices in g_c and internal vertices in τ. As explained in Section 3, η is the least solution of the set of (recursive) constraints 7-10 over the variable μ.

The least-solution can be computed iteratively as follows (*Kleene's iteration*). Say the value of μ computed at the end of iteration $i, i \geq 0$ is μ^i. Initially, $\forall x \in (\mathsf{IV} \cup \mathsf{EV}), \mu^0(x) = \emptyset$. At every iteration $i, i \geq 1$, μ^i is computed by applying the rules on the value of μ computed in the previous iteration i.e, μ^{i-1}. The least solution for μ (which is η) is equal to μ^j where j is the smallest integer such that $\mu^{j-1} = \mu^j$.

We represent $\eta[\![\tau, g_c]\!]$ as η wherever τ and g_c are clear from the context. Let $con(\eta(x))$ denote the concrete vertices (runtime objects) in $\eta(x)$ and $int(\eta(x))$ denote the internal vertices in $\eta(x)$.

Lemma 12. *If* $\tau_1 = (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1), \tau_2 = (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2) \in \mathcal{F}_a$ *and* $\tau_1 \preceq_h^{\zeta} \tau_2$ *then for all* $g_c \in \mathbb{G}_c$ *and for all* $x \in \mathsf{IV}_1 \cup \mathsf{EV}_1$*, we have:*

$$con(\eta[\![\tau_1, g_c]\!](x)) \subseteq con(\eta[\![\tau_2, g_c]\!](h(x))$$
$$\hat{h}(int(\eta[\![\tau_1, g_c]\!](x))) \subseteq int(\eta[\![\tau_2, g_c]\!](h(x))$$

Proof. We prove this by induction over the iterations in the computations of the least solution of the rules (7-10).

Claim. Given a $g_c = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c) \in \mathbb{G}_c$, $\forall x \in \mathsf{V}_1.$,

$$con(\mu^i[\![\tau_1, g_c]\!](x)) \subseteq con(\eta[\![\tau_2, g_c]\!](h(x))$$
$$\hat{h}(int(\mu^i[\![\tau_1, g_c]\!](x))) \subseteq int(\eta[\![\tau_2, g_c]\!](h(x))$$

Since $\eta[\![\tau_2, g_c]\!]$ is the least solution for the set of constraints 7-10, it must satisfy the constraints. Hence,

$$v \in \mathsf{IV}_2 \Rightarrow v \in \eta(v) \tag{54}$$
$$v \in \pi_2(\mathtt{X}) \Rightarrow \sigma_c(\mathtt{X}) \in \eta(v) \tag{55}$$
$$\langle u, f, v\rangle \in \mathsf{EE}_2, u' \in \eta(u), \langle u', f, v'\rangle \in \mathsf{E}_c \Rightarrow v' \in \eta(v) \tag{56}$$
$$\langle u, f, v\rangle \in \mathsf{EE}_2, \eta(u) \cap \eta(u') \neq \emptyset, \langle u', f, v'\rangle \in \mathsf{IE}_2 \Rightarrow \eta(v') \subseteq \eta(v) \tag{57}$$

Base case (iteration 1). Say $x \in (\mathsf{IV}_1 \cup \mathsf{EV}_1)$ and $y \in \mu^1(x)$. The *external edges rule* and *aliasing rule* do not apply in this case as $\mu^0 = \emptyset$. Hence, y should have been added to $\mu^1(x)$ by the *param nodes* rule (8) or *internal nodes* rule (7).

a) Say y was added by the *Param nodes* rule (8). Hence the following must hold,

$$\exists \mathtt{V} \in \textit{Vars} \text{ s.t. } x \in \pi_1(\mathtt{V}) \wedge y \in \mathsf{V}_c \wedge y = \sigma_c(\mathtt{V})$$
$$\text{Since } \tau_1 \preceq_h \tau_2,\ x \in \pi_1(\mathtt{V}) \text{ implies that, } h(x) \in \pi_2(\mathtt{V})$$

From constraint 55, $y = \sigma_c(\mathtt{V})$ and $h(x) \in \pi_2(\mathtt{V})$ implies that,

$$y \in \eta[\![\tau_2, g_c]\!](h(x)) \tag{58}$$

b) Say y was added by the *Internal nodes* (Constraint 7). Hence the following must hold,

$$x \in \mathsf{IV}_1 \wedge y = x \tag{59}$$
$$\text{Since } \tau_1 \preceq_h \tau_2,\ x \in \mathsf{IV}_1 \text{ implies that, } h(x) \in \mathsf{IV}_2 \tag{60}$$
$$\text{By constraint 54, } \ h(x) \in \eta[\![\tau_2, g_c]\!](h(x)) \tag{61}$$
$$\text{Since } y = x,\ h(y) \in \eta[\![\tau_2, g_c]\!](h(x)) \tag{62}$$

Thus the claim holds in the base case.

Induction step. Now let us assume that the claim holds for all iterations from 1 to $i-1$. Now say y is added in iteration i by any of the four rules. If y is added by the *internal nodes* or *param nodes* rule, by the proof shown for the *base case*,

the claim holds. In the rest of the proof we denote $\mu[\![\tau_1, g_c]\!]$ as μ and $\eta[\![\tau_2, g_c]\!]$ as η.

(a)Say y is added by the *external edge* rule (Constraint 9). In this case the following must hold,

$$\exists u, u' \text{ s.t. } \langle u, f, x\rangle \in \mathsf{EE}_1 \wedge u' \in \mu^{i-1}(u) \wedge \langle u', f, y\rangle \in \mathsf{E}_c \tag{63}$$

In the above formulae, u' is a concrete node as $\langle u', f, v'\rangle \in \mathsf{E}_c$. By the inductive hypothesis $u' \in \mu^{i-1}(u)$ implies that,

$$u' \in \eta(h(u)) \tag{64}$$

Since $\tau_1 \preceq_h \tau_2$,

$$\langle u, f, x\rangle \in \mathsf{EE}_1 \implies \langle h(u), f, h(x)\rangle \in \mathsf{EE}_2 \tag{65}$$

Hence, from constraints 64 and 65,

$$\langle h(u), f, h(x)\rangle \in \mathsf{EE}_2 \wedge u' \in \eta(h(u)) \wedge \langle u', f, y\rangle \in \mathsf{E}_c \tag{66}$$

$$\text{From constraint 56, } y \in \eta(h(x)) \tag{67}$$

(b)Say y is added by the *aliasing* rule (Constraint 10). Hence the following must hold,

$$\exists u, u', v' \text{ s.t.}$$

$$\langle u, f, x\rangle \in \mathsf{EE}_1 \wedge \mu^{i-1}(u) \cap \mu^{i-1}(u') \neq \emptyset \wedge \langle u', f, v'\rangle \in \mathsf{IE}_1 \wedge y \in \mu^{i-1}(v') \tag{68}$$

Since $\mu^{i-1}(u) \cap \mu^{i-1}(u') \neq \emptyset$, by the inductive hypothesis,

$$\eta(h(u)) \cap \eta(h(u')) \neq \emptyset \tag{69}$$

Since $\tau_1 \preceq \tau_2$,

$$\langle u, f, x\rangle \in \mathsf{EE}_1 \implies \langle h(u), f, h(x)\rangle \in \mathsf{EE}_2 \tag{70}$$

$$\langle u', f, v'\rangle \in \mathsf{IE}_1 \implies \langle h(u'), f, h(v')\rangle \in \mathsf{IE}_2 \tag{71}$$

From (69), (70) and (71),

$$\eta(h(u)) \cap \eta(h(u')) \neq \emptyset \wedge \langle h(u), f, h(x)\rangle \in \mathsf{EE}_2 \wedge \langle h(u'), f, h(v')\rangle \in \mathsf{IE}_2 \tag{72}$$

$$\text{From Constraint 57, } \eta(h(v')) \subseteq \eta(h(x)) \tag{73}$$

By the inductive hypothesis,

$$con(\mu^{i-1}(v')) \subseteq con(\eta(h(v'))) \tag{74}$$

$$\hat{h}(int(\mu^{i-1}(v'))) \subseteq int(\eta(h(v'))) \tag{75}$$

Hence, from (73), (74) and (75),

$$con(\mu^{i-1}(v')) \subseteq con(\eta(h(x))) \tag{76}$$

$$\hat{h}(int(\mu^{i-1}(v'))) \subseteq int(\eta(h(x))) \tag{77}$$

By (68), $y \in \mu^{i-1}(v')$. Therefore by (76) and (77),

$$y \in con(\mu^{i-1}(v')) \implies y \in con(\eta(h(x))) \tag{78}$$

$$y \in int(\mu^{i-1}(v')) \implies h(y) \in int(\eta(h(x))) \tag{79}$$

Corollary 1. *Let* $\tau_1 = (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1)$, $\tau_2 = (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2)$. *If* $\tau_1 \preceq_h \tau_2$ *then for all* $g_c \in \mathbb{G}_c$ *and for all* $x \in (\mathsf{EV}_1 \cup \mathsf{IV}_1)$, *we have:*

$$con(\eta[\![\tau_1, g_c]\!](x)) \subseteq con(\eta[\![\tau_2, g_c]\!](h(x))$$

$$\hat{h}(int(\eta[\![\tau_1, g_c]\!](x))) \subseteq int(\eta[\![\tau_2, g_c]\!](h(x))$$

Lemma 13. $\forall g_c \in \mathbb{G}_c.\ (\tau_1 \preceq^{\zeta} \tau_2) \implies (\tau_1\langle g_c\rangle \preceq^{\zeta} \tau_2\langle g_c\rangle)$

Proof. Let $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$. As mentioned in the Section 3, $\tau\langle g_c\rangle$ is defined as follows: $\tau\langle g_c\rangle = (\mathsf{V}_{\tau\langle g_c\rangle}, \mathsf{E}_{\tau\langle g_c\rangle}, \sigma_{\tau\langle g_c\rangle})$ where,

$$\mathsf{V}_{\tau\langle g_c\rangle} = \mathsf{V}_c \cup \mathsf{IV} \tag{80}$$

$$\mathsf{E}_{\tau\langle g_c\rangle} = \mathsf{E}_c \cup \{\langle v_1, f, v_2\rangle \mid \langle u, f, v\rangle \in \mathsf{IE}, v_1 \in \eta(u), v_2 \in \eta(v)\} \tag{81}$$

$$\sigma_{\tau\langle g_c\rangle} = \lambda x.\ \bigcup_{u \in \sigma(x)} \eta(u) \tag{82}$$

To prove that $\tau_1\langle g_c\rangle \preceq \tau_2\langle g_c\rangle$ we need to construct a function (say $h' : \mathsf{V}_{\tau_1\langle g_c\rangle} \mapsto \mathsf{V}_{\tau_2\langle g_c\rangle}$) that satisfies the embedding conditions given by Constraints (5).

Let h be any function that induces an embedding from τ_1 and τ_2. Define h' using the function h as follows:

$$\forall x \in \mathsf{V}_{\tau_1\langle g_c\rangle}.\quad h'(x) = \begin{cases} h(x), & x \in \mathsf{IE}_1 \\ x, & otherwise\ (i.e, x \in \mathsf{V}_c) \end{cases}$$

By the Lemma 12, if $\tau_1 \preceq_h^{\zeta} \tau_2$ then

$$w \in \eta[\![\tau_1, g_c]\!](x), w \in \mathsf{IV}_1 \implies h(w) \in \eta[\![\tau_2, g_c]\!](h(x)) \tag{83}$$

$$w \in \eta[\![\tau_1, g_c]\!](x), w \in \mathsf{V}_c \implies w \in \eta[\![\tau_2, g_c]\!](h(x)) \tag{84}$$

From the definition of h',

$$w \in \eta[\![\tau_1, g_c]\!](x) \implies h'(w) \in \eta[\![\tau_2, g_c]\!](h(x)) \tag{85}$$

(a) Say $\langle v_1, f, v_2\rangle \in \mathsf{E}_{\tau_1\langle g_c\rangle}$.

$$\text{By (81)},\quad \langle v_1, f, v_2\rangle \in \mathsf{E}_{\tau_1\langle g_c\rangle} \implies$$
$$\exists u, v \text{ s.t. } \langle u, f, v\rangle \in \mathsf{IE}_1, v_1 \in \eta[\![\tau_1, g_c]\!](u), v_2 \in \eta[\![\tau_1, g_c]\!](v) \tag{86}$$

$$\text{Since } \tau_1 \preceq_h^{\zeta} \tau_2,\quad \langle u, f, v\rangle \in \mathsf{IE}_1 \implies \langle h(u), f, h(v)\rangle \in \mathsf{IE}_2 \tag{87}$$

$$\text{By (85)},\quad v_1 \in \eta[\![\tau_1, g_c]\!](u) \implies h'(v_1) \in \eta[\![\tau_2, g_c]\!](h(u)) \tag{88}$$

$$\text{and},\quad v_2 \in \eta[\![\tau_1, g_c]\!](v) \implies h'(v_2) \in \eta[\![\tau_2, g_c]\!](h(v)) \tag{89}$$

$$\text{By (81)},\quad \langle h'(v_1), f, h'(v_2)\rangle \in \mathsf{E}_{\tau_2\langle g_c\rangle} \tag{90}$$

(b) Say $v \in \sigma_{\tau_1\langle g_c\rangle}(\mathtt{V})$.

$$\text{By (82)}, \quad v \in \sigma_{\tau_1\langle g_c\rangle}(\mathtt{V}) \implies \exists u \in \sigma_1(\mathtt{V}) \text{ s.t. } v \in \eta[\![\tau_1, g_c]\!](u)$$

$$\text{Since } \tau_1 \preceq_h^\zeta \tau_2, \quad h(u) \in \sigma_2(\mathtt{V}) \tag{91}$$

$$\text{By (85)}, \quad v \in \eta[\![\tau_1, g_c]\!](u) \implies h'(v) \in \eta[\![\tau_2, g_c]\!](h(u)) \tag{92}$$

$$\text{By (82)}, \quad h'(v) \in \sigma_{\tau_2\langle g_c\rangle}(\mathtt{V}) \tag{93}$$

Corollary 2. *If $\tau_1 \preceq_{id}^\zeta \tau_2$ then $\tau_1\langle g_c\rangle \preceq_{id}^\zeta \tau_2\langle g_c\rangle$ for all $g_c \in \mathbb{G}_c$.*

Proof. By the above proof, if $\tau_1 \preceq_h^\zeta \tau_2$ then $\tau_1\langle g_c\rangle \preceq_{h'}^\zeta \tau_2\langle g_c\rangle$. By the definition of h', when $h = id$, $h' = id$.

Lemma 14. $\tau_1 \preceq \tau_2 \implies \gamma_T(\tau_1) \sqsubseteq_c \gamma_T(\tau_2)$

Proof. Directly follows from the definition of γ_T, Lemma 11 and Lemma 13.

Corollary 3. *$\tau_1 \preceq \tau_2$ and $\tau_2 \preceq \tau_1$ implies that $\gamma(\tau_1) = \gamma(\tau_2)$. The concrete images of τ_1 and τ_2 are equal if they belong to the same equivalence class in $\mathcal{F}_a$ wrt $\preceq$.*

8.3 Abstract Semantics Of Primitive Statements (Lemma 2)

We now show that, for any statement S that is not a procedure call statement, if f is correctly abstracted by τ then $f \circ [\![S]\!]_c$ is correctly abstracted by $[\![S]\!]_a(\tau)$ i.e,

$$f \sim \tau \implies f \circ [\![S]\!]_c \sim [\![S]\!]_a(\tau) \tag{94}$$

where, $f \sim \tau \iff f \sqsubseteq_c \gamma_T(\tau)$. It is easy to see that if $f_1 \sqsubseteq_c f_2$ then $f_1 \circ [\![S]\!]_c \sqsubseteq_c f_2 \circ [\![S]\!]_c$. Hence, it suffices to prove that

$$\gamma_T(\tau) \circ [\![S]\!]_c \sqsubseteq_c \gamma_T([\![S]\!]_a(\tau)) \tag{95}$$

$$\equiv \forall g_c \in \mathbb{G}_c,\ (\gamma_T(\tau) \circ [\![S]\!]_c)(g_c) \subseteq \gamma_T([\![S]\!]_a(\tau))(g_c) \tag{96}$$

$$\equiv \forall g_c \in \mathbb{G}_c, \{[\![S]\!]_c(ptg_1) \mid ptg_1 \in \gamma_T(\tau)(g_c)\} \subseteq \gamma_T([\![S]\!]_a(\tau))(g_c) \tag{97}$$

Using the definition of γ_T the above can be reduced to

$$\forall g_c \in \mathbb{G}_c, \{[\![S]\!]_c(ptg_1) \mid ptg_1 \in \gamma_G(\tau\langle g_c\rangle)\} \subseteq \gamma_G([\![S]\!]_a(\tau)\langle g_c\rangle) \tag{98}$$

By the definition of γ_G,

$$\forall g_c \in \mathbb{G}_c,\ \forall ptg_1 \in \mathbb{G}_c,\ ptg_1 \preceq \tau\langle g_c\rangle \implies [\![S]\!]_c(ptg_1) \preceq [\![S]\!]_a(\tau)\langle g_c\rangle \tag{99}$$

Hence it suffices to show that for each of the transfer functions shown in Figure 3, (99) holds.

We now prove a lemma that applies to all the transfer functions shown in Figure 3.

Lemma 15. *Let S be a statement, $\tau_a = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a)$ be a transformer graph and $ptg = (\mathsf{V}_{ptg}, \mathsf{E}_{ptg}, \sigma_{ptg})$ be a concrete graph. Let $\tau'_a = [\![S]\!]_a(\tau_a)$. If $[\![S]\!]_a(\tau_a)$ is of the form*

$$(\mathsf{EV}_a \cup \mathsf{EV}_{new}, \mathsf{EE}_a \cup \mathsf{EE}_{new}, \pi_a, \mathsf{IV}_a \cup \mathsf{IV}_{new}, \mathsf{IE}_a \cup \mathsf{IE}_{new}, \sigma_a[v_1 \mapsto \chi])$$

where $\chi \subseteq (\mathsf{EV}_a \cup \mathsf{EV}_{new} \cup \mathsf{IV}_a \cup \mathsf{IV}_{new})$ then $ptg \preceq_h \tau_a\langle g_c\rangle$ implies that

$$\langle x, f, y\rangle \in \mathsf{E}_{ptg} \implies \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle}$$
$$\forall v \in \mathit{Vars} \setminus \{v_1\}. \quad h(\sigma_{ptg}(v)) \subseteq \sigma_{\tau'_a\langle g_c\rangle}(v)$$

Proof. Say $\langle x, f, y\rangle \in \mathsf{E}_c$,

$$\text{Since } ptg \preceq_h \tau_a\langle g_c\rangle, \quad \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau_a\langle g_c\rangle} \tag{100}$$
$$\text{By our assumption about } [\![S]\!]_a(\tau_a), \quad \tau_a \preceq_{id}^{\{v_1\}} \tau'_a \tag{101}$$
$$\text{By Corollary 2,} \quad \tau_a\langle g_c\rangle \preceq_{id}^{\{v_1\}} \tau'_a\langle g_c\rangle \tag{102}$$
$$\text{From (100) and (102),} \quad \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{103}$$

Say $x = \sigma_c(var) \wedge var \neq v_1$,

$$\text{Since } ptg \preceq_h \tau_a\langle g_c\rangle, \quad h(x) \in \sigma_{\tau_a\langle g_c\rangle}(var) \tag{104}$$
$$\text{From (102)} \quad \tau_a\langle g_c\rangle \preceq_{id}^{\{v_1\}} \tau'_a\langle g_c\rangle \tag{105}$$
$$\text{Since } var \neq v_1, \quad h(x) \in \sigma_{\tau'_a\langle g_c\rangle}(var) \tag{106}$$

Without loss of generality assume that in all the statements in Figure 3, $v_1 \neq v_2$.

Lemma 16. *For the statement $S : v_1 = v_2$, the abstract semantics of S correctly approximates the concrete semantics of S.*

Proof. Let $\tau_a = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a)$ and $ptg_1 = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c)$. $[\![S]\!]_c(ptg_1)$ and $[\![S]\!]_a(\tau_a)$ are defined as follows:

$$[\![S]\!]_c(\mathsf{V}_c, \mathsf{E}_c, \sigma_c) = \{(\mathsf{V}_c, \mathsf{E}_c, \sigma_c[v_1 \mapsto \sigma_c(v_2)])\}$$
$$[\![S]\!]_a(\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a) = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a[v_1 \mapsto \sigma_a(v_2)])$$

Let,

$$ptg'_1 = [\![S]\!]_c(ptg_1) = (\mathsf{V}'_c, \mathsf{E}'_c, \sigma'_c)$$
$$\tau'_a = [\![S]\!]_a(\tau_a) = (\mathsf{EV}'_a, \mathsf{EE}'_a, \pi'_a, \mathsf{IV}'_a, \mathsf{IE}'_a, \sigma'_a)$$
$$\tau_a\langle g_c\rangle = (\mathsf{V}_{\tau_a\langle g_c\rangle}, \mathsf{E}_{\tau_a\langle g_c\rangle}, \sigma_{\tau_a\langle g_c\rangle})$$
$$\tau'_a\langle g_c\rangle = (\mathsf{V}_{\tau'_a\langle g_c\rangle}, \mathsf{E}_{\tau'_a\langle g_c\rangle}, \sigma_{\tau'_a\langle g_c\rangle})$$

From (99), we need to prove that,

$$\forall g_c \in \mathbb{G}_c, \ \forall ptg_1 \in \mathbb{G}_c, \ ptg_1 \preceq \tau_a\langle g_c\rangle \implies ptg'_1 \preceq \tau'_a\langle g_c\rangle$$

Since, $ptg_1 \preceq \tau_a\langle g_c\rangle$, there exists a function h s.t., $ptg_1 \preceq_h \tau_a\langle g_c\rangle$. We prove that h induces an embedding from ptg'_1 to $\tau_a\langle g_c\rangle$ by proving the following two conditions (see Conditions 5):

$$\langle x, f, y\rangle \in \mathsf{E}'_c \Rightarrow \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{107}$$

$$\forall v \in \mathit{Vars}.\ \sigma_{\tau'_a\langle g_c\rangle}(v) \supseteq \{h(\sigma'_c(v))\} \tag{108}$$

(Proof of 107). Say $\langle x, f, y\rangle \in \mathsf{E}'_c$,

$$\text{From the concrete semantics,}\quad \langle x, f, y\rangle \in \mathsf{E}_c \tag{109}$$

$$\text{By Lemma 15,}\quad \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{110}$$

(Proof of 108). Say $x = \sigma'_c(var)$,

1. *Case (a)*, $var \neq v_1$

$$\text{From the concrete semantics,}\quad x = \sigma_c(var) \tag{111}$$

$$\text{By Lemma 15,}\quad h(x) \in \sigma_{\tau'_a\langle g_c\rangle}(var) \tag{112}$$

2. *Case (b)*, $var = v_1$

$$\text{From concrete semantics,}\quad x \in \sigma'_c(v_1) \implies x \in \sigma'_c(v_2) \tag{113}$$

$$\text{Since } v_2 \neq v_1\text{, by case(a),}\quad x \in \sigma'_c(v_2) \implies h(x) \in \sigma_{\tau'_a\langle g_c\rangle}(v_2) \tag{114}$$

$$\text{From abstract semantics,}\quad \sigma'_a(v_2) = \sigma'_a(v_1) \tag{115}$$

$$\text{By the definition of } \langle\rangle,\quad \sigma_{\tau'_a\langle g_c\rangle}(v_2) = \sigma_{\tau'_a\langle g_c\rangle}(v_1) \tag{116}$$

$$\text{By (114) and (116),}\quad h(x) \in \sigma_{\tau'_a\langle g_c\rangle}(v_1) \tag{117}$$

Lemma 17. *For the statement $S : \ell : v = \mathit{new}\ C$, the abstract semantics of S correctly approximates the concrete semantics of S.*

Proof. Let $\tau_a = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a)$ and $ptg_1 = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c)$. $[\![S]\!]_c(ptg_1)$ and $[\![S]\!]_a(\tau_a)$ are defined as follows:

$$[\![S]\!]_c(\mathsf{V}_c, \mathsf{E}_c, \sigma_c) = \{(\mathsf{V}_c \cup \{n\}, \mathsf{E}_c \cup \{n\} \times \mathit{Fields} \times \{\mathit{null}\}, \sigma_c[v \mapsto n]) \mid n \in N_c \setminus \mathsf{V}\}$$

$$[\![S]\!]_a(\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a) =$$
$$(\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a \cup \{n_\ell\}, \mathsf{IE}_a \cup \{n_\ell\} \times \mathit{Fields} \times \{\mathit{null}\}, \sigma_a[v \mapsto \{n_\ell\}])$$

Let,

$$ptg'_1 = [\![S]\!]_c(ptg_1) = (\mathsf{V}'_c, \mathsf{E}'_c, \sigma'_c)$$
$$\tau'_a = [\![S]\!]_a(\tau_a) = (\mathsf{EV}'_a, \mathsf{EE}'_a, \pi'_a, \mathsf{IV}'_a, \mathsf{IE}'_a, \sigma'_a)$$
$$\tau_a\langle g_c\rangle = \langle \mathsf{V}_{\tau_a\langle g_c\rangle}, \mathsf{E}_{\tau_a\langle g_c\rangle}, \sigma_{\tau_a\langle g_c\rangle}\rangle$$
$$\tau'_a\langle g_c\rangle = \langle \mathsf{V}_{\tau'_a\langle g_c\rangle}, \mathsf{E}_{\tau'_a\langle g_c\rangle}, \sigma_{\tau'_a\langle g_c\rangle}\rangle$$

From (99), we need to prove that,

$$\forall g_c \in \mathbb{G}_c,\ \forall ptg_1 \in \mathbb{G}_c,\ ptg_1 \preceq \tau_a\langle g_c\rangle \implies ptg'_1 \preceq \tau'_a\langle g_c\rangle$$

Since, $ptg_1 \preceq \tau_a\langle g_c\rangle$, there exists a function h s.t., $ptg_1 \preceq_h \tau_a\langle g_c\rangle$. Define a function $h' : \mathsf{V}'_c \mapsto V_{\tau_a\langle g_c\rangle}$ as follows:

$$\forall x \in \mathsf{V}'_c. \quad h'(x) = \begin{cases} h(x), & x \neq n \\ n_\ell, & x = n \end{cases}$$

To prove that h' induces an embedding from ptg'_1 to $\tau_a\langle g_c\rangle$, we need to prove the following two conditions (see (5)):

$$\langle x, f, y\rangle \in \mathsf{E}'_c \Rightarrow \langle h'(x), f, h'(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{118}$$

$$\forall v \in \mathit{Vars}.\ \sigma_{\tau'_a\langle g_c\rangle}(v) \supseteq \{h'(\sigma'_c(v))\} \tag{119}$$

(Proof of 118). Say $\langle x, f, y\rangle \in \mathsf{E}'_c$,

1. *Case(a)* $\langle x, f, y\rangle \in \mathsf{E}_c$

$$\text{By Lemma 15,} \quad \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{120}$$

$$\text{Since } \langle x, f, y\rangle \in \mathsf{E}_c, \quad x \neq n \wedge y \neq n \tag{121}$$

$$\text{By the def. of } h' \quad h'(x) = h(x) \wedge h'(y) = h(y) \tag{122}$$

$$\text{Sub in (120),} \quad \langle h'(x), f, h'(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{123}$$

2. *Case(b)* $\langle x, f, y\rangle \in \{n\} \times \mathit{Fields} \times \{\mathit{null}\}$

$$\text{In this case,} \quad x = n \wedge y = \mathit{null} \tag{124}$$

$$\text{By the def. of } h', \quad h'(x) = n_\ell \wedge h'(y) = \mathit{null} \tag{125}$$

$$\text{From the abstract semantics,} \quad \langle n_\ell, f, \mathit{null}\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{126}$$

$$\text{By (125),} \quad \langle h'(x), f, h'(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \tag{127}$$

(Proof of 119). Say $x = \sigma'_c(var)$,

1. *Case (a)*, $var \neq v$

$$\text{From the concrete semantics,} \quad x = \sigma_c(var) \wedge x \neq n \tag{128}$$

$$\text{By Lemma 15,} \quad h(x) \in \sigma_{\tau'_a\langle g_c\rangle}(var) \tag{129}$$

$$\text{Since } x \neq n, \quad h(x) = h'(x) \in \sigma_{\tau'_a\langle g_c\rangle}(var) \tag{130}$$

2. *Case (b)*, $var = v$

$$\text{From concrete semantics,} \quad x = n = \sigma'_c(v) \tag{131}$$

$$\text{From abstract semantics,} \quad n_\ell = \sigma'_a(v) \tag{132}$$

Since n_ℓ is an internal node, $\eta[\![\tau'_a, g_c]\!](n_\ell) = \{n_\ell\}$ (see *internal nodes* rule, Constraint 7). Hence,

$$\text{By the def. of } \langle\rangle, \quad \sigma_{\tau'_a\langle g_c\rangle}(v) = \eta(n_\ell) = n_\ell \tag{133}$$

$$\text{By the definition of } h', \quad h'(n) = n_\ell \tag{134}$$

$$\text{Sub. in (133),} \quad h'(n) = \sigma_{\tau'_a\langle g_c\rangle}(v) \tag{135}$$

$$\text{Since } v = var \text{ and } n = x, \quad h'(x) \in \sigma_{\tau'_a\langle g_c\rangle}(var) \tag{136}$$

Lemma 18. *For the statement $S : v_1.f = v_2$ the abstract semantics of S is a correct approximation of the concrete semantics of S.*

Proof. Let $\tau_a = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a)$ and $ptg_1 = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c)$. $[\![S]\!]_c(ptg_1)$ and $[\![S]\!]_a(\tau_a)$ are defined as follows:

$$[\![S]\!]_c(\mathsf{V}_c, \mathsf{E}_c, \sigma_c) = \{(\mathsf{V}_c,\ \{\langle u, l, v\rangle \in \mathsf{E}_c \mid u \neq \sigma_c(v_1) \vee l \neq f\} \cup \{\langle \sigma_c(v_1), f, \sigma_c(v_2)\rangle\},\ \sigma_c)\}$$

$$[\![S]\!]_a(\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a) = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a \cup \sigma_a(v_1) \times \{f\} \times \sigma_a(v_2), \sigma_a)$$

Let,

$$\begin{aligned} ptg_1' &= [\![S]\!]_c(ptg_1) = (\mathsf{V}_c', \mathsf{E}_c', \sigma_c') \\ \tau_a' &= [\![S]\!]_a(\tau_a) = (\mathsf{EV}_a', \mathsf{EE}_a', \pi_a', \mathsf{IV}_a', \mathsf{IE}_a', \sigma_a') \\ \tau_a\langle g_c\rangle &= \langle \mathsf{V}_{\tau_a\langle g_c\rangle}, \mathsf{E}_{\tau_a\langle g_c\rangle}, \sigma_{\tau_a\langle g_c\rangle}\rangle \\ \tau_a'\langle g_c\rangle &= \langle \mathsf{V}_{\tau_a'\langle g_c\rangle}, \mathsf{E}_{\tau_a'\langle g_c\rangle}, \sigma_{\tau_a'\langle g_c\rangle}\rangle \end{aligned}$$

From condition 99, we need to prove that,

$$\forall g_c \in \mathbb{G}_c,\ \forall ptg_1 \in \mathbb{G}_c,\ ptg_1 \preceq \tau_a\langle g_c\rangle \implies ptg_1' \preceq \tau_a'\langle g_c\rangle$$

Since, $ptg_1 \preceq \tau_a\langle g_c\rangle$, there exists a function h s.t., $ptg_1 \preceq_h \tau_a\langle g_c\rangle$. We prove that h induces an embedding from ptg_1' to $\tau_a\langle g_c\rangle$ by proving that the following holds:

$$\langle x, f, y\rangle \in \mathsf{E}_c' \implies \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau_a'\langle g_c\rangle} \tag{137}$$

$$\forall v \in \mathit{Vars}.\ \sigma_{\tau_a'\langle g_c\rangle}(v) \supseteq \{h(\sigma_c'(v))\} \tag{138}$$

(Proof of 137). Say $\langle x, f, y\rangle \in \mathsf{E}_c'$,

1. *Case(a)* $\langle x, f, y\rangle \in \mathsf{E}_c$.
   By Lemma 15, since $ptg_1 \preceq_h \tau_a\langle g_c\rangle$, $\langle h(x), f, h(y)\rangle \in \tau_a'\langle g_c\rangle$.

2. *Case(b)* $\langle x, f, y\rangle \in \langle \sigma_c(v_1), f, \sigma_c(v_2)\rangle$. In this case $x = \sigma_c(v_1)$ and $y = \sigma_c(v_2)$.

$$\text{Since } ptg_1 \preceq_h \tau_a\langle g_c\rangle, \quad h(\sigma_c(v_1)) \in \sigma_{\tau_a\langle g_c\rangle}(v_1) \tag{139}$$

$$\text{From the abs. semantics,} \quad \tau_a \preceq_{id} \tau_a' \tag{140}$$

$$\text{By Corollary 2,} \quad \tau_a\langle g_c\rangle \preceq_{id} \tau_a'\langle g_c\rangle \tag{141}$$

$$\text{By the def. of } \preceq, \quad \forall v \in \mathit{Vars}.\ \ \sigma_{\tau_a\langle g_c\rangle}(var) \subseteq \sigma_{\tau_a'\langle g_c\rangle}(var) \tag{142}$$

$$\text{The above implies that,} \quad \sigma_{\tau_a\langle g_c\rangle}(v_1) \subseteq \sigma_{\tau_a'\langle g_c\rangle}(v_1) \tag{143}$$

$$\text{By (139),} \quad h(\sigma_c(v_1)) \in \sigma_{\tau_a'\langle g_c\rangle}(v_1) \tag{144}$$

$$\text{From the def. of } \langle\rangle \quad h(\sigma_c(v_1)) \in \sigma_{\tau_a'\langle g_c\rangle}(v_1) \implies \exists w \text{ s.t. } h(\sigma_c(v_1)) \in \eta[\![\tau_a', g_c]\!](w) \wedge w \in \sigma_{\tau_a'}(v_1) \tag{145}$$

$$\text{Since } \sigma_{\tau_a'}(v_1) = \sigma_{\tau_a}(v_1), \quad \exists w\ s.t.\ h(\sigma_c(v_1)) \in \eta[\![\tau_a', g_c]\!](w) \wedge w \in \sigma_{\tau_a}(v_1) \tag{146}$$

$$\text{Similarly,} \quad \exists w,\ s.t.\ , h(\sigma_c(v_2)) \in \eta[\![\tau_a', g_c]\!](w) \wedge w \in \sigma_{\tau_a}(v_2) \tag{147}$$

From the abstract semantics, $\sigma_a(v_1) \times \{f\} \times \sigma_a(v_2) \in \mathsf{IE}'_a$. By the def. of $\langle\rangle$,

$$\begin{aligned} &\forall w_1 \in \sigma_{\tau_a}(v_1), \forall w_2 \in \sigma_{\tau_a}(v_2), \\ &\eta[\![\tau'_a, g_c]\!](w_1) \times \{f\} \times \eta[\![\tau'_a, g_c]\!](w_2) \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \end{aligned} \tag{148}$$

From (146), (147) and (148)

$$\begin{aligned} &\langle h(\sigma_c(v_1)), f, h(\sigma_c(v_2))\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \\ &\equiv \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau'_a\langle g_c\rangle} \end{aligned}$$

(Proof of 138). Say $x = \sigma'_c(var)$.

$$\text{Form the concrete semantics,} \quad x = \sigma_c(var) \tag{149}$$

$$\text{Given,} \quad ptg_1 \preceq_h \tau_a\langle g_c\rangle \tag{150}$$

$$\text{Hence, by Lemma 15,} \quad h(x) \in \sigma_{\tau'_a\langle g_c\rangle}(var) \tag{151}$$

To prove the remaining transfer functions correct we make use of the following lemmas.

Lemma 19. *Let $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$ be a transformer graph, $y \in \mathsf{EV} \implies y \in Escaping(\tau)$*

Proof. Say $y \in \mathsf{EV}$.
Case(a), $y \in range(\pi)$. By the def. of $Escaping(\tau)$, $y \in Escaping(\tau)$
Case(b), $y \notin range(\pi)$. By the definition of the transformer graph,

$$y \in \mathsf{EV} \wedge y \notin range(\pi) \iff \exists \langle x, f, y\rangle \in \mathsf{EE} \tag{152}$$

$$\langle x, f, y\rangle \in \mathsf{EE} \implies x \in Escaping(\tau) \tag{153}$$

By the definition of $Escaping(\tau)$, $x \in Escaping(\tau)$ and $\langle x, f, y\rangle \in \mathsf{EE}$ imply that $y \in Escaping(\tau)$.

Lemma 20. *Let $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$ be a transformer graph. Let $g_c \in \mathbb{G}_c$ be a concrete graph.*

$$x \in \mathsf{EV} \wedge y \in \mathsf{IV} \wedge y \in \eta[\![\tau, g_c]\!](x) \implies y \in Escaping(\tau)$$

Proof. We denote $\eta[\![\tau, g_c]\!]$ as η. As explained earlier, η is the least solution of the set of constraints (7)-(10) over variable μ. We assume that the least-solution is computed iteratively, starting from $\mu = \emptyset$. We denote the value of μ computed at the end of iteration $i, i \geq 0$ using μ^i ($\mu^0 = \emptyset$).

Claim. Given a $g_c = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c) \in \mathbb{G}_c$, $\forall x \in \mathsf{V}_1$.,

$$x \in \mathsf{EV} \wedge y \in \mathsf{IV} \wedge y \in \mu^i(x) \implies y \in Escaping(\tau) \tag{154}$$

Base case, iteration 1. $\exists x \in \mathsf{EV}, y \in \mathsf{IV}$ s.t. $y \in \mu^1(x)$ is not possible. From constraints (7)-(10), the only rule that maps an external node to an internal

node is the *alias rule* (10). But since $\mu^0 = \emptyset$, it does not apply in the base case. Hence, The claim holds trivially.

Assume that the claim holds for all iterations form 0 to $i-1$.

Iteration i. Say $\exists x \in \mathsf{EV}, y \in \mathsf{IV}$ s.t. $y \in \mu^i(x)$. If $y \in \mu^{i-1}(x)$ the claim holds by the inductive hypothesis. Say y was added by the *alias rule* in iteration i (y could not have been added by any other rule). In this case the following must hold.

$$\langle u, f, x\rangle \in \mathsf{EE} \wedge \langle r, f, s\rangle \in \mathsf{IE} \wedge \\ \mu^{i-1}(u) \cap \mu^{i-1}(r) \neq \emptyset \wedge y \in \mu^{i-1}(s) \tag{155}$$

Now consider the following two cases:
Case (a) $s \in \mathsf{EV}$. By the inductive hypothesis $y \in \mathit{Escaping}(\tau)$.
Case (b) $s \in \mathsf{IV}$.

$$s \in \mathsf{IV} \text{ implies that,} \quad \mu^{i-1}(s) = \{s\} \tag{156}$$
$$\text{Since, } y \in \mu^{i-1}(s) \quad y = s \tag{157}$$
$$\text{From (155),} \quad \langle r, f, y\rangle \in \mathsf{IE} \tag{158}$$

Case (b.1) $r \in \mathit{Escaping}(\tau)$. By the def. of $\mathit{Escaping}(\tau)$, (158) implies that $y \in \mathit{Escaping}(\tau)$.
Case (b.2) $r \notin \mathit{Escaping}(\tau)$.

$$\text{By Lemma 19,} \quad r \in \mathsf{IV} \tag{159}$$
$$\text{By rule (7),} \quad \mu^{i-1}(r) = \{r\} \tag{160}$$
$$\text{By (155),} \quad \mu^{i-1}(u) \cap \mu^{i-1}(r) \neq \emptyset \tag{161}$$
$$\text{From (160) and (161),} \quad r \in \mu^{i-1}(u) \tag{162}$$

If say $u \in \mathsf{EV}$ then, by the inductive hypothesis, $r \in \mathit{Escaping}(\tau)$ which is a contradiction to the assumption that $r \notin \mathit{Escaping}(\tau)$. Say, $u \in \mathsf{IV}$.

$$\text{Since, } u \in \mathsf{IV}, \quad \mu^{i-1}(u) = \{u\} \tag{163}$$
$$\text{From (162),} \quad r = u \tag{164}$$
$$\text{From (155),} \quad \langle u, f, x\rangle \in \mathsf{EE} \tag{165}$$
$$\text{Hence, from (164),} \quad \langle r, f, x\rangle \in \mathsf{EE} \tag{166}$$
$$\text{By the def. of transformer graphs,} \quad \langle r, f, x\rangle \in \mathsf{EE} \implies r \in \mathit{Escaping}(\tau) \tag{167}$$

(167) is again a contradiction to the assumption that $r \notin \mathit{Escaping}(\tau)$. Hence ***Case(b.2)*** is not possible.

Lemma 21. *Let $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$ be a transformer graph. Let $g_c \in \mathbb{G}_c$ be a concrete graph.*

$$\exists x, y \in (\mathsf{EV} \cup \mathsf{IV}) \text{ s.t.} \\ \eta[\![\tau, g_c]\!](x) \cap \eta[\![\tau, g_c]\!](y) \neq \emptyset \wedge x \neq y \implies x, y \in \mathit{Escaping}(\tau)$$

Proof. ***Case(a) Say both*** x ***and*** y ***are internal nodes***.

By the def. of η (rule *internal nodes*),

$$\eta[\![\tau, g_c]\!](x) = \{x\}$$
$$\eta[\![\tau, g_c]\!](y) = \{y\} \tag{168}$$
$$\text{Since, } x \neq y, \quad \eta[\![\tau, g_c]\!](x) \cap \eta[\![\tau, g_c]\!](y) = \emptyset \tag{169}$$

Since it is given that $\eta[\![\tau, g_c]\!](x) \cap \eta[\![\tau, g_c]\!](y) \neq \emptyset$, (169) is a contradiction. Hence, this case is not possible. At least one of x or y should be an external node.

Case(b) Say $x \in \mathsf{EV}$ ***and*** $y \in \mathsf{IV}$.

$$\text{By lemma 19,} \quad x \in \mathit{Escaping}(\tau) \tag{170}$$
$$\text{Since } y \in \mathsf{IV}, \quad \eta[\![\tau, g_c]\!](y) = \{y\} \tag{171}$$
$$\text{Given,} \quad \eta[\![\tau, g_c]\!](x) \cap \eta[\![\tau, g_c]\!](y) \neq \emptyset \tag{172}$$
$$\text{By (171) and (172),} \quad y \in \eta[\![\tau, g_c]\!](x) \tag{173}$$
$$\text{By Lemma 20,} \quad x \in \mathsf{EV} \wedge y \in \mathsf{IV} \wedge y \in \eta[\![\tau, g_c]\!](x)$$
$$\implies y \in \mathit{Escaping}(\tau) \tag{174}$$

Case(c) Say $x \in \mathsf{EV}$ ***and*** $y \in \mathsf{EV}$. By Lemma 19, $x, y \in \mathsf{EV} \implies x, y \in \mathit{Escaping}(\tau)$.

Lemma 22. *For the statement* $S : \ell : v_1 = v_2.f$*, the abstract semantics of* S *is a correct approximation of the concrete semantics of* S*.*

Proof. Let $\tau_a = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a)$ and $ptg_1 = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c)$. $[\![S]\!]_c(ptg_1)$ and $[\![S]\!]_a(\tau_a)$ are defined as follows:

$$[\![S]\!]_c(\mathsf{V}_c, \mathsf{E}_c, \sigma_c) = \quad \{(\mathsf{V}_c, \mathsf{E}_c, \sigma[v_1 \mapsto n]) \mid \langle \sigma_c(v_2), f, n\rangle \in \mathsf{E}_c\}$$
$$[\![S]\!]_a(\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a) =$$
$$\text{let} \quad A = \{n \mid \exists n_1 \in \sigma_a(v_2), \langle n_1, f, n\rangle \in \mathsf{IE}_a\} \quad \text{in}$$
$$\text{let} \quad B = \sigma_a(v_2) \cap \mathit{Escaping}(\tau) \quad \text{in}$$
$$\text{if} \quad (B = \phi)$$
$$\text{then} \quad (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a[v_1 \mapsto A])$$
$$\text{else} \quad (\mathsf{EV}_a \cup \{n_\ell\}, \mathsf{EE}_a \cup B \times \{f\} \times \{n_\ell\}, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a[v \mapsto A \cup \{n_\ell\}])$$

Let,

$$ptg_1' = [\![S]\!]_c(ptg_1) = (\mathsf{V}_c', \mathsf{E}_c', \sigma_c')$$
$$\tau_a' = [\![S]\!]_a(\tau_a) = (\mathsf{EV}_a', \mathsf{EE}_a', \pi_a', \mathsf{IV}_a', \mathsf{IE}_a', \sigma_a')$$
$$\tau_a\langle g_c\rangle = \langle \mathsf{V}_{\tau_a\langle g_c\rangle}, \mathsf{E}_{\tau_a\langle g_c\rangle}, \sigma_{\tau_a\langle g_c\rangle}\rangle$$
$$\tau_a'\langle g_c\rangle = \langle \mathsf{V}_{\tau_a'\langle g_c\rangle}, \mathsf{E}_{\tau_a'\langle g_c\rangle}, \sigma_{\tau_a'\langle g_c\rangle}\rangle$$

From (99), we need to prove that,

$$\forall g_c \in \mathbb{G}_c,\ \forall ptg_1 \in \mathbb{G}_c,\ ptg_1 \preceq \tau_a\langle g_c\rangle \implies ptg_1' \preceq \tau_a'\langle g_c\rangle$$

Since, $ptg_1 \preceq \tau_a\langle g_c\rangle$, there exists a function h s.t., $ptg_1 \preceq_h \tau_a\langle g_c\rangle$. We now prove that h induces an embedding from ptg_1' to $\tau_a\langle g_c\rangle$ by proving that the following two conditions hold:

$$\langle x, f, y\rangle \in \mathsf{E}_c' \implies \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau_a'\langle g_c\rangle} \tag{175}$$

$$\forall v \in \mathit{Vars}.\ \sigma_{\tau_a'\langle g_c\rangle}(v) \supseteq \{h(\sigma_c'(v))\} \tag{176}$$

Say $\langle x, f, y\rangle \in \mathsf{E}_c'$. From the concrete semantics, $\langle x, f, y\rangle \in \mathsf{E}_c$. By Lemma 15, $ptg_1 \preceq_h \tau_a\langle g_c\rangle$ implies that $\langle h(x), f, h(y)\rangle \in \mathsf{E}_{\tau_a'\langle g_c\rangle}$.

Say $x = \sigma_c'(var) \wedge var \neq v_1$. From the concrete semantics, $x = \sigma_c(var)$. By Lemma 15, $ptg_1 \preceq_h \tau_a\langle g_c\rangle$ implies that $h(x) \in \sigma_{\tau_a'\langle g_c\rangle}$.

Say $x = \sigma_c'(v_1)$.

$$\text{Form the concrete semantics,}\quad \sigma_c'(v_1) = n \text{ where } \langle \sigma_c(v_2), f, n\rangle \in \mathsf{E}_c \tag{177}$$

$$\text{Since } ptg_1 \preceq_h \tau_a\langle g_c\rangle,\quad \langle h(\sigma_c(v_2)), f, h(n)\rangle \in \mathsf{E}_{\tau_a\langle g_c\rangle} \tag{178}$$

By the definition of $\langle\rangle$, an edge $\langle h(\sigma_c(v_2)), f, h(n)\rangle$ belongs to $E_{\tau_a\langle g_c\rangle}$ only if at least one of the following two conditions holds.

$$\langle h(\sigma_c(v_2)), f, h(n)\rangle \in \mathsf{E}_c \tag{179}$$

$$\exists x, y \in (\mathsf{EV}_a \cup \mathsf{IV}_a),\ \text{s.t.}$$
$$h(\sigma_c(v_2)) \in \eta[\![\tau_a, g_c]\!](x) \wedge h(n) \in \eta[\![\tau_a, g_c]\!](y) \wedge \langle x, f, y\rangle \in \mathsf{IE}_a \tag{180}$$

Say Condition 179 holds i.e, $\langle h(\sigma_c(v_2)), f, h(n)\rangle \in \mathsf{E}_c$.

$$\text{Since } ptg_1 \preceq_h \tau_a\langle g_c\rangle,\quad h(\sigma_c(v_2)) \in \sigma_{\tau_a\langle g_c\rangle}(v_2) \tag{181}$$

$$\text{By the definition of } \langle\rangle,\quad \exists w \in \sigma_a(v_2) \text{ s.t.}$$
$$h(\sigma_c(v_2)) \in \eta[\![\tau_a, g_c]\!](w) \tag{182}$$

Since $\langle h(\sigma_c(v_2)), f, h(n)\rangle \in \mathsf{E}_c$, $h(\sigma_c(v_2)) \in \mathsf{V}_c$.

$$\text{By the definition of } \eta,\quad \exists t \in \mathsf{V}_c, t \in \eta[\![\tau_a, g_c]\!](x) \implies x \in \mathsf{EV}_a \tag{183}$$

$$\text{From (182) and (183)},\quad w \in \mathsf{EV}_a \tag{184}$$

$$\text{By lemma 19},\quad w \in \mathsf{EV}_a \implies w \in \mathit{Escaping}(\tau) \tag{185}$$

$$\text{Hence, by the abs. semantics},\quad \langle w, f, n_\ell\rangle \in \mathsf{EE}(\tau_a') \tag{186}$$

Since $\tau_a \preceq_{id}^{\{v_1\}} \tau_a'$, by Lemma 12, $h(\sigma_c(v_2)) \in \eta[\![\tau_a, g_c]\!](w)$ implies that,

$$h(\sigma_c(v_2)) \in \eta[\![\tau_a', g_c]\!](w) \tag{187}$$

We have assumed that $\langle h(\sigma_c(v_2)), f, h(n)\rangle \in \mathsf{E}_c$. This together with (186) and (187) imply that the *external edge* rule of η computation applies (see (9)).

Therefore,

$$h(n) \in \eta[\![\tau'_a, g_c]\!](n_\ell) \tag{188}$$
$$\text{By the abstract semantics,} \quad n_\ell \in \sigma'_a(v_1) \tag{189}$$
$$\text{By the def. of } \langle\rangle, \quad h(n) \in \sigma_{\tau'_a\langle g_c\rangle}(v_1) \tag{190}$$
$$\text{which is same as,} \quad h(\sigma'_c(v_1)) \in \sigma_{\tau'_a\langle g_c\rangle}(v_1) \tag{191}$$

Say Condition 180 holds i.e,

$$\exists x, y \in (\mathsf{EV}_a \cup \mathsf{IV}_a), \text{ s.t. } h(\sigma_c(v_2)) \in \eta[\![\tau_a, g_c]\!](x) \\ \wedge h(n) \in \eta[\![\tau_a, g_c]\!](y) \wedge \langle x, f, y\rangle \in \mathsf{IE}_a \tag{192}$$

$$\text{Since } ptg_1 \preceq_h \tau_a\langle g_c\rangle, \quad h(\sigma_c(v_2)) \in \sigma_{\tau_a\langle g_c\rangle}(v_2) \tag{193}$$
$$\text{By the definition of } \langle\rangle, \quad \exists w \in \sigma_a(v_2) \text{ s.t.}$$
$$h(\sigma_c(v_2)) \in \eta[\![\tau_a, g_c]\!](w) \tag{194}$$
$$\text{It is assumed that,} \quad \exists x, h(\sigma_c(v_2)) \in \eta[\![\tau_a, g_c]\!](x) \tag{195}$$
$$\text{Hence,} \quad \eta[\![\tau_a, g_c]\!](w) \cap \eta[\![\tau_a, g_c]\!](x) \neq \emptyset \tag{196}$$

Now there are two cases to consider $w \neq x$ and $w = x$.
Consider the case where $w = x$.

$$\text{By assumption,} \quad \langle x, f, y\rangle \in \mathsf{IE}_a \tag{197}$$
$$\text{By (194),} \quad w = x \in \sigma_a(v_2) \tag{198}$$
$$\text{Hence, from abstract semantics,} \quad y \in \sigma'_a(v_1) \tag{199}$$
$$\text{We have assumed that,} \quad h(n) \in \eta[\![\tau_a, g_c]\!](y) \tag{200}$$
$$\text{Since } \tau_a \preceq_{id}^{\{v_1\}} \tau'_a, \quad h(n) \in \eta[\![\tau'_a, g_c]\!](y) \tag{201}$$

By the definition of $\langle\rangle$, $y \in \sigma'_a(v_1)$ (199) and $h(n) \in \eta[\![\tau'_a, g_c]\!](y)$ (201) imply that $h(n) \in \sigma_{\tau_a\langle g_c\rangle}(v_1)$ as required.

Consider the case where $w \neq x$.

$$\text{By Lemma 21, (196) implies that} \quad w, x \in \mathit{Escaping}(\tau) \tag{202}$$
$$\text{By the abs. semantics,} \quad \langle w, f, n_\ell\rangle \in \mathsf{EE}'_a \tag{203}$$
$$\text{By assumption,} \quad \langle x, f, y\rangle \in \mathsf{IE}_a \tag{204}$$
$$\text{Hence, From the abs. semantics,} \quad \langle x, f, y\rangle \in \mathsf{IE}'_a \tag{205}$$

Since, $\tau_a \preceq_{id}^{\{v_1\}} \tau'_a$, $\eta[\![\tau_a, g_c]\!](w) \cap \eta[\![\tau_a, g_c]\!](x) \neq \emptyset$ implies that

$$\eta[\![\tau'_a, g_c]\!](w) \cap \eta[\![\tau'_a, g_c]\!](x) \neq \emptyset \tag{206}$$

(203),(205) and (206) imply that the *alias* rule of η computation applies (see 10). Hence,

$$\eta[\![\tau'_a, g_c]\!](y) \subseteq \eta[\![\tau'_a, g_c]\!](n_\ell) \quad (207)$$

$$\text{Since } h(n) \in \eta[\![\tau'_a, g_c]\!](y) \quad h(n) \in \eta[\![\tau'_a, g_c]\!](n_\ell) \quad (208)$$

$$\text{By the abstract semantics,} \quad n_\ell \in \sigma'_a(v_1) \quad (209)$$

$$\text{By the def. of } \langle\rangle, \quad h(n) \in \sigma_{\tau_a\langle g_c\rangle}(v_1) \quad (210)$$

$$\text{which is same as,} \quad h(\sigma'_c(v_1)) \in \sigma_{\tau'_a\langle g_c\rangle}(v_1) \quad (211)$$

8.4 Abstract Semantics Of Procedure Call (Lemma 2)

In this section we prove the correctness of the procedure call transfer function (i.e, summary application). Let S be a procedure call $Call\ P(v_1, \cdots, v_k)$. Let $\tau_r = (\mathsf{EV}_r, \mathsf{EE}_r, \pi_r, \mathsf{IV}_r, \mathsf{IE}_r, \sigma_r)$ be the transformer graph at a program point before the call and let $\tau_e = (\mathsf{EV}_e, \mathsf{EE}_e, \pi_e, \mathsf{IV}_e, \mathsf{IE}_e, \sigma_e)$ be the summary of P.

Let $\tau_e\langle\!\langle \tau_r, \nu \rangle\!\rangle = (\mathsf{EV}', \mathsf{EE}', \pi', \mathsf{IV}', \mathsf{IE}', \sigma')$, where $\nu : (\mathsf{IV}_e \cup \mathsf{IE}_e) \mapsto (\mathsf{IV}_r \cup \mathsf{EV}_r \cup \mathsf{IV}_e \cup \mathsf{EV}_e))$,

$$\begin{aligned}
\mathsf{V}' &= (\mathsf{IV}_r \cup \mathsf{EV}_r) \cup \hat{\nu}(\mathsf{IV}_e \cup \mathsf{EV}_e) \\
\mathsf{IV}' &= \mathsf{V}' \cap (\mathsf{IV}_r \cup \mathsf{IV}_e) \\
\mathsf{EV}' &= \mathsf{V}' \cap (\mathsf{EV}_r \cup \mathsf{EV}_e) \\
\mathsf{IE}' &= \mathsf{IE}_r \cup \{\langle v_1, f, v_2\rangle \mid \langle u, f, v\rangle \in \mathsf{IE}_e, v_1 \in \nu(u), v_2 \in \nu(v)\} \\
\mathsf{EE}' &= \mathsf{EE}_r \cup \{\langle u', f, v\rangle \mid \langle u, f, v\rangle \in \mathsf{EE}_e, u' \in \nu(u), escapes(u')\} \\
\pi' &= \pi_r \\
\sigma' &= \lambda x.\ \hat{\nu}(\sigma_e(x)) \\
escapes(v) &\equiv \exists u \in range(\pi').v \text{ is reachable from } u \text{ via } \mathsf{IE}' \cup \mathsf{EE}' \text{ edges}
\end{aligned}$$

Let η_a be the least solution of the set of following constraints over the variable μ_a.

$$x \in \mathsf{IV}_e \Rightarrow x \in \mu_a(x) \quad (212)$$

$$x \in \pi(var) \Rightarrow \sigma_r(var) \subseteq \mu_a(x) \quad (213)$$

$$\langle u, f, v\rangle \in \mathsf{EE}_e, u' \in \mu_a(u), \langle u', f, v'\rangle \in \mathsf{IE}_r \Rightarrow v' \in \mu_a(v) \quad (214)$$

$$\langle u, f, v\rangle \in \mathsf{EE}_e, \mu_a(u) \cap \mu_a(u') \neq \{\}, \langle u', f, v'\rangle \in \mathsf{IE}_e \Rightarrow \mu_a(v') \subseteq \mu_a(v) \quad (215)$$

$$\langle u, f, v\rangle \in \mathsf{EE}_e, \mu_a(u) \cap Escaping(\tau_e\langle\!\langle \tau_r, \mu_a \rangle\!\rangle) \neq \{\} \Rightarrow v \in \mu_a(v) \quad (216)$$

We now define $push_S$, pop_S operations for the transformer graphs as follows:

$$push_S(\sigma) = \lambda v.\ v \in Globals \rightarrow \sigma(v) \mid v \in Locals \rightarrow null \mid v = Param(i) \rightarrow \sigma(a_i)$$

$$pop_S(\sigma, \sigma') = \lambda v.\ v \in Globals \rightarrow \sigma'(v) \mid v \in Locals \cup Params \rightarrow \sigma(v)$$

$$push_S((\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1)) = (\mathsf{EV}_1, \mathsf{IV}_1, \mathsf{EE}_1, \mathsf{IE}_1, push_S(\sigma_1))$$

$$pop_S((\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1), (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2)) = (\mathsf{EV}_2, \mathsf{IV}_2, \mathsf{EE}_2, \mathsf{IE}_2, pop_S(\sigma_1, \sigma_2))$$

The transformer graph after the procedure call statement S is given by

$$pop_S(\tau_r, \tau_e\langle\langle push_S(\tau_r)\rangle\rangle) \tag{217}$$

$$\text{where, } \tau_e\langle\langle\tau_r\rangle\rangle = \tau_e\langle\langle\tau_r, \eta_a\rangle\rangle \tag{218}$$

Note that the above definition of abstract semantics of a call statement is semantically equivalent to the definition presented earlier in Section 3 (i.e, $\tau_e\langle\langle\tau_r\rangle\rangle_a^S$) but unlike $\tau_e\langle\langle\tau_r\rangle\rangle_a^S$ which incorporated the push and pop operations into the operator $\langle\langle\rangle\rangle$ this definition makes them explicit similar to the concrete semantics.

As mentioned in the Section 3, *Globals* denote the set of global variables in the program, *Params* denote the set of parameter variables and *Locals* denote the set of local variables. We assume that all the procedures use the same set of parameters and local variables.

In the rest of the proof we use the following naming convention.

$$\begin{aligned}
\tau_e\langle\langle\tau_r\rangle\rangle &= (\mathsf{EV}_{\mathsf{absx}}, \mathsf{EE}_{\mathsf{absx}}, \pi_{absx}, \mathsf{IV}_{\mathsf{absx}}, \mathsf{IE}_{\mathsf{absx}}, \sigma_{absx}) \\
g_c &= (\mathsf{V}_c, \mathsf{E}_c, \sigma_c) \\
ptg_1 &= (\mathsf{V}_{c_1}, \mathsf{E}_{c_1}, \sigma_{c_1}) \\
\tau_r\langle g_c\rangle &= (\mathsf{V}_r^c, \mathsf{E}_r^c, \sigma_r^c) \\
\tau_e\langle ptg_1\rangle &= (\mathsf{V}_e^1, \mathsf{E}_e^1, \sigma_e^1) \\
(\tau_e\langle\langle\tau_r\rangle\rangle)\langle g_c\rangle &= (\mathsf{V}_{\mathsf{absx}}{}^c, \mathsf{E}_{\mathsf{absx}}{}^c, \sigma_{absx}^c)
\end{aligned}$$

To prove that the summary application is correct we need to show that,

$$f_r \sim \tau_r \wedge f_e \sim \tau_e \implies f_r \circ \mathit{CallReturn}_S(f_e) \sim pop_S(\tau_r, \tau_e\langle\langle push_S(\tau_r)\rangle\rangle) \tag{219}$$

However, it suffices to prove constraint shown below (details omitted for brevity).

$$\forall g_c, ptg_1 \in \mathbb{G}_c, \quad ptg_1 \preceq \tau_r\langle g_c\rangle \implies \tau_e\langle ptg_1\rangle \preceq (\tau_e\langle\langle\tau_r\rangle\rangle)\langle g_c\rangle \tag{220}$$

Since $ptg_1 \preceq \tau_r\langle g_c\rangle$, there exists a function h s.t., $ptg_1 \preceq_h \tau_r\langle g_c\rangle$. Define a function $h' : \mathsf{V}_e^1 \mapsto \mathsf{V}_{\mathsf{absx}}{}^c$ as follows (note that $\mathsf{V}_e^1 = \mathsf{V}_{c_1} \uplus \mathsf{IV}_e$):

$$\forall x \in \mathsf{V}_e^1. \quad h'(x) = \begin{cases} h(x), & x \in \mathsf{V}_{c_1} \\ x, & x \in \mathsf{IV}_e \end{cases}$$

Lemma 23. $\tau_r \preceq_{id}^{Vars} \tau_e\langle\langle\tau_r\rangle\rangle_a^S$

Proof. Directly follows from the definition of $\tau_e\langle\langle\tau_r\rangle\rangle_a^S = \tau_e\langle\langle\tau_r, \eta_a\rangle\rangle$.

Lemma 24.

$\forall g_c, ptg_1 \in \mathbb{G}_c,\ \forall x \in (\mathsf{IV}_e \cup \mathsf{EV}_e)$,

$$ptg_1 \preceq_h \tau_r\langle g_c\rangle \wedge u \in \eta[\![\tau_e, ptg_1]\!](x) \implies (\exists n \in \eta_a(x)\ s.t.\ h'(u) \in \eta[\![\tau_e\langle\langle\tau_r\rangle\rangle, g_c]\!](n))$$

Proof. We prove this by inducting over the iterations in the computation of $\eta[\![\tau_e, ptg_1]\!]$. If μ^i denotes the value of μ computed at the end of iteration $i, i \geq 0$ then,

$$\eta[\![\tau_e, ptg_1]\!] = \mu^j[\![\tau_e, ptg_1]\!] \text{ where,}$$
$$j \text{ is the smallest integer such that } \mu^{j-1} = \mu^j. \tag{221}$$

Claim. Given $g_c = (\mathsf{V}_c, \mathsf{E}_c, \sigma_c) \in \mathbb{G}_c$, $x \in (\mathsf{IV}_e \cup \mathsf{EV}_e)$,

$$ptg_1 \preceq \tau_r\langle g_c\rangle \wedge u \in \mu^i[\![\tau_e, ptg_1]\!](x) \implies$$
$$(\exists n \in \eta_a(x)\ s.t.\ h'(u) \in \eta[\![\tau_e\langle\!\langle\tau_r\rangle\!\rangle, g_c]\!](n)) \tag{222}$$

In (222), η_a is the least solution of the constraints 212-216. Hence the following holds.

$$x \in \mathsf{IV}_e \Rightarrow x \in \eta_a(x) \tag{223}$$
$$x \in \pi_e(var) \Rightarrow \sigma_r(var) \subseteq \eta_a(x) \tag{224}$$
$$\langle u, f, v\rangle \in \mathsf{EE}_e, u' \in \eta_a(u), \langle u', f, v'\rangle \in \mathsf{IE}_r \Rightarrow v' \in \eta_a(v) \tag{225}$$
$$\langle u, f, v\rangle \in \mathsf{EE}_e, \eta_a(u) \cap \eta_a(u') \neq \{\}, \langle u', f, v'\rangle \in \mathsf{IE}_e \Rightarrow \eta_a(v') \subseteq \eta_a(v) \tag{226}$$
$$\langle u, f, v\rangle \in \mathsf{EE}_e, \eta_a(u) \cap \mathit{Escaping}(\tau_e\langle\!\langle\tau_r, \eta_a\rangle\!\rangle) \neq \{\} \Rightarrow v \in \eta_a(v) \tag{227}$$

Since $\tau_e\langle\!\langle\tau_r\rangle\!\rangle = \tau_e\langle\!\langle\tau_r, \eta_a\rangle\!\rangle$ the following holds,

$$\mathsf{V}_{\mathsf{absx}} = (\mathsf{IV}_r \cup \mathsf{EV}_r) \cup \hat{\eta_a}(\mathsf{IV}_e \cup \mathsf{EV}_e) \tag{228}$$
$$\mathsf{IV}_{\mathsf{absx}} = \mathsf{V}_{\mathsf{absx}} \cap (\mathsf{IV}_r \cup \mathsf{IV}_e) \tag{229}$$
$$\mathsf{EV}_{\mathsf{absx}} = \mathsf{V}_{\mathsf{absx}} \cap (\mathsf{EV}_r \cup \mathsf{EV}_e) \tag{230}$$
$$\mathsf{IE}_{\mathsf{absx}} = \mathsf{IE}_r \cup \{\langle v_1, f, v_2\rangle \mid \langle u, f, v\rangle \in \mathsf{IE}_e, v_1 \in \eta_a(u), v_2 \in \eta_a(v)\} \tag{231}$$
$$\mathsf{EE}_{\mathsf{absx}} = \mathsf{EE}_r \cup \{\langle u', f, v\rangle \mid \langle u, f, v\rangle \in \mathsf{EE}_e, u' \in \eta_a(u), \mathit{escapes}(u')\} \tag{232}$$
$$\pi_{absx} = \pi_r \tag{233}$$
$$\sigma_{absx} = \lambda x.\ \hat{\eta_a}(\sigma_e(x)) \tag{234}$$
$$\mathit{escapes}(v) \equiv v \in \mathit{Escaping}(\tau_e\langle\!\langle\tau_r\rangle\!\rangle_a^S) \tag{235}$$

Also, since $\eta[\![\tau_e\langle\!\langle\tau_r\rangle\!\rangle, g_c]\!]$ is a least solution of the constraints 7-10 the following holds. In the rest of the proof we denote $\eta[\![\tau_e\langle\!\langle\tau_r\rangle\!\rangle, g_c]\!]$ as η'.

$$x \in \mathsf{IV}_{\mathsf{absx}} \Rightarrow x \in \eta'(x) \tag{236}$$
$$x \in \pi_{absx}(\mathtt{V}) \Rightarrow \sigma_c(\mathtt{V}) \in \eta'(x) \tag{237}$$
$$\langle x, f, y\rangle \in \mathsf{EE}_{\mathsf{absx}},\ x' \in \eta'(x),\ \langle x', f, y'\rangle \in \mathsf{E}_c \Rightarrow y' \in \eta'(y) \tag{238}$$
$$\langle x, f, y\rangle \in \mathsf{EE}_{\mathsf{absx}},\ \eta'(x) \cap \eta'(x') \neq \emptyset,\ \langle x', f, y'\rangle \in \mathsf{IE}_{\mathsf{absx}} \Rightarrow \eta'(y') \subseteq \eta'(y) \tag{239}$$

Base case (iteration 1). Say $y \in \mu^1[\![\tau_e, ptg_1]\!](x)$. Since $\mu^0 = \emptyset$, y should have been added to μ^1 by one of the rules *internal nodes* or *parameter nodes* (Constraints 7,8).

Say y was added by the *Internal nodes* rule (Constraint 7). In this case, the following must hold.

$$x \in \mathsf{IV}_e \wedge x = y \tag{240}$$

We need to show that $\exists n \in \eta_a(x)\ s.t.\ h'(y) \in \eta[\![\tau_e\langle\!\langle \tau_r \rangle\!\rangle, g_c]\!](n)$.

$$\begin{aligned}
\text{By (223)}, \quad & x \in \mathsf{IV}_e \implies x \in \eta_a(x) && (241)\\
\text{By the (229)} \quad & x \in \eta_a(x) \wedge x \in \mathsf{IV}_e \implies x \in \mathsf{IV}_{\mathsf{absx}} && (242)\\
\text{By (236)} \quad & x \in \mathsf{IV}_{\mathsf{absx}} \implies x \in \eta'(x) && (243)\\
\text{By the def. of } h' \quad & x \in \mathsf{IV}_e \implies h'(x) = x && (244)\\
\text{Sub. in (243)}, \quad & h'(x) \in \eta'(x) && (245)\\
\text{Since } x = y, \quad & h'(y) \in \eta'(x) && (246)
\end{aligned}$$

From (241) and (245), $\exists n \in \eta_a(x)\ s.t.\ x \in \eta'(n)$. In this case $n = x$.

Now say y was added to $\mu^1[\![\tau_e, ptg_1]\!](x)$ by the *Param nodes* rule (constraint 8). Hence,

$$\begin{aligned}
& x \in \pi_e(var) \wedge y = \sigma_{c_1}(var) && (247)\\
\text{Since } ptg_1 \preceq_h \tau_r\langle g_c \rangle, \quad & h(\sigma_{c_1}(var)) \in \sigma_r^c(var) && (248)\\
\text{By the def. of } \langle\rangle, \quad & \exists w \in \sigma_r(var),\ h(\sigma_{c_1}(var)) \in \eta[\![\tau_r, g_c]\!](w) && (249)\\
\text{By Lemma 23}, \quad & \tau_r \preceq_{id}^{Vars} \tau_e\langle\!\langle \tau_r \rangle\!\rangle_a^S && (250)\\
\text{By Lemma 12}, \quad & \eta[\![\tau_r, g_c]\!](w) \subseteq \eta'(w) && (251)\\
\text{Sub. in (249)}, \quad & \exists w \in \sigma_r(var),\ h(\sigma_{c_1}(var)) \in \eta'(w) && (252)\\
\text{By (224)}, \quad & x \in \pi_e(var) \implies \sigma_r(var) \subseteq \eta_a(x) && (253)\\
\text{Sub. in (252)}, \quad & \exists w \in \eta_a(v),\ h(\sigma_{c_1}(v)) \in \eta'(w) && (254)\\
\text{Since } \sigma_{c_1}(v) \in \mathsf{V}_{c_1} \quad & h'(\sigma_{c_1}(v)) = h(\sigma_{c_1}(v)) && (255)\\
\text{Sub. in (254)}, \quad & \exists w \in \eta_a(v),\ h'(\sigma_{c_1}(v)) \in \eta'(w) && (256)\\
\text{By (247)}, \quad & \exists w \in \eta_a(v),\ h'(y)) \in \eta'(w) && (257)
\end{aligned}$$

Hence the claim holds in the base case.

Induction Step. Assume that the claim holds for all iterations from 1 to $i-1$. Now consider the case where y' is added in the i^{th} iteration by one of the four rules (7-10). If y' is added by the *internal nodes* rule or the *parameter nodes* rule then the claim holds by the proof shown for the base case. Hence, consider the cases wherein y' is added by the *external edge* rule or the *alias* rule.

Case(1) Say y' is added by the *external edge* rule (9). The following must hold in this case.

$$\langle x, f, y \rangle \in \mathsf{EE}_e \wedge x' \in \mu^{i-1}[\![\tau_e, ptg_1]\!](x) \wedge \langle x', f, y' \rangle \in \mathsf{E}_{c_1} \tag{258}$$

We need to show that,

$$\exists n \in \eta_a(y)\ s.t.\ h'(y') \in \eta'(n) \tag{259}$$

Since $x' \in \mu^{i-1}[\![\tau_e, ptg_1]\!](x)$, by the inductive hypothesis,

$$\exists m \in \eta_a(x) \; s.t. \; h'(x') \in \eta'(m) \tag{260}$$

Now consider the following two cases:

Case(1.1) $m \in Escaping(\tau_e \langle\!\langle \tau_r \rangle\!\rangle)$.

$$\text{Since } m \in \eta_a(x), \quad \eta_a(x) \cap Escaping(\tau_e \langle\!\langle \tau_r, \eta_a \rangle\!\rangle) \neq \emptyset \tag{261}$$

$$\text{Given}, \quad \langle x, f, y \rangle \in \mathsf{EE}_e \tag{262}$$

By the Constraint (227), (261) and (262) implies that, $y \in \eta_a(y)$. Hence, to prove (259) it suffices to prove that,

$$h'(y') \in \eta[\![\tau_e \langle\!\langle \tau_r \rangle\!\rangle, g_c]\!](y)$$

$$\text{Given}, \quad \langle x', f, y' \rangle \in \mathsf{E}_{c_1} \tag{263}$$

$$\text{Since, } ptg_1 \preceq_h \tau_r \langle g_c \rangle, \quad \langle h(x'), f, h(y') \rangle \in \mathsf{E}_r^c \tag{264}$$

$$\text{Since, } x', y' \in \mathsf{V}_{c_1}, \quad h(x') = h'(x') \wedge h(y') = h'(y') \tag{265}$$

$$\text{Hence}, \quad \langle h'(x'), f, h'(y') \rangle \in \mathsf{E}_r^c \tag{266}$$

By the definition of $\langle\rangle$, $\langle h'(x'), f, h'(y') \rangle \in \mathsf{E}_r^c$ only because at least one of the following conditions hold.

$$\text{(i)} \quad \langle h'(x'), f, h'(y') \rangle \in \mathsf{E}_c \tag{267}$$

$$\text{(ii)} \quad \exists w, u \; s.t. \; h'(x') \in \eta[\![\tau_r, g_c]\!](w) \wedge h'(y') \in \eta[\![\tau_r, g_c]\!](u) \quad \langle w, f, u \rangle \in \mathsf{IE}_r \tag{268}$$

(i) Say $\langle h'(x'), f, h'(y') \rangle \in \mathsf{E}_c$.

$$\text{Given}, \quad \langle x, f, y \rangle \in \mathsf{EE}_e \tag{269}$$

$$\text{We have assumed that}, \quad m \in \eta_a(x) \wedge m \in Escaping(\tau_e \langle\!\langle \tau_r \rangle\!\rangle) \tag{270}$$

By Constraint (232), (269) and (270) imply that, $\langle m, f, y \rangle \in \mathsf{EE}_{\mathsf{absx}}$.

$$\text{By (260)}, \quad h'(x') \in \eta'(m) \tag{271}$$

$$\text{We have assumed that}, \quad \langle h'(x'), f, h'(y') \rangle \in \mathsf{E}_c \tag{272}$$

By the *external edge* rule of η construction, (271),(272) and $\langle m, f, y \rangle \in \mathsf{EE}_{\mathsf{absx}}$ imply that $h'(y') \in \eta'(y)$.

(ii) Consider the case where,

$$\exists w, u \; s.t. \; h'(x') \in \eta[\![\tau_r, g_c]\!](w) \wedge h'(y') \in \eta[\![\tau_r, g_c]\!](u) \wedge \langle w, f, u \rangle \in \mathsf{IE}_r$$

$$\text{By Lemma 23}, \quad \tau_r \preceq_{id}^{Vars} \tau_e \langle\!\langle \tau_r \rangle\!\rangle \tag{273}$$

$$\text{Hence}, \quad h'(x') \in \eta[\![\tau_r, g_c]\!](w) \implies h'(x') \in \eta'(w), \tag{274}$$

$$h'(y') \in \eta[\![\tau_r, g_c]\!](u) \implies h'(y') \in \eta'(u) \tag{275}$$

$$\text{By (260)}, \quad h'(x') \in \eta'(m) \wedge m \in \eta_a(x) \tag{276}$$

Hence, from (274) and (276), $\eta'(m) \cap \eta'(w) \neq \emptyset$.

$$\text{Given,} \quad \langle x, f, y\rangle \in \mathsf{EE}_e \tag{277}$$

$$\text{We have assumed that,} \quad m \in \eta_a(x) \wedge m \in \mathit{Escaping}(\tau_e\langle\!\langle \tau_r \rangle\!\rangle) \tag{278}$$

By Constraint (232), (277) and (278) imply that, $\langle m, f, y\rangle \in \mathsf{EE}_{\mathsf{absx}}$. By Constraint (231), $\langle w, f, u\rangle \in \mathsf{IE}_r$ implies that $\langle w, f, u\rangle \in \mathsf{IE}_{\mathsf{absx}}$.

Hence, $\eta'(m) \cap \eta'(w) \neq \emptyset$, $\langle m, f, y\rangle \in \mathsf{EE}_{\mathsf{absx}}$ and $\langle w, f, u\rangle \in \mathsf{IE}_{\mathsf{absx}}$. Therefore, by (239),

$$\eta'(u) \subseteq \eta'(y) \tag{279}$$

$$\text{By (275),} \quad h'(y') \in \eta'(u) \tag{280}$$

$$\text{By (279),} \quad h'(y') \in \eta'(y) \tag{281}$$

Case(1.2) $m \notin \mathit{Escaping}(\tau_e\langle\!\langle \tau_r \rangle\!\rangle)$.

$$\text{By Lemma 19,} \quad m \in \mathsf{IV}_{\mathsf{absx}} \tag{282}$$

$$\text{By the def. of } \eta, \quad \eta'(m) = \{m\} \tag{283}$$

$$\text{By (260),} \quad h'(x') \in \eta'(m) \wedge m \in \eta_a(x) \tag{284}$$

$$\text{By (283),} \quad h'(x') = m \tag{285}$$

It is given that $\langle h'(x'), f, h'(y')\rangle \in \mathsf{E}_r^c$. By (285), $\langle m, f, h'(y')\rangle \in \mathsf{E}_r^c$. By the definition of $\langle\rangle$, $\langle m, f, h'(y')\rangle \in \mathsf{E}_r^c$ only if at least one of the following conditions hold.

$$\text{(i)} \quad \langle m, f, h'(y')\rangle \in \mathsf{E}_c \tag{286}$$

$$\text{(ii)} \quad \exists w, u \; s.t. \; m \in \eta[\![\tau_r, g_c]\!](w), \; h'(y') \in \eta[\![\tau_r, g_c]\!](u), \; \langle w, f, u\rangle \in \mathsf{IE}_r \tag{287}$$

Since $m \in \mathsf{IV}_{\mathsf{absx}}$, case(i) is not possible (as $m \notin \mathsf{V}_c$). Hence, case(II) is only possible.

$$\text{By Lemma 23,} \quad \tau_r \preceq_{id}^{Vars} \tau_e\langle\!\langle \tau_r \rangle\!\rangle \tag{288}$$

$$\text{by Lemma 12,} \quad m \in \eta'(w) \tag{289}$$

$$\text{By (283),} \quad m \in \eta'(m) \tag{290}$$

$$\text{Hence,} \quad \eta'(w) \cap \eta'(m) \neq \emptyset \tag{291}$$

$$\text{By Lemma 21 and (291),} \quad \text{if } w \neq m \text{ then } w, m \in \mathit{Escaping}(\tau_e\langle\!\langle \tau_r \rangle\!\rangle) \tag{292}$$

$$\text{But it is given that,} \quad m \notin \mathit{Escaping}(\tau_e\langle\!\langle \tau_r \rangle\!\rangle) \tag{293}$$

$$\text{Hence,} \quad w = m \tag{294}$$

$$\text{By (287) and (294),} \quad \exists u \; s.t. \; h'(y') \in \eta[\![\tau_r, g_c]\!](u), \; \langle m, f, u\rangle \in \mathsf{IE}_r \tag{295}$$

$$\text{By Lemma 23,} \quad \tau_r \preceq_{id}^{Vars} \tau_e\langle\!\langle \tau_r \rangle\!\rangle \tag{296}$$

$$\text{by Lemma 12,} \quad h'(y') \in \eta[\![\tau_r, g_c]\!](u) \implies h'(y') \in \eta'(u) \tag{297}$$

$$\text{Given,} \quad m \in \eta_a(x), \langle x, f, y\rangle \in \mathsf{EE}_e, \langle m, f, u\rangle \in \mathsf{IE}_r \tag{298}$$

$$\text{By Constraint (225),} \quad u \in \eta_a(y) \tag{299}$$

$$\text{From (297) and (299),} \quad \exists u \in \eta_a(y) \; s.t. \; h'(y') \in \eta'(u) \tag{300}$$

Case(2) Now say t is added to $\mu^i[\![\tau_e, ptg_1]\!](y)$ by the *(alias rule)* (constraint 57). Therefore, the following must hold.

$$\begin{aligned}\langle x, f, y\rangle \in \mathsf{EE}_e \wedge \langle x', f, y'\rangle \in \mathsf{IE}_e \wedge \\ \mu^{i-1}[\![\tau_e, ptg_1]\!](x) \cap \mu^{i-1}[\![\tau_e, ptg_1]\!](x') \neq \emptyset \wedge \\ t \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y')\end{aligned} \tag{301}$$

We need to show that,

$$t \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y') \implies \exists n \in \eta_a(y) \; s.t. \; h'(t) \in \eta'(n) \tag{302}$$

By the inductive hypothesis,

$$s \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y') \implies \exists m \in \eta_a(y') \text{ s.t. } h'(s) \in \eta'(m) \tag{303}$$

$$\begin{aligned}\mu^{i-1}[\![\tau_e, ptg_1]\!](x) \cap \mu^{i-1}[\![\tau_e, ptg_1]\!](x') \neq \emptyset \implies \\ \exists u \in \eta_a(x), \; w \in \eta_a(x') \; s.t. \; \eta'(u) \cap \eta'(w) \neq \emptyset\end{aligned} \tag{304}$$

Case(2.1) $u \in Escaping(\tau_e \langle\!\langle \tau_r \rangle\!\rangle)$.

$$\text{Since } u \in \eta_a(x), \quad \eta_a(x) \cap Escaping(\tau_e \langle\!\langle \tau_r, \eta_a \rangle\!\rangle) \neq \emptyset \tag{305}$$

$$\text{Given}, \quad \langle x, f, y\rangle \in \mathsf{EE}_e \tag{306}$$

By Constraint 227, (305) and (306) implies that $y \in \eta_a(y)$. Hence, to prove (302) it suffices to prove that,

$$t \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y') \implies h'(t) \in \eta'(y) \tag{307}$$

We have,

$$\langle x, f, y \in \mathsf{EE}_e\rangle \tag{308}$$

$$\text{By (304)}, \quad u \in \eta_a(x) \tag{309}$$

$$u \in Escaping(\tau_e \langle\!\langle \tau_r, \eta_a \rangle\!\rangle) \tag{310}$$

$$\text{By Constraint (232)}, \quad \langle u, f, y\rangle \in \mathsf{EE}_{\mathsf{absx}} \tag{311}$$

$$\text{Given}, \quad \langle x', f, y'\rangle \in \mathsf{IE}_e \tag{312}$$

$$\text{By Constraint (231)}, \quad \eta_a(x') \times f \times \eta_a(y') \in \mathsf{IE}_{\mathsf{absx}} \tag{313}$$

$$\text{By (304)}, \quad w \in \eta_a(x') \tag{314}$$

$$\text{By (313) and (314)}, \quad \{w\} \times f \times \eta_a(y') \in \mathsf{IE}_{\mathsf{absx}} \tag{315}$$

By the Constraint (226), (304),(311) and (315) imply that,

$$\forall z \in \eta_a(y'), \quad \eta'(z) \subseteq \eta'(y) \tag{316}$$

Form (303),

$$s \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y') \implies \exists m \in \eta_a(y') \text{ s.t. } h'(s) \in \eta'(m) \tag{317}$$

Therefore, from (316) and (317), we have

$$s \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y') \implies h'(s) \in \eta'(y) \tag{318}$$

Case(2.2) $u \notin Escaping(\tau_e\langle\!\langle \tau_r \rangle\!\rangle)$.

$$\begin{aligned}
\text{By Lemma 19,} \quad & u \in \mathsf{IV}_{\mathsf{absx}} && (319)\\
\text{By the def. of } \langle\rangle, \quad & \eta'(u) = \{u\} && (320)\\
\text{Sub. in (304), we get,} \quad & u \in \eta'(w) \text{ where, } w \in \eta_a(x') && (321)\\
\text{From (320) and (321),} \quad & \eta'(w) \cap \eta'(u) \neq \emptyset && (322)\\
\text{By Lemma 21,} \quad & u \neq w \wedge \eta'(w) \cap \eta'(u) \neq \emptyset \\
& \qquad \implies u, w \in Escaping(\tau_e\langle\!\langle \tau_r \rangle\!\rangle) && (323)\\
\text{We have assumed that,} \quad & u \notin Escaping(\tau_e\langle\!\langle \tau_r \rangle\!\rangle) && (324)\\
\text{Therefore,} \quad & u = w && (325)\\
\text{From (304),} \quad & u \in \eta_a(x) \wedge w \in \eta_a(x') && (326)\\
\text{Using (325),} \quad & u \in \eta_a(x) \wedge u \in \eta_a(x') && (327)\\
\text{Hence,} \quad & \eta_a(x) \cap \eta_a(x') && (328)
\end{aligned}$$

By Constraint (226), $\langle x, f, y\rangle \in \mathsf{EE}_e$, $\langle x', f, y'\rangle \in \mathsf{IE}_e$ and (328) imply that,

$$\eta_a(y') \subseteq \eta_a(y) \tag{329}$$

By the inductive hypothesis,

$$s \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y') \implies \exists m \in \eta_a(y') \text{ s.t. } h'(s) \in \eta'(m) \tag{330}$$

From (329) and (330),

$$s \in \mu^{i-1}[\![\tau_e, ptg_1]\!](y') \implies \exists m \in \eta_a(y) \text{ s.t. } h'(s) \in \eta'(m) \tag{331}$$

Hence, the claim holds in the i^{th} iteration implying that it will hold in all iterations. Hence the lemma.

Lemma 25. *Let S be a procedure call Call $P(v_1, \cdots, v_k)$.*

$$f_r \sim \tau_r \wedge f_e \sim \tau_e \implies f_r \circ CallReturn_S(f_e) \sim \tau_e\langle\!\langle \tau_r \rangle\!\rangle_a^S \tag{332}$$

Proof. As explained earlier, to prove (332) it suffices to prove the following implication.

$$\forall g_c, ptg_1 \in \mathbb{G}_c, \quad ptg_1 \preceq \tau_r\langle g_c\rangle \implies \tau_e\langle ptg_1\rangle) \preceq (\tau_e\langle\!\langle \tau_r \rangle\!\rangle)\langle g_c\rangle \tag{333}$$

As before, let $h' : \mathsf{V}_e^1 \mapsto {\mathsf{V}_{\mathsf{absx}}}^c$ denote

$$\forall x \in \mathsf{V}_e^c. \quad h'(x) = \begin{cases} h(x), & x \in \mathsf{V}_{c_1} \\ x, & x \in \mathsf{V}_e \end{cases}$$

As mentioned earlier we use the following naming conventions:

$$\begin{aligned}
\tau_e\langle\langle\tau_r\rangle\rangle &= (\mathsf{EV}_{\mathsf{absx}}, \mathsf{EE}_{\mathsf{absx}}, \pi_{absx}, \mathsf{IV}_{\mathsf{absx}}, \mathsf{IE}_{\mathsf{absx}}, \sigma_{absx}) \\
g_c &= (\mathsf{V}_c, \mathsf{E}_c, \sigma_c) \\
ptg_1 &= (\mathsf{V}_{c_1}, \mathsf{E}_{c_1}, \sigma_{c_1}) \\
\tau_r\langle g_c\rangle &= (\mathsf{V}_r^c, \mathsf{E}_r^c, \sigma_r^c) \\
\tau_e\langle ptg_1\rangle &= (\mathsf{V}_e^1, \mathsf{E}_e^1, \sigma_e^1) \\
(\tau_e\langle\langle\tau_r\rangle\rangle)\langle g_c\rangle &= (\mathsf{V}_{\mathsf{absx}}{}^c, \mathsf{E}_{\mathsf{absx}}{}^c, \sigma_{absx}^c)
\end{aligned}$$

We now prove that, if $ptg_1 \preceq_h \tau_r\langle g_c\rangle$, h', as defined above, induces an embedding from $\tau_e\langle ptg_1\rangle$ to $(\tau_e\langle\langle\tau_r\rangle\rangle\langle g_c\rangle$ by proving that the following conditions hold.

$$\langle x, f, y\rangle \in \mathsf{E}_e^1 \implies \langle h'(x), f, h'(y)\rangle \in \mathsf{E}_{\mathsf{absx}}{}^c \tag{334}$$

$$\forall var \in Vars.\ \sigma_{absx}^c(var) \supseteq \hat{h'}(\sigma_e^1(var)) \tag{335}$$

Proof for (334), Say $\langle x, f, y\rangle \in \mathsf{E}_e^1$.

By the construction of $\langle\rangle$, $\langle x, f, y\rangle \in \mathsf{E}_e^1$ only if at least one of the following conditions hold:

$$(a) \quad \langle x, f, y\rangle \in \mathsf{E}_{c_1} \tag{336}$$

$$(b) \quad \exists u, w \in (\mathsf{IV}_e \cup \mathsf{EV}_e)\ s.t.\ x \in \eta[\![\tau_e, ptg_1]\!](u),\ y \in \eta[\![\tau_e, ptg_1]\!](w),\ \langle u, f, w\rangle \in \mathsf{IE}_e \tag{337}$$

Consider the case where $\langle x, f, y\rangle \in \mathsf{E}_{c_1}$,

$$\text{Since } ptg_1 \preceq_h \tau_r\langle g_c\rangle, \quad \langle h(x), f, h(y)\rangle \in \mathsf{E}_r^c \tag{338}$$

$$\text{By Lemma 23,} \quad \tau_r \preceq_{id}^{Vars} \tau_e\langle\langle\tau_r\rangle\rangle \tag{339}$$

$$\text{From Corollary 2,} \quad \tau_r\langle g_c\rangle \preceq_{id}^{Vars} (\tau_e\langle\langle\tau_r\rangle\rangle)\langle g_c\rangle \tag{340}$$

$$\text{Therefore by (338),} \quad \langle h(x), f, h(y)\rangle \in \mathsf{E}_{\mathsf{absx}}{}^c \tag{341}$$

$$\text{Given that,} \quad \langle x, f, y\rangle \in \mathsf{E}_{c_1} \tag{342}$$

$$\text{Which implies that,} \quad x, y \in \mathsf{V}_{c_1} \tag{343}$$

$$\text{By the def. of } h', \quad h'(x) = h(x) \text{ and } h'(y) = h(y) \tag{344}$$

$$\text{Sub. in (341),} \quad \langle h'(x), f, h'(y)\rangle \in \mathsf{E}_{\mathsf{absx}}{}^c \tag{345}$$

Now consider the case where,

$$\exists u, w \in (\mathsf{IV}_e \cup \mathsf{EV}_e)\ s.t.\ x \in \eta[\![\tau_e, ptg_1]\!](u), \quad y \in \eta[\![\tau_e, ptg_1]\!](w), \quad \langle u, f, w\rangle \in \mathsf{IE}_e$$

$$\text{By the def. of } \tau_e\langle\langle\tau_r\rangle\rangle, \quad \langle u,f,w\rangle \in \mathsf{IE}_e \implies$$
$$\eta_a(u) \times f \times \eta_a(w) \subseteq \mathsf{IE}_{\mathsf{absx}} \tag{346}$$
$$\text{By Lemma 24,} \quad ptg_1 \preceq_h \tau_r\langle g_c\rangle \wedge x \in \eta[\![\tau_e, ptg_1]\!](u)$$
$$\implies \exists n \in \eta_a(u) \text{ s.t. } h'(x) \in \eta[\![\tau_e\langle\langle\tau_r\rangle\rangle, g_c]\!](n) \tag{347}$$
$$\text{Similarly,} \quad \exists m \in \eta_a(w) \text{ s.t. } h'(y) \in \eta[\![\tau_e\langle\langle\tau_r\rangle\rangle, g_c]\!](m) \tag{348}$$
$$\text{By (346),} \quad n \in \eta_a(u) \wedge m \in \eta_a(w) \implies \langle n,f,m\rangle \in \mathsf{IE}_{\mathsf{absx}} \tag{349}$$

By the definition of $\langle\rangle$, (347), (348) and (349) imply that,

$$\langle h'(x), f, h'(y)\rangle \in {\mathsf{E}_{\mathsf{absx}}}^c$$

Proof for (335), $\forall var \in Vars.\ \sigma^c_{absx}(var) \supseteq \hat{h'}(\sigma^c_e(var))$
Say $x = \sigma^c_e(var)$,

$$\text{By the def. of } \langle\rangle, \quad \exists w \in \sigma_e(v) \wedge x \in \eta[\![\tau_e\langle ptg_1\rangle]\!](w) \tag{350}$$
$$\text{By Lemma 24,} \quad ptg_1 \preceq_h \tau_r\langle g_c\rangle \wedge x \in \eta[\![\tau_e, ptg_1]\!](w)$$
$$\implies \exists n \in \eta_a(w) \text{ s.t. } h'(x) \in \eta[\![\tau_e\langle\langle\tau_r\rangle\rangle, g_c]\!](n) \tag{351}$$
$$\text{From (350) and (351),} \quad n \in \eta_a(w) \wedge w \in \sigma_e(v) \tag{352}$$
$$\text{From the def. of } \tau_e\langle\langle\tau_r\rangle\rangle^S_a, \quad n \in \sigma_{absx}(v) \tag{353}$$
$$\text{From (351) and (353),} \quad n \in \sigma_{absx}(v) \wedge h'(x) \in \eta[\![\tau_e\langle\langle\tau_r\rangle\rangle, g_c]\!](n) \tag{354}$$
$$\text{From the def. of } \langle\rangle, \quad h'(x) \in \sigma^c_{absx}(v) \tag{355}$$

9 Proofs For The Extended Abstract Interpretation

As mentioned in Section 5, in order to support the node merging optimization we enhance the abstract domain to include an equivalence relation on nodes. We define the enhanced abstract domain $\mathcal{F}_m$ as the set of pairs $(\tau,\xi) \in \mathcal{F}_a \times \mathcal{P}$ that satisfy the following conditions:

$$\forall x \in N_a, \quad x \in (\mathsf{EV} \cup \mathsf{IV}) \implies \xi(x) = x \tag{356}$$

where $\mathcal{P} \subseteq N_a \mapsto N_a$ is a set of functions representing the set of all equivalence relations on N_a. Given a equivalence relation $\equiv$ on N_a the function $\xi \in \mathcal{P}$ that represents the equivalence relation is given by:

$$\forall x \in N_a, \quad \xi(x) = \max_v(\{y \mid y \equiv x\}) \tag{357}$$

where $\max_v(N)$ denotes the maximum element in the set $N \subseteq N_a$ w.r.t to a total order $(\leq_v)$ on N_a. A function $\xi \in \mathcal{P}$ satisfies the following properties (which are a direct consequence of (357)):

Property 4. $\forall \xi \in \mathcal{P}, \forall x \in N_a, \quad x \leq_v \xi(x)$

Property 5. $\forall \xi \in \mathcal{P}, \forall x \in N_a, \quad \xi(x) = \xi(\xi(x))$

Given a function $\xi \in \mathcal{P}$, the equivalence relation that it represents (denoted as $\equiv_\xi$) is given by:

$$\forall x, y \in N_a, \quad x \equiv_\xi y \iff \xi(x) = \xi(y) \tag{358}$$

9.1 The Partial Ordering On Equivalence Relations (Lemma 4)

Define a binary relation $\leq_p$ on $\mathcal{P}$ as follows: $\xi_1 \leq_p \xi_2$ iff the equivalence relation $\equiv_{\xi_1}$ is a refinement of (or subset of) the equivalence relation $\equiv_{\xi_2}$. The following lemma gives an equivalent definition.

Lemma 26.

$$\forall \xi_1, \xi_2 \in \mathcal{P}, \quad \xi_1 \leq_p \xi_2 \iff \xi_1 \circ \xi_2 = \xi_2 \tag{359}$$

Given two functions $\xi_1, \xi_2 \in \mathcal{P}$ we define a binary operator, denoted by $\xi_1 \sqcup_p \xi_2$, as the function corresponding to the equivalence relation that is a superset of $\equiv_{\xi_1}$ and $\equiv_{\xi_2}$. The following is an equivalent definition of $\sqcup_p$.

Lemma 27.

$$\forall \xi_1, \xi_2 \in \mathcal{P}, \quad \xi_1 \sqcup_p \xi_2 = \lambda x. \max_v(f'(x)) \tag{360}$$

where $f' : N_a \mapsto 2^{N_a}$ *is defined as follows:*

$$f' = lfp\ f \quad where, \tag{361}$$

$$f(x) = \{x\} \cup \{z \in N_a \mid \exists y \in f(x)\ s.t. \\ (\xi_1(z) = \xi_1(y) \vee \xi_2(z) = \xi_2(y))\} \tag{362}$$

Lemma 28. *If* $\xi_1 \leq_p \xi_2$ *then* $\xi_1 \circ \xi_2 = \xi_2 \circ \xi_1 = \xi_2$.

Proof. Say $x \in N_a$.

$$\text{Since } \xi_1 \leq_p \xi_2, \quad \xi_1 \circ \xi_2 = \xi_2 \tag{363}$$

$$\text{Let} \quad \xi_2(\xi_1(x)) = y_1 = \xi_2(x) \tag{364}$$

$$\text{Let} \quad \xi_1(\xi_2(x)) = y_2 \tag{365}$$

$$\text{Since } \xi_2(x) = y_1, \quad \xi_1(y_1) = y_2 \tag{366}$$

$$\text{By the def. of } \mathcal{P}, \quad y_1 \leq_v y_2 \tag{367}$$

$$\text{By (365)}, \quad \xi_2(\xi_1(\xi_2(x))) = \xi_2(y_2) \tag{368}$$

$$\text{Which reduces to}, \quad \xi_2(x) = \xi_2(y_2) \tag{369}$$

$$\text{By (364)}, \quad y_1 = \xi_2(y_2) \tag{370}$$

$$\text{Hence}, \quad y_2 \leq_v y_1 \tag{371}$$

$$\text{By (367) and (371)}, \quad y_1 = y_2 \tag{372}$$

Lemma 29. *(a) $\leq_p$ is a partial order.*
(b) $\sqcup_p$ is the join operator i.e.the least upper bound operator w.r.t $\leq_p$
(c) The least element in $\mathcal{P}$ w.r.t $\leq_p$ is $\bot_{\mathcal{P}} = \lambda x.x$.

Proof. ***(a) $\leq_p$ is a partial order.***

(i) $\leq_p$ is Reflexive. Let $\xi_1 \in \mathcal{P}$. Since ξ_1 is idempotent $\xi_1 \circ \xi_1 = \xi_1$. Hence, $\xi_1 \leq_p \xi_1$.

(ii) $\leq_p$ is anti-symmetric. Let $\xi_1, \xi_2 \in \mathcal{P}$ such that $\xi_1 \leq_p \xi_2$ and $\xi_2 \leq_p \xi_1$.

$$\text{Since } \xi_1 \leq_p \xi_2, \quad \xi_1 \circ \xi_2 = \xi_2 \tag{373}$$
$$\text{By Lemma 28}, \quad \xi_2 \circ \xi_1 = \xi_2 \tag{374}$$
$$\text{Since } \xi_2 \leq_p \xi_1, \quad \xi_2 \circ \xi_1 = \xi_1 \tag{375}$$
$$\text{By (374) and (375)} \quad \xi_1 = \xi_2 \tag{376}$$

(iii) $\leq_p$ is transitive. Let $\xi_1, \xi_2, \xi_3 \in \mathcal{P}$ such that $\xi_1 \leq_p \xi_2$ and $\xi_2 \leq_p \xi_3$.

$$\text{Since } \xi_1 \leq_p \xi_2, \quad \xi_1 \circ \xi_2 = \xi_2 \tag{377}$$
$$\text{Since } \xi_2 \leq_p \xi_3, \quad \xi_2 \circ \xi_3 = \xi_3 \tag{378}$$
$$\text{Sub. (377) in (378)}, \quad (\xi_1 \circ \xi_2) \circ \xi_3 = \xi_3 \tag{379}$$
$$\text{Since } \circ \text{ is associative}, \quad \xi_1 \circ (\xi_2 \circ \xi_3) = \xi_3 \tag{380}$$
$$\text{By (378)}, \quad \xi_1 \circ \xi_3 = \xi_3 \tag{381}$$

(b) $\sqcup_p$ is the least upper bound operator. We need to show that the following two properties hold: $\forall \xi_1, \xi_2, \xi' \in \mathcal{P}$,

$$\xi_1 \leq_p (\xi_1 \sqcup_p \xi_2) \text{ and } \xi_2 \leq_p (\xi_1 \sqcup_p \xi_2) \tag{382}$$
$$\xi_1 \leq_p \xi' \wedge \xi_2 \leq_p \xi' \implies (\xi_1 \sqcup_p \xi_2) \leq_p \xi' \tag{383}$$

We assume that the fix point of the function f is computed iteratively starting from $f = \emptyset$. The value of f at the end of iteration i is denoted using f^i ($f^0 = \emptyset$).

Proof of (382). Let $\xi_j = \xi_1 \sqcup_p \xi_2$. We first prove that $\xi_1 \leq_p \xi_j$ i.e, $\xi_1 \circ \xi_j = \xi_j$. From the def. of $\sqcup_p$, it suffices to prove that $f'(x) = f'(\xi_1(x))$. We prove this by inducting over the iterations in the computation of $f(x)$.

Claim: $\forall i \in \mathbb{N}, \quad f^i(x) \subseteq f'(\xi_1(x))$.

Base case (iteration 1): $f^1(x) = \{x\}$. We now show that $x \in f'(\xi_1(x))$.

$$\text{By the def. of } f', \quad \xi_1(x) \in f'(\xi_1(x)) \tag{384}$$
$$\text{Since } \xi_1 \text{ is idempotent}, \quad \xi_1(\xi_1(x)) = \xi_1(x) \tag{385}$$
$$\text{Hence}, \quad \exists y \in f'(\xi_1(x)) \text{ s.t. } \xi_1(x) = \xi_1(y) \quad (y = \xi_1(x)) \tag{386}$$
$$\text{Which implies that}, \quad x \in f'(\xi_1(x)) \tag{387}$$

induction step (iteration i+1): Say $z \in f^{i+1}(x)$

$$\text{By the def. of } f, \quad \exists y \in f^i(x) \text{ s.t. } (\xi_1(z) = \xi_1(y) \vee \xi_2(z) = \xi_2(y)) \tag{388}$$

$$\text{By inductive hypothesis,} \quad f^i(x) \subseteq f'(\xi_1(x)) \tag{389}$$

$$\text{Hence,} \quad \exists y \in f'(\xi_1(x)) \text{ s.t. } (\xi_1(z) = \xi_1(y) \vee \xi_2(z) = \xi_2(y)) \tag{390}$$

$$\text{Which implies that,} \quad z \in f'(\xi_1(x)) \tag{391}$$

Similarly, it can be proved that $f'(\xi_1(x)) \subseteq f'(x)$. Hence, $f'(x) = f'(\xi_1(x))$. Therefore $\xi_1 \leq_p \xi_j$. By a similar argument $\xi_2 \leq_p \xi_j$.

Proof of (383). We first prove that, if $\xi_1 \leq_p \xi'$ and $\xi_2 \leq_p \xi'$ then,

$$\forall x \in N_a, \quad y \in f'(x) \implies \xi'(x) = \xi'(y)$$

We prove this using induction over the iterations in the computation of f'.

Claim: if $\xi_1 \leq_p \xi'$ and $\xi_2 \leq_p \xi'$ then $\forall i \in \mathbb{N}, \quad y \in f^i(x) \implies \xi'(x) = \xi'(y)$.

Base case (iteration 1): $f^1(x) = \{x\}$. We know that $\xi'(x) = \xi'(x)$
induction step (iteration i+1): Say $z \in f^{i+1}(x)$

$$\text{By the def. of } f, \quad \exists y \in f^i(x) \text{ s.t. } (\xi_1(z) = \xi_1(y) \vee \xi_2(z) = \xi_2(y)) \tag{392}$$

$$\text{By inductive hypothesis,} \quad \xi'(y) = \xi'(x) \tag{393}$$

$$\text{By (392),} \quad \xi'(y) = \xi'(x) \wedge (\xi_1(z) = \xi_1(y) \vee \xi_2(z) = \xi_2(y)) \tag{394}$$

$$\text{Since } \xi_1 \leq_p \xi' \text{ and } \xi_2 \leq_p \xi', \quad \xi_1(z) = \xi_1(y) \vee \xi_2(z) = \xi_2(y) \implies \xi'(z) = \xi'(y) \tag{395}$$

$$\text{Sub. in (394),} \quad \xi'(y) = \xi'(x) \wedge \xi'(z) = \xi'(y) \tag{396}$$

$$\text{Which implies that,} \quad \xi'(x) = \xi'(z) \tag{397}$$

Now we prove that $\xi_j \leq_p \xi'$ i.e, $\xi_j \circ \xi' = \xi'$.

$$\text{By the def. of } \xi_1 \sqcup_p \xi_2, \quad \xi_j(x) = \max_v(f'(x)) \in f'(x) \tag{398}$$

$$\text{Proved before,} \quad \forall x \in N_a, \quad y \in f'(x) \implies \xi'(x) = \xi'(y) \tag{399}$$

$$\text{Hence,} \quad \xi'(\xi_j(x)) = \xi'(x) \tag{400}$$

9.2 The Extended Abstract Domain

Partial Order. Define a binary relation $\leq_m$ on $\mathcal{F}_m$ as follows: $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2)$ iff

$$\xi_1 \leq_p \xi_2 \tag{401}$$

$$\tau_1 \preceq_{\xi_2} \tau_2 \tag{402}$$

Lemma 30. *$\leq_m$ induces a partial order on $\mathcal{F}_m$ i.e, $\leq_m$ is reflexive, anti-symmetric and transitive.*

Proof. (i) $\leq_m$ is reflexive.

Say $(\tau_1, \xi_1) \in \mathcal{F}_m$. Since $\leq_p$ is a partial order $\xi_1 \leq_p \xi_1$. By the def. of $\mathcal{F}_m$, since $(\tau_1, \xi_1) \in \mathcal{F}_m$,

$$x \in (\mathsf{EV}_1 \cup \mathsf{IV}_1) \implies \xi_1(x) = x \tag{403}$$

$$\text{Which implies that,} \quad \tau_1 \preceq_{\xi_1} \tau_1 \tag{404}$$

Hence, $(\tau_1, \xi_1) \leq_m (\tau_1, \xi_1)$.

(ii) $\leq_m$ is anti-symmetric.

Say $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2)$ and $(\tau_2, \xi_2) \leq_m (\tau_1, \xi_1)$. We first show that $\xi_1 = \xi_2$.

$$\text{Since } (\tau_1, \xi_1) \leq_m (\tau_2, \xi_2), \quad \xi_1 \leq_p \xi_2 \tag{405}$$

$$\text{Since } (\tau_2, \xi_2) \leq_m (\tau_1, \xi_1), \quad \xi_2 \leq_p \xi_1 \tag{406}$$

$$\text{Since } \leq_p \text{ is a partial order,} \quad \xi_1 = \xi_2 \tag{407}$$

We now show that $\tau_1 = \tau_2$.

$$\text{By property (356),} \quad x \in (\mathsf{EV}_1 \cup \mathsf{IV}_1) \implies \xi_1(x) = x \tag{408}$$

$$\text{By (407),} \quad x \in (\mathsf{EV}_1 \cup \mathsf{IV}_1) \implies \xi_2(x) = x \tag{409}$$

$$\text{By def. of } \leq_m, \quad \tau_1 \preceq_{\xi_2} \tau_2 \tag{410}$$

$$\text{By (409),} \quad \tau_1 \sqsubseteq_{co} \tau_2 \tag{411}$$

$$\text{Similarly, } \tau_2 \preceq_{\xi_1} \tau_1 \text{ implies,} \quad \tau_2 \sqsubseteq_{co} \tau_1 \tag{412}$$

$$\text{By (411) and (412),} \quad \tau_1 = \tau_2 \tag{413}$$

Hence, $(\tau_1, \xi_1) = (\tau_2, \xi_2)$.

(iii) $\leq_m$ is transitive.

Say $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2)$ and $(\tau_2, \xi_2) \leq_m (\tau_3, \xi_3)$. We now show that $(\tau_1, \xi_1) \leq_m (\tau_3, \xi_3)$ by proving that the conditions (401) and (402) holds. Condition (401) holds as $\leq_p$ is a partial oder and hence transitive i.e, $\xi_1 \leq_p \xi_2$ and $\xi_2 \leq_p \xi_3$ imply that $\xi_1 \leq_p \xi_3$. We now show that Condition (402) also holds.

$$\text{Since } (\tau_1, \xi_1) \leq_m (\tau_2, \xi_2), \quad \tau_1 \preceq_{\xi_2} \tau_2 \tag{414}$$

$$\text{Since } (\tau_2, \xi_2) \leq_m (\tau_3, \xi_3), \quad \tau_2 \preceq_{\xi_3} \tau_3 \tag{415}$$

$$\text{By the prop. of } \preceq, \quad \tau_1 \preceq_{\xi_2 \circ \xi_3} \tau_3 \tag{416}$$

$$\text{Since } \xi_2 \leq_p \xi_3, \quad \xi_2 \circ \xi_3 = \xi_3 \tag{417}$$

$$\text{Sub. in (416),} \quad \tau_1 \preceq_{\xi_3} \tau_3 \tag{418}$$

Join operator. Define a function *apply* : $\mathcal{P} \times \mathcal{F}_a \mapsto \mathcal{F}_a$ as follows: Let $\tau = (\mathsf{EV}, \mathsf{EE}, \pi, \mathsf{IV}, \mathsf{IE}, \sigma)$ and $\xi \in \mathcal{P}$.

$apply(\xi, \tau) = (\mathsf{EV}_m, \mathsf{EE}_m, \pi_m, \mathsf{IV}_m, \mathsf{IE}_m, \sigma_m)$ where,

$$\begin{aligned}
\mathsf{IV}_m &= \{\xi(x) \mid x \in \mathsf{IV}\} \\
\mathsf{EV}_m &= \{\xi(x) \mid x \in \mathsf{EV}\} \\
\mathsf{EE}_m &= \{\langle \xi(x), f, \xi(y)\rangle \mid \langle x, f, y\rangle \in \mathsf{EE}\} \\
\mathsf{IE}_m &= \{\langle \xi(x), f, \xi(y)\rangle \mid \langle x, f, y\rangle \in \mathsf{IE}\} \\
\pi_m(var) &= \{\xi(x) \mid x \in \pi(var)\} \\
\sigma_m(var) &= \{\xi(x) \mid x \in \sigma(var)\}
\end{aligned}$$

From the definition of *apply*, it is clear that, $\tau \preceq_\xi apply(\xi, \tau)$.

Define a binary operator $\sqcup_m$ on $\mathcal{F}_m$ as follows: for all $(\tau_1, \xi_1), (\tau_2, \xi_2) \in \mathcal{F}_m$

$$(\tau_1, \xi_1) \sqcup_m (\tau_2, \xi_2) = (apply(\xi_1 \sqcup_p \xi_2, \tau_1 \sqcup_{co} \tau_2), \xi_1 \sqcup_p \xi_2)$$

Lemma 31. *$(\mathcal{F}_m, \leq_m, \sqcup_m)$ is a join semi-lattice*

Proof. For all $(\tau_1, \xi_1), (\tau_2, \xi_2), (\tau', \xi') \in \mathcal{F}_m$, we need to show that the following two properties hold:

$$(\tau_1, \xi_1) \leq_m (\tau_1, \xi_1) \sqcup_m (\tau_2, \xi_2) \tag{419}$$
$$(\tau_2, \xi_2) \leq_m (\tau_1, \xi_1) \sqcup_m (\tau_2, \xi_2) \tag{420}$$
$$(\tau_1, \xi_1) \leq_m (\tau', \xi') \wedge (\tau_2, \xi_2) \leq_m (\tau', \xi') \implies (\tau_1, \xi_1) \sqcup_m (\tau_2, \xi_2) \leq_m (\tau', \xi') \tag{421}$$

Let $\xi_j = \xi_1 \sqcup_p \xi_2$. *Proof of (419).*
(i) By the property of join, $\xi_1 \leq_p \xi_j$ and $\xi_2 \leq_p \xi_j$.
(ii) Proof of $\tau_1 \preceq_{\xi_j} apply(\xi_j, \tau_1 \sqcup_{co} \tau_2)$.

$$\text{By the def. of } \sqcup_{co}, \quad \tau_1 \preceq_{id} (\tau_1 \sqcup_{co} \tau_2) \tag{422}$$
$$\text{By the def. of } apply, \quad (\tau_1 \sqcup_{co} \tau_2) \preceq_{\xi_j} apply(\xi_j, \tau_1 \sqcup_{co} \tau_2) \tag{423}$$
$$\text{Hence,} \quad \tau_1 \preceq_{\xi_j} apply(\xi_j, \tau_1 \sqcup_{co} \tau_2) \tag{424}$$

Similarly, $\tau_2 \preceq_{\xi_j} apply(\xi_j, \tau_1 \sqcup_{co} \tau_2)$.

Proof of (421). Let

$$\begin{aligned}
\tau_1 &= (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1) \\
\tau_2 &= (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2) \\
\tau_1 \sqcup_{co} \tau_2 = \tau_{co} &= (\mathsf{EV}_{co}, \mathsf{EE}_{co}, \pi_{co}, \mathsf{IV}_{co}, \mathsf{IE}_{co}, \sigma_{co}) \\
apply(\xi_j, \tau_1 \sqcup_{co} \tau_2) = \tau_j &= (\mathsf{EV}_j, \mathsf{EE}_j, \pi_j, \mathsf{IV}_j, \mathsf{IE}_j, \sigma_j)
\end{aligned}$$

By the property of join, $\xi_1 \leq_p \xi'$, $\xi_2 \leq_p \xi'$ imply that $\xi_j \leq_p \xi'$. Hence, it suffices to prove that, $\tau_j \preceq_{\xi'} \tau'$.

Say $\langle u, f, w\rangle \in (\mathsf{EE}_j \cup \mathsf{IE}_j)$. We now show that $\langle \xi'(u), f, \xi'(w)\rangle \in (\mathsf{EE}' \cup \mathsf{IE}')$.

$$\text{By the def. of } \textit{apply}, \quad \exists \langle x, f, y\rangle \in (\mathsf{EE}_{co} \cup \mathsf{IE}_{co}) \text{ s.t. } u = \xi_j(x) \wedge w = \xi_j(y) \tag{425}$$

$$\text{By the def. of } \sqcup_{co}, \quad (\mathsf{EE}_{co} \cup \mathsf{IE}_{co}) = (\mathsf{EV}_1 \cup \mathsf{IV}_1) \cup (\mathsf{EV}_2 \cup \mathsf{IV}_2) \tag{426}$$

Hence, $\langle x, f, y\rangle \in (\mathsf{EV}_1 \cup \mathsf{IV}_1)$ or $\langle x, f, y\rangle \in (\mathsf{EV}_2 \cup \mathsf{IV}_2)$. Since $(\tau_1, \xi_1) \leq_m (\tau', \xi')$ and $(\tau_2, \xi_2) \leq_m (\tau', \xi')$, $\tau_1 \preceq_{\xi'} \tau'$ and $\tau_2 \preceq_{\xi'} \tau'$. Therefore,

$$\langle \xi'(x), f, \xi'(y)\rangle \in (\mathsf{EV}' \cup \mathsf{IV}') \tag{427}$$

$$\text{Since } \xi_j \leq_p \xi', \quad \xi_j \circ \xi' = \xi' \tag{428}$$

$$\text{Sub in (427)}, \quad \langle \xi'(\xi_j(x)), f, \xi'(\xi_j(y))\rangle \in (\mathsf{EV}' \cup \mathsf{IV}') \tag{429}$$

$$\text{By (425)}, \quad \langle \xi'(u), f, \xi'(w)\rangle \in (\mathsf{EE}' \cup \mathsf{IE}') \tag{430}$$

Similarly, it can be shown that $\forall var \in \mathit{Vars}.\ \hat{\xi}'(\sigma_j(var)) \subseteq \sigma'(var)$.

9.3 Monotonicity Of γ_M (Lemma 5)

Let $(\tau_1, \xi_1) \in \mathcal{F}_m$, we define the concretization function γ_M as $\gamma_M((\tau_1, \xi_1)) = \gamma_T(\tau_1)$.

Lemma 32. $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2) \implies \gamma_M((\tau_1, \xi_1)) \sqsubseteq_c \gamma_M((\tau_2, \xi_2))$

Proof.

$$\text{By the def. of } \gamma_M, \quad \gamma_M((\tau_1, \xi_1)) = \gamma_T(\tau_1) \tag{431}$$

$$\gamma_M((\tau_2, \xi_2)) = \gamma_T(\tau_2) \tag{432}$$

$$\text{Since } (\tau_1, \xi_1) \leq_m (\tau_2, \xi_2), \quad \tau_1 \preceq_{\xi_2} \tau_2 \tag{433}$$

$$\text{By Lemma 3}, \quad \gamma_T(\tau_1) \sqsubseteq_c \gamma_T(\tau_2) \tag{434}$$

9.4 Correctness of Abstract Transformers (Lemma 6)

Abstract transfer functions.

As mentioned in Section 5, the abstract semantics ($[\![S]\!]_m$) for every statement other than a method call statement is given by:

$$[\![S]\!]_m((\tau, \xi)) = (\tau', \xi) \text{ where,} \tag{435}$$

$$\tau' = \mathit{apply}(\xi, [\![S]\!]_a(\tau)) \tag{436}$$

The abstract semantics of a procedure call statement S is given by,

$$(\tau_e, \xi_e)\langle\!\langle (\tau_r, \xi_r)\rangle\!\rangle_m^S = (\tau', \xi_r \sqcup_p \xi_e) \text{ where,} \tag{437}$$

$$\tau' = \mathit{apply}(\xi_r \sqcup_p \xi_e, \mathit{pop}_S(\tau_r, \tau_e\langle\!\langle \mathit{push}_S(\tau_r)\rangle\!\rangle)) \tag{438}$$

Lemma 33. *Let S be any statement other than the method call statement. If $f \in \mathcal{F}_c$ and $(\tau, \xi) \in \mathcal{F}_m$ then*

$$f \sim (\tau, \xi) \implies f \circ [\![S]\!]_c \sim [\![S]\!]_m((\tau, \xi))$$

Proof. Let $[\![S]\!]_m((\tau, \xi)) = (\tau', \xi)$ where, $\tau' = \mathit{apply}(\xi, [\![S]\!]_a(\tau))$. By the def. of *apply*, $[\![S]\!]_a(\tau) \preceq_\xi \tau'$. Hence, the claim follows from Lemma 2 and Lemma 14.

Lemma 34. *Let S be a method call statement. If $f_r, f_e \in \mathcal{F}_c$ and $(\tau_r, \xi_r), (\tau_e, \xi_e) \in \mathcal{F}_m$ then*

$$f_r \sim (\tau_r, \xi_r) \wedge f_e \sim (\tau_e, \xi_e) \implies f_r \circ \mathit{CallReturn}_S(f_e) \sim (\tau_e, \xi_e)\langle\!\langle(\tau_r, \xi_r)\rangle\!\rangle_m^S$$

Proof. Let $(\tau_e, \xi_e)\langle\!\langle(\tau_r, \xi_r)\rangle\!\rangle_m^S = (\tau', \xi_r \sqcup_p \xi_e)$ where, $\tau' = \mathit{apply}(\xi_r \sqcup_p \xi_e, \mathit{pop}_S(\tau_r, \tau_e\langle\!\langle \mathit{push}_S(\tau_r)\rangle\!\rangle))$. By the def. of *apply*, $\mathit{pop}_S(\tau_r, \tau_e\langle\!\langle \mathit{push}_S(\tau_r)\rangle\!\rangle) \preceq_{\xi_r \sqcup_p \xi_e} \tau'$. Hence, the claim follows from Lemma 2 and Lemma 14.

9.5 Monotonicity of Abstract Transformers (Lemma 7)

In the rest the section we use the following naming convention.

$$\begin{aligned}
\tau_1 &= (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1) \\
\tau_2 &= (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2) \\
[\![S]\!]_a(\tau_1) &= \tau_{1_a} = (\mathsf{EV}_{1_a}, \mathsf{EE}_{1_a}, \pi_{1_a}, \mathsf{IV}_{1_a}, \mathsf{IE}_{1_a}, \sigma_{1_a}) \\
[\![S]\!]_a(\tau_2) &= \tau_{2_a} = (\mathsf{EV}_{2_a}, \mathsf{EE}_{2_a}, \pi_{2_a}, \mathsf{IV}_{2_a}, \mathsf{IE}_{2_a}, \sigma_{2_a}) \\
\tau_{1_m} &= (\mathsf{EV}_{1_m}, \mathsf{EE}_{1_m}, \pi_{1_m}, \mathsf{IV}_{1_m}, \mathsf{IE}_{1_m}, \sigma_{1_m}) \\
\tau_{2_m} &= (\mathsf{EV}_{2_m}, \mathsf{EE}_{2_m}, \pi_{2_m}, \mathsf{IV}_{2_m}, \mathsf{IE}_{2_m}, \sigma_{2_m}) \\
[\![S]\!]_m((\tau_1, \xi_1)) &= (\tau_{1_m}, \xi_{1_m}) \\
[\![S]\!]_m((\tau_2, \xi_2)) &= (\tau_{2_m}, \xi_{2_m})
\end{aligned}$$

Consider a statement S other than a procedure call statement. Let $(\tau_1, \xi_1), (\tau_2, \xi_2) \in \mathcal{F}_m$ and $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2)$. From the abstract semantics of S, $\xi_{1_m} = \xi_1$, $\xi_{2_m} = \xi_2$. Hence, $\xi_1 \leq_p \xi_2$ imply that $\xi_{1_m} \leq_p \xi_{2_m}$. Therefore, to prove monotonicity of $[\![S]\!]_m$ (w.r.t $\leq_m$) it suffices to prove the following:

$$(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2) \implies \tau_{1_m} \preceq_{\xi_2} \tau_{2_m} \tag{439}$$

We now state and prove a lemma that applies to all statements in our language other than the procedure call statement.

Lemma 35. *Let S be any statement other than a procedure call statement. Let $\tau_1 = (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1)$ and $\tau_2 = (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2)$. If $[\![S]\!]_a(\tau_1)$ and $[\![S]\!]_a(\tau_2)$ are of the form,*

$$\begin{aligned}
[\![S]\!]_a(\tau_1) = (&\mathsf{EV}_1 \cup \mathsf{EV}_{new_1}, \mathsf{EE}_1 \cup \mathsf{EE}_{new_1}, \pi, \mathsf{IV}_1 \cup \mathsf{IV}_{new_1}, \mathsf{IE}_1 \cup \mathsf{IE}_{new_1}, \\
&\lambda var.(var \in \mathit{Vars}_{mod} \to \mathit{newmap}_1(var) \mid \sigma_1(var)) \\
[\![S]\!]_a(\tau_2) = (&\mathsf{EV}_2 \cup \mathsf{EV}_{new_2}, \mathsf{EE}_2 \cup \mathsf{EE}_{new_2}, \pi, \mathsf{IV}_2 \cup \mathsf{IV}_{new_2}, \mathsf{IE}_2 \cup \mathsf{IE}_{new_2}, \\
&\lambda var.(var \in \mathit{Vars}_{mod} \to \mathit{newmap}_2(var) \mid \sigma_2(var))
\end{aligned}$$

and if $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2)$ *then*

$$(i)\ \langle x, f, y\rangle \in (\mathsf{EE}_{new_1} \cup \mathsf{IE}_{new_1}) \implies \exists \langle x', f, y'\rangle \in (\mathsf{EE}_{new_2} \cup \mathsf{IE}_{new_2})\ s.t.\ \xi_2(x) = \xi_2(x') \wedge \xi_2(y) = \xi_2(y')$$

$$(ii)\ var \in Vars_{mod} \implies \hat{\xi_2}(newmap_1(var)) \subseteq \hat{\xi_2}(newmap_2(var))$$

Proof. The overall structure of the proof is as follows, for every statement S in our language other than the procedure call statement we first express its abstract semantics ($[\![S]\!]_a$) in the form shown in the lemma statement and determine the values of the sets (also referred to as entities) $\mathsf{EV}_{new_i}, \mathsf{EE}_{new_i}, \mathsf{IV}_{new_i}, \mathsf{IE}_{new_i}$ and $newmap_i$ for $i \in \{1, 2\}$. We then prove that the property mentioned in the lemma holds for each of the entities.

(I) $S : v_1 = v_2$**.**

Form the def. of $[\![S]\!]_a$, $\mathsf{EE}_{new_1}, \mathsf{IE}_{new_1}$ are empty,

$$Vars_{mod} = \{v_1\}$$
$$newmap_1(v_1) = \sigma_1(v_2)$$
$$newmap_2(v_1) = \sigma_2(v_2)$$

Part(i) trivially holds as ($\mathsf{EE}_{new_1} \cup \mathsf{IE}_{new_1} = \emptyset$). To prove Part(ii) we need to show that, $x \in newmap_1(v_1) \implies \xi_2(x) \in \hat{\xi_2}(newmap_2(v_1))$.

$$\text{By the def. of } [\![S]\!]_a, \quad x \in newmap_1(v_1) \implies x \in \sigma_1(v_2) \tag{440}$$
$$\text{Since } (\tau_1, \xi_1) \leq_m (\tau_2, \xi_2), \quad x \in \sigma_1(v_2) \implies \xi_2(x) \in \sigma_2(v_2) \tag{441}$$
$$\text{Hence,} \quad \xi_2(x) \in newmap_2(v_1) \tag{442}$$
$$\text{Since } \xi_2 \text{ is idempotent,} \quad \xi_2(x) \in \hat{\xi_2}(newmap_2(v_1)) \tag{443}$$

(II) $S : \ell : v = new\ C$**.**

In this case,

$$Vars_{mod} = \{v\}$$
$$newmap_1(v) = newmap_2(v) = \{n_\ell\}$$
$$\mathsf{IE}_{new_1} = \mathsf{IE}_{new_2} = \{n_\ell\} \times Fields \times \{Null\}$$

Proof of *Part(i)*. Say $\langle n_\ell, f, Null\rangle \in \mathsf{IE}_{new_1}$.

$$\text{Given,} \quad \langle n_\ell, f, Null\rangle \in \mathsf{IE}_{new_2} \tag{444}$$
$$\text{Hence,} \quad \exists x', f, y' \in \mathsf{IE}_{new_2} \text{ s.t. } \xi_2(n_\ell) = \xi_2(x') \wedge \xi_2(Null) = \xi_2(y') \text{ where, } x' = n_\ell \wedge y' = Null \tag{445}$$

Since $newmap_1(v) = newmap_2(v) = \{n_\ell\}$, *Part(ii)* also holds.

(III) $S : v_1.f = v_2$**.**

In this case, $newmap_1$ and $newmap_2$ are empty,

$$\mathsf{IE}_{new_1} = \sigma_1(v_1) \times \{f\} \times \sigma_1(v_2)$$
$$\mathsf{IE}_{new_2} = \sigma_2(v_1) \times \{f\} \times \sigma_2(v_2)$$

Proof of Part(i). Say $\langle x, f, y\rangle \in \mathsf{IE}_{new_1}$.

$$\text{By the def. of } \mathsf{IE}_{new_1}, \quad x \in \sigma_1(v_1) \wedge y \in \sigma_1(v_2) \tag{446}$$
$$\text{Since } (\tau_1, \xi_1) \leq_m (\tau_2, \xi_2), \quad \xi_2(x) \in \sigma_2(v_1) \wedge \xi_2(y) \in \sigma_2(v_2) \tag{447}$$
$$\text{By the def. of } \mathsf{IE}_{new_2}, \quad \langle \xi_2(x), f, \xi_2(y)\rangle \in \mathsf{IE}_{new_2} \tag{448}$$
$$\text{By idempotence of } \xi_2, \quad \exists x', f, y' \in \mathsf{IE}_{new_2} \text{ s.t. } \xi_2(x) = \xi_2(x') \wedge \xi_2(y) = \xi_2(y') \tag{449}$$

Part(ii) trivially holds as $newmap_1 = \emptyset$.

(IV) $S : \ell : v_1 = v_2.f$**.** The abstract semantics of S ($[\![S]\!]_a(\tau_i), i \in \{1, 2\}$) is given by,

$$\begin{array}{l}
\text{let} \quad A_i = \{n \mid \exists n_1 \in \sigma_i(v_2), \langle n_1, f, n\rangle \in \mathsf{IE}_i\} \quad \text{in} \\
\text{let} \quad B_i = \sigma_i(v_2) \cap \mathit{Escaping}(\tau_i) \quad \text{in} \\
\text{if} \quad (B_i = \phi) \\
\text{then } (\mathsf{EV}_i, \mathsf{EE}_i, \pi_i, \mathsf{IV}_a, \mathsf{IE}_i, \sigma_i[v_1 \mapsto A_i]) \\
\text{else } (\mathsf{EV}_i \cup \{n_\ell\}, \mathsf{EE}_i \cup B_i \times \{f\} \times \{n_\ell\}, \pi_i, \mathsf{IV}_i, \mathsf{IE}_i, \sigma_i[v_1 \mapsto A_i \cup \{n_\ell\}])
\end{array}$$

Case(a). Consider the case where $B_1 = \phi$. In this case, EE_{new_1}, IE_{new_1} are empty,

$$\begin{aligned}
\mathit{Vars}_{mod} &= \{v_1\} \\
newmap_1(v_1) &= A_1 \\
newmap_2(v_1) &\supseteq A_2
\end{aligned}$$

Part(i) trivially holds as ($\mathsf{EE}_{new_1} \cup \mathsf{IE}_{new_1} = \emptyset$). Proof of Part(ii). Say $x \in newmap_1(v_1)$,

$$\text{By the def. of } newmap_1, \quad newmap_1(v_1) = A_1 \tag{450}$$
$$\text{By the def. of } A_1, \quad A_1 = \{n \mid \exists n_1 \in \sigma_1(v_2), \langle n_1, f, n\rangle \in \mathsf{IE}_1\} \tag{451}$$
$$\text{Since } x \in newmap_1(v_1), \quad \exists n_1 \in \sigma_1(v_2), \langle n_1, f, x\rangle \in \mathsf{IE}_1 \tag{452}$$
$$\text{Since } (\tau_1, \xi_1) \leq_m (\tau_2, \xi_2), \quad n_1 \in \sigma_1(v_2) \implies \xi_2(n_1) \in \sigma_2(v_2) \tag{453}$$
$$\langle n_1, f, x\rangle \in \mathsf{IE}_1 \implies \langle \xi_2(n_1), f, \xi_2(x)\rangle \in \mathsf{IE}_2 \tag{454}$$
$$\text{By the def. of } A_2, \quad \xi_2(x) \in A_2 \tag{455}$$
$$\text{Given}, \quad A_2 \subseteq newmap_2(v_1) \tag{456}$$
$$\text{Hence}, \quad \xi_2(x) \in newmap_2(v_1) \tag{457}$$
$$\text{Since } \xi_2 \text{ is idempotent}, \quad \xi_2(x) \in \hat{\xi_2}(newmap_2(v_1)) \tag{458}$$

Case(b). Consider the case where $B_1 \neq \phi$.

$$\text{By the def. of } B_1, \quad B_1 = \sigma_1(v_2) \cap \mathit{Escaping}(\tau_1) \tag{459}$$
$$\text{Hence}, \quad \exists x \text{ s.t. } x \in \sigma_1(v_2) \wedge x \in \mathit{Escaping}(\tau_1) \tag{460}$$
$$\text{Since } (\tau_1, \xi_1) \leq_m (\tau_2, \xi_2), \quad \xi_2(x) \in \sigma_2(v_2) \tag{461}$$

If $\tau_1 \preceq_h \tau_2$ then $\hat{h}(\mathit{Escaping}(\tau_1)) \subseteq \mathit{Escaping}(\tau_2)$ (proof not shown for brevity). Hence, $x \in \mathit{Escaping}(\tau_1)$ implies that $\xi_2(x) \in \mathit{Escaping}(\tau_2)$.

$$\text{Hence,} \quad \xi_2(x) \in \sigma_2(v_2) \cap \mathit{Escaping}(\tau_2) \tag{462}$$
$$\text{By the def. of } B_2, \quad \xi_2(x) \in B_2 \tag{463}$$
$$\text{Hence,} \quad x \in B_1 \implies \xi_2(x) \in B_2 \tag{464}$$

From (464), $B_1 \neq \phi \implies B_2 \neq \phi$. Therefore,

$$\begin{aligned} \mathsf{EE}_{new_1} &= B_1 \times \{f\} \times \{n_\ell\} \\ \mathit{newmap}_1(v_1) &= A_1 \cup \{n_\ell\} \\ \mathsf{EE}_{new_2} &= B_2 \times \{f\} \times \{n_\ell\} \\ \mathit{newmap}_2(v_1) &= A_2 \cup \{n_\ell\} \end{aligned}$$

Proof of Part(i). Say $\langle x, f, y\rangle \in \mathsf{EE}_{new_1}$.

$$\text{By def. of } \mathsf{EE}_{new_1}, \quad x \in B_1 \wedge y = n_\ell \tag{465}$$
$$\text{By (464) ,} \quad \xi_2(x) \in B_2 \tag{466}$$
$$\text{By def. of } \mathsf{EE}_{new_2}, \quad \langle \xi_2(x), f, y\rangle \in \mathsf{EE}_{new_2} \tag{467}$$
$$\text{Hence,} \quad \exists x', f, y' \in \mathsf{IE}_{new_2} \text{ s.t. } \xi_2(x) = \xi_2(x') \wedge \xi_2(y) = \xi_2(y') \tag{468}$$

Proof of Part(ii). Say $x \in \mathit{newmap}_1(v_1)$.

$$\text{By the def. of } \mathit{newmap}_1, \quad x \in A_1 \cup \{n_\ell\} \tag{469}$$

If $x = n_\ell$ then $\xi_2(x) \in \hat{\xi_2}(\mathit{newmap}_2(v_1))$ as $n_\ell \in \mathit{newmap}_2(v_1)$.
If $x \in A_1$ then as shown in *Case(a)* $\xi_2(x) \in \hat{\xi_2}(\mathit{newmap}_2(v_1))$.

Lemma 36. *Let S be any statement. Let $\tau_1 = (\mathsf{EV}_1, \mathsf{EE}_1, \pi_1, \mathsf{IV}_1, \mathsf{IE}_1, \sigma_1)$ and $\tau_2 = (\mathsf{EV}_2, \mathsf{EE}_2, \pi_2, \mathsf{IV}_2, \mathsf{IE}_2, \sigma_2)$. If $[\![S]\!]_a(\tau_1)$ and $[\![S]\!]_a(\tau_2)$ are of the form,*

$$\begin{aligned} [\![S]\!]_a(\tau_1) = (&\mathsf{EV}_1 \cup \mathsf{EV}_{new_1}, \mathsf{EE}_1 \cup \mathsf{EE}_{new_1}, \pi, \mathsf{IV}_1 \cup \mathsf{IV}_{new_1}, \mathsf{IE}_1 \cup \mathsf{IE}_{new_1}, \\ &\lambda var.(var \in \mathit{Vars}_{mod} \to \mathit{newmap}_1(var) \mid \sigma_1(var)) \\ [\![S]\!]_a(\tau_2) = (&\mathsf{EV}_2 \cup \mathsf{EV}_{new_2}, \mathsf{EE}_2 \cup \mathsf{EE}_{new_2}, \pi, \mathsf{IV}_2 \cup \mathsf{IV}_{new_2}, \mathsf{IE}_2 \cup \mathsf{IE}_{new_2}, \\ &\lambda var.(var \in \mathit{Vars}_{mod} \to \mathit{newmap}_2(var) \mid \sigma_2(var)) \end{aligned}$$

and if $(\tau_1, \xi_1) \leq_m (\tau_2, \xi_2)$ then $\tau_{1_m} \preceq_{\xi_2} \tau_{2_m}$.

Proof. By Lemma 35,

$$\langle x, f, y\rangle \in (\mathsf{EE}_{new_1} \cup \mathsf{IE}_{new_1}) \implies \exists \langle x', f, y'\rangle \in (\mathsf{EE}_{new_2} \cup \mathsf{IE}_{new_2}) \text{ s.t. } \xi_2(x) = \xi_2(x') \wedge \xi_2(y) = \xi_2(y') \tag{470}$$
$$var \in \mathit{Vars}_{mod} \implies \hat{\xi_2}(\mathit{newmap}_1(var)) \subseteq \hat{\xi_2}(\mathit{newmap}_2(var)) \tag{471}$$

By the def. of $[\![S]\!]_m$,

$$\tau_{1_m} = apply(\xi_1, [\![S]\!]_a(\tau_1)) \tag{472}$$
$$\tau_{2_m} = apply(\xi_2, [\![S]\!]_a(\tau_2)) \tag{473}$$

Say $\langle u, f, w\rangle \in (\mathsf{EE}_{1_m} \cup \mathsf{IE}_{1_m})$.

$$\text{By (472)}, \quad \exists x, y.\ u = \xi_2(x), w = \xi_2(x), \langle x, f, y\rangle \in (\mathsf{EE}_{1_a} \cup \mathsf{IE}_{1_a}) \tag{474}$$

By the assumption about of $[\![S]\!]_a(\tau_1)$, $\langle x, f, y\rangle \in (\mathsf{EE}_{1_a} \cup \mathsf{IE}_{1_a})$ implies that at least one of the following two conditions must hold.

$$\langle x, f, y\rangle \in (\mathsf{EE}_1 \cup \mathsf{IE}_1) \tag{475}$$
$$\langle x, f, y\rangle \in (\mathsf{EE}_{new_1} \cup \mathsf{IE}_{new_2}) \tag{476}$$

Say condition 475 holds i.e, $\langle x, f, y\rangle \in (\mathsf{EE}_1 \cup \mathsf{IE}_1)$.

$$\text{Since } \tau_1 \preceq_{\xi_2} \tau_2 \quad \xi_2(x), f, \xi_2(y) \in (\mathsf{EE}_2 \cup \mathsf{IE}_2) \tag{477}$$
$$\text{Since } \xi_1 \circ \xi_2 = \xi_2 \quad \xi_2(\xi_1(x)), f, \xi_2(\xi_1(y)) \in (\mathsf{EE}_2 \cup \mathsf{IE}_2) \tag{478}$$
$$\text{By (474)}, \quad \xi_2(u), f, \xi_2(w) \in (\mathsf{EE}_2 \cup \mathsf{IE}_2) \tag{479}$$
$$\text{By the assump. about } [\![S]\!]_a(\tau_2), \quad \xi_2(u), f, \xi_2(w) \in (\mathsf{EE}_{2_a} \cup \mathsf{IE}_{2_a}) \tag{480}$$
$$\text{By (473)}, \quad \xi_2(\xi_2(u)), f, \xi_2(\xi_2(w)) \in (\mathsf{EE}_{2_m} \cup \mathsf{IE}_{2_m}) \tag{481}$$
$$\text{Since } \xi_2 \text{ is idempotent}, \quad \xi_2(u), f, \xi_2(w) \in (\mathsf{EE}_{2_m} \cup \mathsf{IE}_{2_m}) \tag{482}$$

Say condition 476 holds i.e, $\langle x, f, y\rangle \in (\mathsf{EE}_{new_1} \cup \mathsf{IE}_{new_1})$.

$$\text{By (470)}, \quad \exists x', f, y' \in (\mathsf{EE}_{new_2} \cup \mathsf{IE}_{new_2}) \text{ s.t. } \xi_2(x) = \xi_2(x') \wedge \xi_2(y) = \xi_2(y') \tag{483}$$
$$\text{By (473)}, \quad \xi_2(x'), f, \xi_2(y') \in (\mathsf{EE}_{2_m} \cup \mathsf{IE}_{2_m}) \tag{484}$$
$$\text{By (483)}, \quad \xi_2(x), f, \xi_2(y) \in (\mathsf{EE}_{2_m} \cup \mathsf{IE}_{2_m}) \tag{485}$$
$$\text{Since } \xi_1 \circ \xi_2 = \xi_2 \quad \xi_2(\xi_1(x)), f, \xi_2(\xi_1(y)) \in (\mathsf{EE}_{2_m} \cup \mathsf{IE}_{2_m}) \tag{486}$$
$$\text{By (474)}, \quad \xi_2(u), f, \xi_2(w) \in (\mathsf{EE}_{2_m} \cup \mathsf{IE}_{2_m}) \tag{487}$$

Say $u \in \sigma_{1_m}(var)$.

$$\text{By (472)}, \quad \exists x.\ u = \xi_1(x), x \in \sigma_{1_m}(var) \tag{488}$$

By the assumption about of $[\![S]\!]_a(\tau_1)$, $x \in \sigma_{1_m}(var)$ implies that at least one of the following two conditions must hold.

$$var \notin Vars_{mod} \wedge x \in \sigma_1(var) \tag{489}$$
$$var \in Vars_{mods} \wedge x \in newmap_1(var) \tag{490}$$

Say condition 489 holds i.e, $var \notin Vars_{mod} \wedge x \in \sigma_1(var)$

$$\begin{aligned}
\text{Since } \tau_1 \preceq_{\xi_2} \tau_2, &\quad \xi_2(x) \in \sigma_2(var) && (491)\\
\text{Since } \xi_1 \circ \xi_2 = \xi_2, &\quad \xi_2(\xi_1(x))) \in \sigma_2(var) && (492)\\
\text{By (488)}, &\quad \xi_2(u) \in \sigma_2(var) && (493)\\
\text{Since } var \notin Vars_{mod}, &\quad \sigma_2(u) = \sigma_{2_a}(u) && (494)\\
\text{Sub. in (493)}, &\quad \xi_2(u) \in \sigma_{2_a}(var) && (495)\\
\text{By (473)}, &\quad \xi_2(\xi_2(u)) \in \sigma_{2_m}(var) && (496)\\
\text{Since } \xi_2 \text{ is idempotent}, &\quad \xi_2(u) \in \sigma_{2_m}(var) && (497)
\end{aligned}$$

Say condition 490 holds i.e, $var \in Vars_{mod} \wedge x \in newmap_1(var)$.

$$\begin{aligned}
\text{By (471)}, &\quad \xi_2(x) \in \hat{\xi_2}(newmap_2(var)) && (498)\\
\text{Since } var \in Vars_{mod} &\quad newmap_2(var) = \sigma_{2_a}(var) && (499)\\
\text{By (473)}, &\quad \hat{\xi_2}(\sigma_{2_a}(var)) = \sigma_{2_m}(var) && (500)\\
\text{By (499)}, &\quad \hat{\xi_2}(newmap_2(var)) = \sigma_{2_m}(var) && (501)\\
\text{Hence by (498)}, &\quad \xi_2(x) \in \sigma_{2_m}(var) && (502)\\
\text{Since } \xi_1 \circ \xi_2 = \xi_2, &\quad \xi_2(\xi_1(x))) \in \sigma_{2_m}(var) && (503)\\
\text{By (488)}, &\quad \xi_2(u) \in \sigma_{2_m}(var) && (504)
\end{aligned}$$

Hence, for all statement S other than a procedure call statement $[\![S]\!]_m$ is monotonic w.r.t $\leq_m$. We now prove the monotonicity of the procedure call statement.

Lemma 37. *If $\tau_r \preceq_{h_r} \tau_r'$ and $\tau_e \preceq_{h_e} \tau_e'$ then*

$$\begin{aligned}
x \in \eta[\![\tau_e, \tau_r]\!](p) \wedge x \in (\mathsf{IV}_e \cup \mathsf{EV}_e) &\implies h_e(x) \in \eta[\![\tau_e', \tau_r']\!](h_e(p))\\
x \in \eta[\![\tau_e, \tau_r]\!](p) \wedge x \in (\mathsf{IV}_r \cup \mathsf{EV}_r) &\implies h_r(x) \in \eta[\![\tau_e', \tau_r']\!](h_e(p))
\end{aligned}$$

and

$$\begin{aligned}
x \in Escaping(\tau_e\langle\!\langle \tau_r \rangle\!\rangle) \wedge x \in (\mathsf{IV}_e \cup \mathsf{EV}_e) &\implies h_e(x) \in Escaping(\tau_e'\langle\!\langle \tau_r' \rangle\!\rangle)\\
x \in Escaping(\tau_e\langle\!\langle \tau_r \rangle\!\rangle) \wedge x \in (\mathsf{IV}_r \cup \mathsf{EV}_r) &\implies h_r(x) \in Escaping(\tau_e'\langle\!\langle \tau_r' \rangle\!\rangle)
\end{aligned}$$

Proof. Proof not shown for brevity. This can be easily derived from the definition of $\eta[\![\tau_e, \tau_r]\!]$ using the fact that $\tau_r \preceq_{h_r} \tau_r'$ and $\tau_e \preceq_{h_e} \tau_e'$

Lemma 38. *If $(\tau_r, \xi_r) \leq_m (\tau_r', \xi_r')$ and $(\tau_e, \xi_e) \leq_m (\tau_e', \xi_e')$ then*

$$(apply(\xi_j, \tau_e\langle\!\langle \tau_r \rangle\!\rangle), \xi_j) \leq_m (apply(\xi_j', \tau_e'\langle\!\langle \tau_r' \rangle\!\rangle), \xi_j')$$

where $\xi_j = \xi_r \sqcup_p \xi_e$ and $\xi_j' = \xi_r' \sqcup_p \xi_e'$.

Proof. Let

$$\tau_e\langle\langle\tau_r\rangle\rangle = (\mathsf{EV}_a, \mathsf{EE}_a, \pi_a, \mathsf{IV}_a, \mathsf{IE}_a, \sigma_a) \tag{505}$$
$$\tau'_e\langle\langle\tau'_r\rangle\rangle = (\mathsf{EV}'_a, \mathsf{EE}'_a, \pi'_a, \mathsf{IV}'_a, \mathsf{IE}'_a, \sigma'_a) \tag{506}$$
$$\mathit{apply}(\xi_j, \tau_e\langle\langle\tau_r\rangle\rangle) = (\mathsf{EV}_m, \mathsf{EE}_m, \pi_m, \mathsf{IV}_m, \mathsf{IE}_m, \sigma_m) \tag{507}$$
$$\mathit{apply}(\xi'_j, \tau'_e\langle\langle\tau'_r\rangle\rangle) = (\mathsf{EV}'_m, \mathsf{EE}'_m, \pi'_m, \mathsf{IV}'_m, \mathsf{IE}'_m, \sigma'_m) \tag{508}$$

We now prove the following two conditions.

$$\xi_j \leq_p \xi'_j \tag{509}$$
$$\mathit{apply}(\xi_j, \tau_e\langle\langle\tau_r\rangle\rangle) \preceq_{\xi'_j} \mathit{apply}(\xi'_j, \tau'_e\langle\langle\tau'_r\rangle\rangle) \tag{510}$$

Proof of Condition 509. This follows from the property of join.
Proof of Condition 510 which is, Say $\langle u, f, w\rangle \in (\mathsf{EE}_m \cup \mathsf{IE}_m)$.

$$\text{By the def. of } \mathit{apply}, \quad \exists\langle x, f, y\rangle \in (\mathsf{EE}_a \cup \mathsf{IE}_a) \text{ s.t. } \xi_j(x) = u \wedge \xi_j(y) = w \tag{511}$$

Let $\eta = \eta[\![\tau_e, \tau_r]\!]$ and $\eta' = \eta[\![\tau'_e, \tau'_r]\!]$. By the def. of $\tau_r\langle\langle\tau_e\rangle\rangle$, $\langle x, f, y\rangle \in (\mathsf{EE}_a \cup \mathsf{IE}_a)$ iff at least one of the following conditions hold.

$$\langle x, f, y\rangle \in (\mathsf{EE}_r \cup \mathsf{IE}_r) \tag{512}$$
$$\langle p, f, q\rangle \in \mathsf{IE}_e \wedge x \in \eta(p) \wedge y \in \eta(q) \tag{513}$$
$$\langle p, f, y\rangle \in \mathsf{EE}_e \wedge x \in \eta(p) \wedge x \in \mathit{Escaping}(\tau_e\langle\langle\tau_r\rangle\rangle) \tag{514}$$

Say condition 512 holds, i.e, $\langle x, f, y\rangle \in (\mathsf{EE}_r \cup \mathsf{IE}_r)$,

$$\text{Since, } (\tau_r, \xi_r) \leq_m (\tau'_r, \xi'_r), \quad \langle\xi'_r(x), f, \xi'_r(y)\rangle \in (\mathsf{EE}'_r \cup \mathsf{IE}'_r) \tag{515}$$
$$\text{By the def. of } \tau'_e\langle\langle\tau'_r\rangle\rangle, \quad \langle\xi'_r(x), f, \xi'_r(y)\rangle \in (\mathsf{EE}'_a \cup \mathsf{IE}'_a) \tag{516}$$

By the def. of $\mathit{apply}(\xi'_j, \tau'_e\langle\langle\tau'_r\rangle\rangle)$,

$$\langle\xi'_j(x), f, \xi'_j(y)\rangle \in (\mathsf{EE}'_m \cup \mathsf{IE}'_m) \tag{517}$$
$$\text{By Condition 509,} \quad \xi'_j = \xi_j \circ \xi'_j \tag{518}$$
$$\text{Hence,} \quad \xi'_j(x) = \xi'_j(\xi_j(x)) \tag{519}$$
$$\text{By (511),} \quad \xi'_j(x) = \xi'_j(u) \tag{520}$$
$$\text{Similarly,} \quad \xi'_j(y) = \xi'_j(w) \tag{521}$$
$$\text{Sub. in (517),} \quad \langle\xi'_j(u), f, \xi'_j(w)\rangle \in (\mathsf{EE}'_m \cup \mathsf{IE}'_m) \tag{522}$$

Say condition 513 holds, i.e, $\langle p, f, q\rangle \in \mathsf{IE}_e \wedge x \in \eta(p) \wedge y \in \eta(q)$. Since, $(\tau_r, \xi_r) \leq_m (\tau'_r, \xi'_r)$ and $(\tau_e, \xi_e) \leq_m (\tau'_e, \xi'_e)$, by Lemma 37,

$$x \in \eta(p) \wedge x \in (\mathsf{IV}_e \cup \mathsf{EV}_e) \implies \xi'_e(x) \in \eta'(\xi'_e(p)) \tag{523}$$
$$x \in \eta(p) \wedge x \in (\mathsf{IV}_r \cup \mathsf{EV}_r) \implies \xi'_r(x) \in \eta'(\xi'_e(p)) \tag{524}$$

Since $\xi'_e \leq_p \xi'_r \sqcup_p \xi'_e$ and $\xi'_r \leq_p \xi'_r \sqcup_p \xi'_e$

$$\xi'_j(\xi'_e(x)) = \xi'_j(x) \tag{525}$$
$$\xi'_j(\xi'_r(x)) = \xi'_j(x) \tag{526}$$

(523), (524), (525) and (526) imply that

$$x \in \eta(p) \implies \xi'_j(x) \in \widehat{\xi'_j}(\eta'(\xi'_e(p))) \tag{527}$$

Since $(\tau_e, \xi_e) \leq_m (\tau'_e, \xi'_e)$, $\langle p, f, q\rangle \in \mathsf{IE}_e$ implies that $\langle \xi'_e(p), f, \xi'_e(q)\rangle \in \mathsf{IE}'_e$. Hence, $\eta'(\xi'_e(p)) \times \{f\} \times \eta'(\xi'_e(p)) \subseteq (\mathsf{EE}'_a \cup \mathsf{IE}'_a)$.

$$\text{By the def. of } \textit{apply} \text{ and (527)}, \quad \langle \xi'_j(x), f, \xi'_j(y)\rangle \in (\mathsf{EE}'_m \cup \mathsf{IE}'_m) \tag{528}$$
$$\text{By (517)-(522)}, \quad \langle \xi'_j(u), f, \xi'_j(w)\rangle \in (\mathsf{EE}'_m \cup \mathsf{IE}'_m) \tag{529}$$

Say condition 514 holds, i.e, $\langle p, f, y\rangle \in \mathsf{EE}_e \wedge x \in \eta(p) \wedge x \in \mathit{Escaping}(\tau_e \langle\!\langle \tau_r \rangle\!\rangle^S_a)$.

$$\text{Since } (\tau_e, \xi_e) \leq_m (\tau'_e, \xi'_e), \quad \langle p, f, y\rangle \in \mathsf{EE}_e \implies \langle \xi'_e(p), f, \xi'_e(y)\rangle \in \mathsf{EE}'_e \tag{530}$$

By Lemma 37,

$$x \in \mathit{Escaping}(\tau_e \langle\!\langle \tau_r \rangle\!\rangle) \wedge x \in (\mathsf{IV}_e \cup \mathsf{EV}_e) \implies \xi'_e(x) \in \mathit{Escaping}(\tau'_e \langle\!\langle \tau'_r \rangle\!\rangle)(\xi'_e(p))$$
$$x \in \mathit{Escaping}(\tau_e \langle\!\langle \tau_r \rangle\!\rangle) \wedge x \in (\mathsf{IV}_r \cup \mathsf{EV}_r) \implies \xi'_r(x) \in \mathit{Escaping}(\tau'_e \langle\!\langle \tau'_r \rangle\!\rangle)(\xi'_e(p))$$

Which implies that,

$$x \in \mathit{Escaping}(\tau_e \langle\!\langle \tau_r \rangle\!\rangle) \wedge x \in (\mathsf{IV}_r \cup \mathsf{EV}_r) \implies \xi'_j(x) \in \hat{\xi}'_j(\mathit{Escaping}(\tau'_e \langle\!\langle \tau'_r \rangle\!\rangle)(\xi'_e(p))) \tag{531}$$

By abstract semantics, $(\eta'(\xi'_e(p)) \cap \mathit{Escaping}(\tau'_e \langle\!\langle \tau'_r \rangle\!\rangle)) \times \{f\} \times \xi'_e(y) \subseteq (\mathsf{EE}'_a \cup \mathsf{IE}'_a)$

$$\text{By (527),(531)}, \quad \xi'_j(x), f, \xi'_j(y) \in (\mathsf{EE}'_m \cup \mathsf{IE}'_m) \tag{532}$$
$$\text{By (517)-(522)}, \quad \xi'_j(u), f, \xi'_j(u) \in (\mathsf{EE}'_m \cup \mathsf{IE}'_m) \tag{533}$$

Say $u \in \sigma_m(\mathit{var})$.

$$\text{By the def. of } \textit{apply}, \quad \exists x \in \sigma_a(\mathit{var}) \wedge \xi_j(x) = u \tag{534}$$
$$\text{By def. of } \langle\!\langle\rangle\!\rangle, \quad x \in \sigma_a(\mathit{var}) \implies \exists p \in (\mathsf{IV}_e \cup \mathsf{EV}_e) \text{ s.t. } x \in \eta(p) \wedge p \in \sigma_e(\mathit{var}) \tag{535}$$
$$\text{Since } (\tau_e, \xi_e) \leq_m (\tau'_e, \xi'_e), \quad p \in \sigma_e(\mathit{var}) \implies \xi'_e(p) \in \sigma'_e(\mathit{var}) \tag{536}$$
$$\text{By def. of } \langle\!\langle\rangle\!\rangle, \quad \eta'(\xi'_e(p)) \subseteq \sigma'_a(\mathit{var}) \tag{537}$$
$$\text{By (527)}, \quad x \in \eta(p) \implies \xi'_j(x) \in \hat{\xi}'_j(\eta'(\xi'_e(p))) \tag{538}$$
$$\text{Hence}, \quad \xi'_j(x) \in \sigma'_m(\mathit{var}) \tag{539}$$
$$\text{By (517)-(522)}, \quad \xi'_j(u) \in \sigma'_m(\mathit{var}) \tag{540}$$

9.6 Isotonicity Of Node Merging (Lemma 10)

Lemma 39. *NM is isotonic w.r.t* $\leq_m$ *i.e,* $(\tau, \xi) \leq_m NM(\tau, \xi)$.

Proof. Let $(\tau_n, \xi_n) = NM(\tau, \xi)$. By the def. of NM, $\xi_n = \xi \sqcup_p ChooseNodesToMerge(\tau)$. Hence, $\xi \leq_p \xi_n$. $\tau \preceq_{\xi_n} \tau_n$ follows from the definition of *apply*.

A Implementation Details

We implemented the purity analysis along with the optimizations described in the previous sections using the *Microsoft Phoenix* program analysis framework [12] which provides support for implementing custom static analyses for Microsoft.NET DLLs. At a high level, given a DLL, our implementation performs a bottom-up traversal of the call-graph iterating over the Strongly connected components (SCCs) until a fix-point is reached. However, instead of arbitrarily iterating over the the methods in the SCC we use an iteration strategy based on the ideas presented in [10] which we empirically found to be effective. To compute the side-effects of a method, we maintain a "may-write-set" along with transformer graphs that tracks all the fields of the abstract objects that may be mutated (i.e, written) by a method.

A.1 Language Extensions

To analyse real world C# programs we extend the analysis presented in the previous sections to support the language extensions discussed below.

Modelling exceptions. We track all the (abstract) exception objects that can be thrown by a method using a special variable *throw*. While analysing catch blocks, we use the *throw* variable's points-to set to infer the possible exception objects that can be caught by the *catch* block. As a special case, we do not track the implicit exceptions that can thrown by the runtime like *NullPointerException, ArrayOutOfBoundException* etc. as the fields of these exceptions are *read only* and cannot be mutated by the library methods. As an optimization, we merge all the abstract exception objects that can be thrown by a method into a single abstract vertex to prevent the explosion of summary graphs.

Modelling static fields accesses. We consider a static field access (say $C.f$ where C is a class) as an access to the field f of a globally accessible external vertex called *global load vertex*. The *global load vertex* is treated as an implicit parameter to every method and is a part of the summary of every method. The reads to the static fields will result in external edges from the *global load vertex* and the writes would result in internal edges.

Handling virtual method calls. Since our analysis does not have access to points-to information (in fact, our analysis computes the points-to information) we use a class hierarchy analysis to find the targets of a virtual method call. However, when the abstract objects pointed to by the receiver of a virtual method call are all internal nodes (i.e, allocated inside the method being analysed or its

transitive callees) we use the type(s) of the internal nodes to determine the targets of the virtual method call.

Handling method delegates. In C# it is possible to invoke methods through *method delegates* which is similar to the function pointers in C. We assume that all method delegates are pure and only return newly created object. Our analysis is sound only when this assumption holds. We describe the rationale behind choosing this approach (instead of a conservative approach) in the later part of the section when we discuss unanalyzable calls.

A.2 Other implementation Issues

Supporting libraries distributed across several DLLs. In general, the DLLs can have dependencies on other DLLs (especially, on the base C# library which comprises of three DLLs viz. *mscorlib.dll, System.dll, System.Core.dll*). Hence, it is necessary to analyse all the DLLs together as a single large program. However, due to the limitations of the phoenix analysis framework we analyse a single DLL at a time but store all the necessary information about the analysed DLL, particularly the class hierarchy and the method summaries, in a database and use the stored information while analysing other dependent DLLs. However, this approach is sound only when the DLLs do not have cyclic dependencies.

Handling unanalyzable calls. We refer to the calls made to the native methods and methods defined in DLLs which cannot be analysed by the phoenix framework as unanalyzable calls. Like in the case of method delegates, we assume that all unanalyzable method calls are pure and return newly created objects. The summaries that we compute are sound only if this assumption holds. An alternative approach would be to make a conservative assumption by treating all the unanalyzable methods as mutating a static field. However we do not make a conservative assumption as, firstly, it results in too many false positives and secondly, the loss of precision due to the optimizations that we propose may get masked by the spurious impurities introduced due to the unanalyzable calls. In all the experiments that we carried out we found that, in most cases, the unanalyzable calls were due to GUI accesses and Database accesses which are defined in the native DLLs that cannot be analysed by our implementation.

References

1. Calcagno, C., Distefano, D., O'Hearn, P.W., Yang, H.: Compositional shape analysis by means of bi-abduction. In: POPL. pp. 289–300 (2009)
2. Chatterjee, R., Ryder, B.G., Landi, W.A.: Relevant context inference. In: POPL. pp. 133–146 (1999)
3. Cheng, B.C., Hwu, W.M.W.: Modular interprocedural pointer analysis using access paths: design, implementation, and evaluation. In: PLDI. pp. 57–69 (2000)
4. Codeplex. http://www.codeplex.com (March 2011)
5. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Log. Comput. 2(4), 511–547 (1992)

6. Cousot, P., Cousot, R.: Modular static program analysis. In: CC. pp. 159–178 (2002)
7. Gulavani, B.S., Chakraborty, S., Ramalingam, G., Nori, A.V.: Bottom-up shape analysis. In: SAS. pp. 188–204 (2009)
8. Gulavani, B.S., Henzinger, T.A., Kannan, Y., Nori, A.V., Rajamani, S.K.: SYNERGY: a new algorithm for property checking. In: FSE. pp. 117–127 (2006)
9. Jeannet, B., Loginov, A., Reps, T., Sagiv, M.: A relational approach to interprocedural shape analysis. ACM Trans. Program. Lang. Syst. 32, 5:1–5:52 (February 2010), http://doi.acm.org/10.1145/1667048.1667050
10. Kanamori, A., Weise, D.: Worklist management strategies for data flow analysis. Tech. rep., Microsoft Research (1994)
11. Knoop, J., Steffen, B.: The interprocedural coincidence theorem. In: CC. pp. 125–140 (1992)
12. Phoenix. https://connect.microsoft.com/Phoenix (March 2011)
13. Prabhu, P., Ramalingam, G., Vaswani, K.: Safe programmable speculative parallelism. In: PLDI. pp. 50–61 (2010)
14. Rinetzky, N., Sagiv, M., Yahav, E.: Interprocedural shape analysis for cutpoint-free programs. In: SAS. pp. 284–302 (2005)
15. Sagiv, S., Reps, T.W., Wilhelm, R.: Parametric shape analysis via 3-valued logic. In: POPL. pp. 105–118 (1999)
16. Salcianu, A.D.: Pointer Analysis and its Applications for Java Programs. Master's thesis, Massachusetts institute of technology (2001)
17. Salcianu, A.D., Rinard, M.C.: Purity and side effect analysis for java programs. In: In VMCAI. Springer-Verlag (2005)
18. Sharir, M., Pnueli, A.: Two approaches to interprocedural data flow analysis. In: Program Flow Analysis: Theory and Applications. pp. 189–234 (1981)
19. Whaley, J., Rinard, M.C.: Compositional pointer and escape analysis for java programs. In: OOPSLA. pp. 187–206 (1999)