



PAPER

# The role of linearity in sharing analysis

Gianluca Amato , Maria Chiara Meo and Francesca Scozzari\* 

University of Chieti–Pescara, Pescara, Italy

\*Corresponding author. Email: [francesca.scozzari@unich.it](mailto:francesca.scozzari@unich.it)

(Received 12 May 2021; revised 30 December 2021; accepted 10 May 2022; first published online 14 June 2022)

## Abstract

Sharing analysis is used to statically discover data structures which may overlap in object-oriented programs. Using the abstract interpretation framework, we show that sharing analysis greatly benefits from linearity information. A variable is linear in a program state when different field paths starting from it always reach different objects. We propose a graph-based abstract domain which can represent aliasing, linearity, and sharing information and define all the necessary abstract operators for the analysis of a Java-like language.

**Keywords:** Sharing analysis; linearity; aliasing; object-oriented programming

## 1. Introduction

In the context of static analysis of object-oriented programs, the aim of sharing analysis is to discover when two data structures may overlap. For instance, this may happen in Java programs, whose objects are stored in a shared memory called heap. Sharing information can be exploited in program parallelization and distribution, since methods working on data structures that do not overlap can be executed on different processors, using disjoint memory. Moreover, knowing sharing information is very useful for improving other kinds of analyses like shape, pointer, class, and cyclicity analysis.

Consider a class `Tree` with two fields `l` and `r` defined as follows:

```
class Tree { Tree l; Tree r; }
```

A concrete state in an object-oriented program is usually described by a *frame*, which is a map from program variables to memory locations (or `null`), and a *memory*, which is a map from locations to objects. Figure 1 shows two states with two variables `x` and `y` referring to two different objects of class `Tree`. In the state of Figure 1A, the variables `x` and `y` share, since `x.r.r.r` and `y.l.l` point to the same object.

Traditionally, sharing analysis has been designed in two different ways: set-sharing analysis and pair-sharing analysis. In set-sharing analysis, we look for sets of variables which share a common object, while in pair-sharing analysis we are only interested in discovering pairs of variables which share. In this paper, we deal with field-sensitive pair-sharing properties.

We abstract concrete states into a new kind of graph we call *Aliasing Linearity Pair Sharing (ALPS) graph*. For instance, the state in Figure 1A is abstracted into the ALPS graph in Figure 1A where the dotted edges between two nodes encode the information that they share. All ALPS graphs have at most two levels. The first level consists of nodes which are labeled with program variables and may have both incoming and outgoing edges. Second level nodes are unlabeled, are

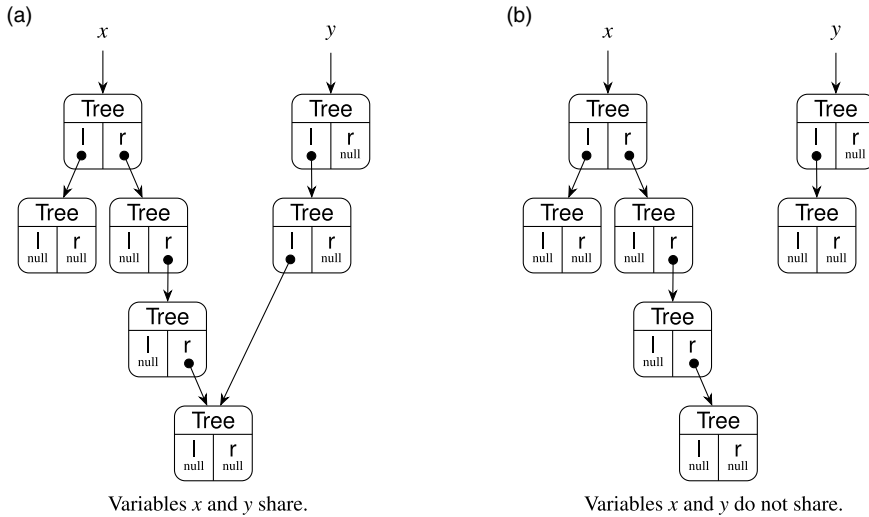


Figure 1. Two concrete states illustrating pair sharing.

connected by at least an incoming edge with first level nodes, and cannot have outgoing edges. Edges are labeled with field names. Dotted edges encode sharing information and can connect two nodes at any level.

As it is customary in the abstract interpretation theory (Cousot and Cousot 1977, 1992), the correspondence between ALPS graphs and concrete states is given by a concretization function that maps each graph to the set of all concrete states it abstracts. In this section, we try to describe the main feature of ALPS graphs while remaining at an informal level. Precise definitions will be given in the next sections.

### 1.1 Field-sensitive pair-sharing information

Sharing properties for logic programs has been studied extensively. The large literature on this topic and the paper by Secci and Spoto (2005a) on object-oriented programs have been the starting point for designing our abstract domain for sharing analysis.

We say that two locations share when it is possible to reach from them a common location. Consider the concrete state depicted in Figure 1A. Here  $x$  and  $y$  are bound to two different data structures which overlap, since  $x.r.r.r$  and  $y.l.l$  are bound to the same object: we say that  $x$  and  $y$  share. In the abstract graph, we represent this information with an (undirected) dotted edge between the two nodes. For instance, the sharing information in Figure 1A is captured by the four dotted edges in Figure 2A. In the following, in order to keep the graphs as simple as possible, we omit those dotted edges which can be inferred by other features of the graph. In this case, the only nonredundant edge is the one between  $x.r$  and  $y.l$ , as shown in Figure 2B.

Note that our graphs encode *possible pair-sharing* information: the presence of a dotted edge means that the corresponding locations might share, while the absence of the dotted edge means that the corresponding locations do not share for sure. Therefore, the graphs in Figure 2 are correct abstractions also for the concrete state in Figure 1B, although  $x.r$  and  $y.l$  do not share in the latter.

### 1.2 Aliasing and nullness

ALPS graphs may encode *definite nullness* of variables and fields: a variable is null when it does not appear as a label, while a field  $v.f$  is null when there is no edge labeled with  $f$  departing from the node  $v$ . For example, in Figure 4 the field  $v_1.x$  is null. Definite nullness means that when a node

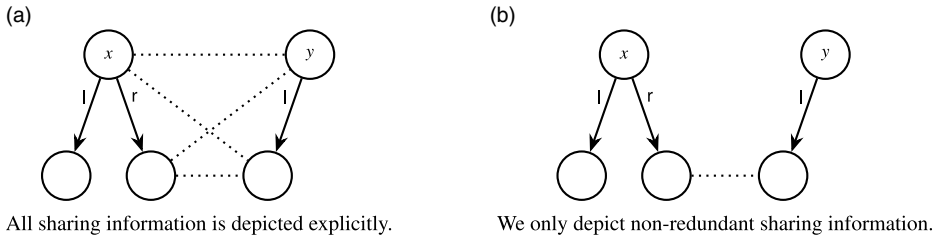


Figure 2. Abstractions of the concrete states in Figure 1.

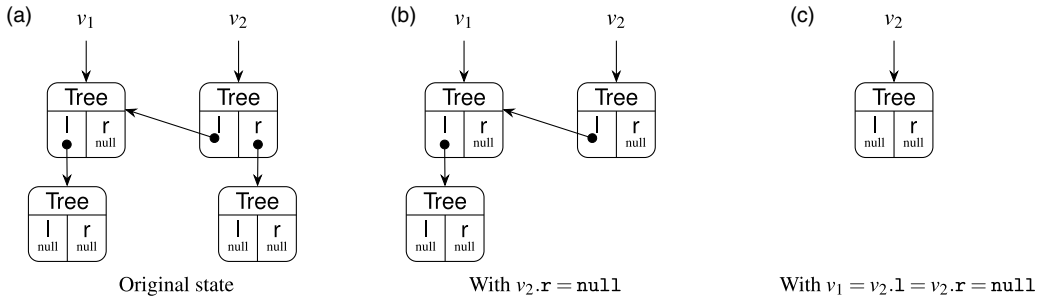


Figure 3. Three concrete states illustrating nullness and aliasing.

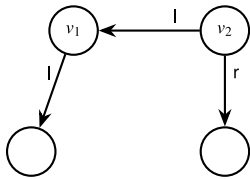


Figure 4. Abstraction of the concrete states in Figure 3. Both  $v_1$  and  $v_2$  are first level nodes, even if there is an edge pointing to  $v_1$ .

is null in the graph, then it has to be null in the corresponding concrete states, but the converse is not true. For example, all the graphs in Figure 3 are correctly abstracted by Figure 4, since in all of them  $v_1.r$  is null, although  $v_2.r$  is null in some states (Figure 3B and C) but not in others (Figure 3A).

The graph also encodes definite *weak aliasing*: two variables or fields are weakly aliased when they point to the same location or they point both to null. For instance, the variable  $v_1$  and the field  $v_2.l$  in the concrete state of Figure 3A are aliased, since they are bound to the same object. ALPS graphs encode aliasing information by using a single node for abstracting both variables/fields. For example, in Figure 4, the same node is labeled by both  $v_1$  and the  $v_2.l$  (more precisely, the node  $v_1$  is reached by the edge labeled  $l$  departing from the node  $v_2$ ). Note that, due to the definition of weak aliasing,  $v_1$  and  $v_2.l$  are considered to be aliased even in the concrete state of Figure 3C, hence Figure 3C is still correctly abstracted by Figure 4. Another example is in Figure 5.

We use the adjective *weak* to distinguish our treatment of aliasing by other proposals which consider two identifiers to be aliased only if they point to the same *non-null* location (see, e.g., Pollet et al. 2001).

### 1.3 Linearity information

In the field of logic programming, the use of a linearity property has proved to be very useful when dealing with sharing information (see Bagnara et al. 2005 for a comprehensive evaluation). We show how the same idea can be reused to enhance sharing analysis of object-oriented programs. We propose a new combined analysis of sharing, aliasing, and linearity properties for



Figure 5. Concrete and abstract states illustrating aliasing.

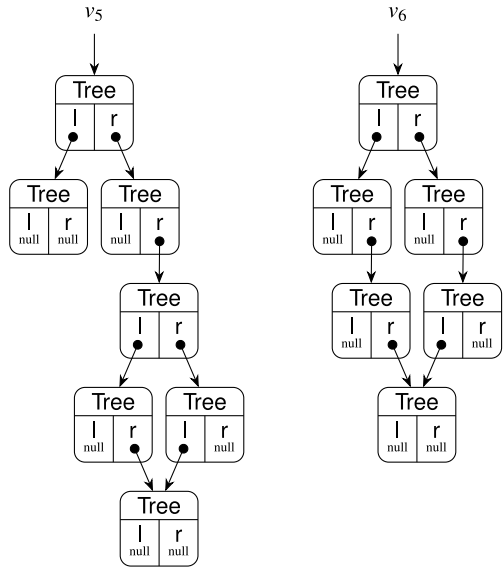


Figure 6. A concrete state illustrating nonlinearity.

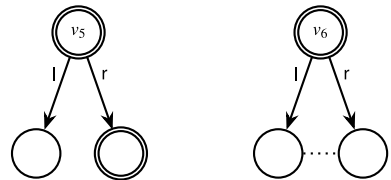


Figure 7. Abstraction of the concrete state in Figure 6.

Java-like programs based on abstract interpretation, inspired by the corresponding domains on logic programs.

We say that a location is *nonlinear* when there are two different paths starting from it and reaching a common location. Consider for instance Figure 6. Starting from  $v_5.r$ , we reach the same object by either  $v_5.r.r.l.r$  or  $v_5.r.r.r.l$ . Therefore, we say that  $v_5.r$  is nonlinear. It is easy to note that also  $v_5$  is nonlinear. In general, whenever a field  $v.f$  is nonlinear, the variable  $v$  is nonlinear too. We represent *possible nonlinearity* information by means of a double circle. For instance, the concrete state for variables  $v_5$  and  $v_6$  in Figure 6 is abstracted as in Figure 7.

### 1.4 An example program

Linearity plays a key role in sharing analysis, since it allows us to propagate precise sharing information when dealing with method calls. We show how the analysis works and the relevance of linearity information with the help of the example program in Figure 8.

```

Tree makeTree(n:Integer)
  with m:Integer is {
    m = 0;
    if (n = m)
      out = null;
    else {
      out = new Tree;
      m = n-1;
      out.l = makeTree(m);
      out.r = makeTree(m);
    } }

void useTree()
  with m:Integer,
    t:Tree, t1:Tree,
    left:Tree,
    right:Tree is {
    m = 10;
    t = makeTree(m);
    t1 = t.l;
    right = t1.r;
    left = t1.l;
  }

```

Figure 8. The example program.

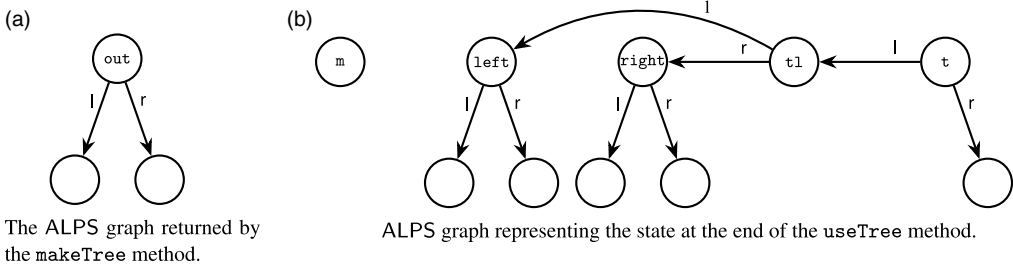


Figure 9. Two ALPS graphs for the example program.

We consider again the class `Tree` previously defined. The method `makeTree` defined in Figure 8 (left) builds a complete tree of depth  $n$ , whose nodes are all distinct. Actually, with a bottom-up static analysis using ALPS graphs, we can easily infer that, for any input  $n \geq 2$ , `makeTree` returns a data structure which may be described by the graph in Figure 9A, where the label `out` denotes the return value of the method. Since there are no undirected dotted edges between `out.l` and `out.r`, it means that `out.l` and `out.r` do not share. Moreover, since there are no double circles, everything is guaranteed to be linear. The latter property implies that, in any concrete state approximated by the ALPS graph in Figure 9A, two different fields of the same object can never share. In particular, we know that `out.l.l` and `out.l.r` do not share.

The `useTree` method in Figure 8 (right) calls `makeTree` and extracts two subtrees which do not share. In detail, in the `useTree` method, since we know that `t` is linear, we can infer that `t1` is linear too. Since `t1` is linear, its fields `t1.r` and `t1.l` do not share, and therefore `right` and `left` do not share. Note that linearity of `t` is not needed to prove that `t.l` and `t.r` do not share (sharing is enough for this). We need linearity when we want to go deeper and prove that `t.l.l` and `t.l.r` do not share. Linearity of `t` is essential here in proving that `left` and `right` do not share. The heap at the end of the `useTree` method may be described by the graph in Figure 9B.

Due to the interaction between sharing and linearity, different ALPS graphs may represent the same set of concrete states. For example, adding a dotted edge between `out.l` and `out.r` in Figure 9A does not allow them to share, since that would violate the linearity of `out`. In Section 5, we will introduce a closure operator on graphs to deal with these interactions.

### 1.5 Plan of the paper

The rest of the paper is organized as follows. Section 2 summarizes the notations used through the paper and describes our simple Java-like language. Section 3 defines the basic notions of reachability, sharing, linearity, and aliasing. In Section 4, we introduce the domain of aliasing graphs, which encodes weak aliasing for variables and fields. Then, in Section 5 we enrich aliasing graphs

with information regarding sharing and linearity, obtaining what we call ALPS graphs. Section 6 defines an abstract semantics (analysis) over ALPS graphs and states its correctness. Section 7 contains a discussion about related work and Section 8 concludes. Appendix A contains all the proofs of the properties and theorems in the paper.

This paper is an extended and revised version of Amato et al. (2015). With respect to the conference version, this paper includes: (1) definitions for all the operators used in the abstract and concrete semantics; (2) the notion of graph morphism and closure; (3) all the proofs; and (4) many new examples.

## 2. Preliminaries

### 2.1 Notations

We use a special notation for ordered pairs. The two components of an *ordered pair* are separated by  $\star$ . A definition of a pair  $s = a \star b$  silently defines the *pair of selectors*  $s.a$  and  $s.b$ .

A total (partial) function  $f$  from  $A$  to  $B$  is denoted by  $A \rightarrow B$  ( $A \dashrightarrow B$ , respectively). Given  $f : A \dashrightarrow B$  and  $x \in A$ , we write  $f(x) = \perp$  when  $f$  is undefined on  $x$ . The composition of functions  $f$  and  $g$  is denoted by  $f \circ g$ . The *domain* and *range* of  $f$  are, respectively,  $\text{dom}(f)$  and  $\text{rng}(f)$ . We denote by  $[v_1 \mapsto t_1, \dots, v_n \mapsto t_n]$  the function  $f$  such that  $\text{dom}(f) = \{v_1, \dots, v_n\}$  and  $f(v_i) = t_i$  for  $i = 1, \dots, n$ . We denote by  $f[w_1 \mapsto d_1, \dots, w_m \mapsto d_m]$  an *update* of  $f$ , with a possibly enlarged domain. By  $f|_s$  ( $f|_{-s}$ ), we denote the *restriction* of  $f$  to  $s \subseteq \text{dom}(f)$  (to  $\text{dom}(f) \setminus s$ , respectively).

Given a set  $X$ , we denote by  $\mathcal{P}(X)$  the *powerset* of  $X$  and with  $\mathcal{P}_2(X)$  those subsets of  $X$  of cardinality 1 or 2. Given  $f : A \dashrightarrow B$ ,  $Y \in \mathcal{P}(B)$  and  $Z \subseteq \mathcal{P}_2(B)$ , we denote by  $f^{-1}(Y) = \{a \in A \mid f(a) \in Y\}$  and  $f^{-1}(Z) = \{\{a, a'\} \in \mathcal{P}_2(A) \mid \{f(a), f(a')\} \in Z\}$ .

For an *ordered set*  $S \star \leq$ , if  $s \in S$ , then  $\downarrow s = \{s' \in S \mid s' \leq s\}$  is the *downward closure* of  $S$ . For a *preordered set*  $S \star \leq$ , we say that  $s_1, s_2 \in S$  are *equivalent*, and we write  $s_1 \sim s_2$ , when  $s_1 \leq s_2$  and  $s_2 \leq s_1$ . The set  $S/\sim$  of equivalence classes modulo  $\sim$  is ordered by  $[s_1] \leq [s_2]$  iff  $s_1 \leq s_2$ . We will freely use preordered sets in the place where ordered sets are expected, implicitly referring to the induced ordered set.

We recall now the basics of abstract interpretation from Cousot and Cousot (1977, 1992). Given two posets  $C \star \leq$  and  $A \star \leq$  (the concrete and the abstract domain), a *Galois connection* is a pair of monotonic maps  $\alpha : C \mapsto A$  and  $\gamma : A \mapsto C$  such that  $\gamma \circ \alpha$  is extensive and  $\alpha \circ \gamma$  is reductive. It is a *Galois insertion* when  $\alpha \circ \gamma$  is the identity map, that is, when the abstract domain does not contain *useless* elements. This is equivalent to  $\alpha$  being onto, or  $\gamma$  one-to-one.

We say that  $a \in A$  is a *correct approximation* of  $c \in C$  when  $\alpha(c) \leq a$  or, equivalently,  $c \leq \gamma(a)$ . An abstract operator  $f^A : A^n \mapsto A$  is *correct* w.r.t.  $f : C^n \rightarrow C$  if, given  $a_1, \dots, a_n$  correct abstractions of  $c_1, \dots, c_n$ , we have that  $f^A(a_1, \dots, a_n)$  is a correct abstraction of  $f(c_1, \dots, c_n)$ . This is equivalent to  $f(\gamma(a_1), \dots, \gamma(a_n)) \leq \gamma(f^A(a_1, \dots, a_n))$  for every tuple  $a_1, \dots, a_n$  of abstract objects.

### 2.2 The language

We use the Java-like object-oriented language defined by Secci and Spoto (2005a), which is a normalized version of Java with downward casts, and which we extend with upper casts. The details of the concrete semantics first appeared in a long version of the above paper (Secci and Spoto 2005b) which is unpublished. Here we present the semantics for the sake of completeness.

#### 2.2.1 Syntax

Each program has a set of *identifiers*  $Ide$  and a finite set of *classes* (or *types*)  $\mathcal{K}$  ordered by a *subclass relation*  $\leq$  such that  $\mathcal{K} \star \leq$  is a poset. The set  $Ide$  includes the special identifiers `this`, `res`, `out`. Since we do not allow multiple inheritance, for any class  $\kappa \in \mathcal{K}$ , the set  $\{\kappa' \mid \kappa' \geq \kappa\}$  is a chain. In the following, we assume that  $Ide$  and  $\mathcal{K}$  have been fixed beforehand.

We use type environments to describe the identifiers in scope in a given program point. A *type environment* is a map from a finite set of identifiers to the associated class. The set of type environments is

$$TypEnv = \{\tau : Ide \rightsquigarrow \mathcal{K} \mid \text{dom}(\tau) \text{ is finite}\}.$$

We call *variables* the identifiers in  $\text{dom}(\tau)$ . Any class  $\kappa \in \mathcal{K}$  defines a type environment, also denoted by  $\kappa$ , which maps the *fields* of  $\kappa$  (including both the fields defined in  $\kappa$  and those inherited by the superclasses) to their types.

We require that fields cannot be redefined in subclasses. It means that if  $f \in \text{dom}(\kappa)$  and  $\kappa' \leq \kappa$ , then  $f \in \text{dom}(\kappa')$  and  $\kappa'(f) = \kappa(f)$ . For consistency of notation, we write  $\kappa.f$  in place of  $\kappa(f)$  for the type of the field  $f$  in the class  $\kappa$ .

Finally, we require the existence of a class  $\top$  with no fields which is the common ancestor of all other classes.

In the examples, we will describe the set of classes and the corresponding type environment using a notation inspired by class definitions in Java.

**Example 1.** Classes in the example program in Section 1.4 may be described by the following Java-like syntax:

```
class Tree { Tree l; Tree r; }
class Integer { }
```

Formally, we have  $\mathcal{K} = \{\top, \text{Tree}, \text{Integer}\}$  with a flat ordering. Moreover,  $\text{Tree} = [l \mapsto \text{Tree}, r \mapsto \text{Tree}]$  and  $\text{Integer} = []$ .

Expressions and commands are normalized versions of those in Java. Their syntax is the following:

$$\begin{aligned} \text{exp} ::= & \text{null } \kappa \mid \text{new } \kappa \mid v \mid v.f \mid (\kappa) v \mid v.m(v_1, \dots, v_n) \\ \text{com} ::= & v := \text{exp} \mid v.f := \text{exp} \mid \{ \text{com}; \dots; \text{com} \} \\ & \mid \text{if } v = w \text{ then } \text{com} \text{ else } \text{com} \mid \text{if } v = \text{null} \text{ then } \text{com} \text{ else } \text{com} \end{aligned}$$

where  $\kappa \in \mathcal{K}$ ,  $f \in Ide$  and  $v, w, v_1, \dots, v_n \in Ide \setminus \{\text{res}\}$  are distinct when they appear in the same clause. Each method  $\kappa.m$  of a class  $\kappa$  is *defined* with a statement like

$$\kappa_0.m(w_1:\kappa_1, \dots, w_n:\kappa_n) \text{ with } w_{n+1}:\kappa_{n+1}, \dots, w_{n+m}:\kappa_{n+m} \text{ is } \text{com}$$

where  $w_1, \dots, w_n, w_{n+1}, \dots, w_{n+m} \in Ide$  are distinct and are not `res` nor `this` nor `out`. Their *declared types* are  $\kappa_1, \dots, \kappa_n, \kappa_{n+1}, \dots, \kappa_{n+m} \in \mathcal{K}$ , respectively. Variables  $w_1, \dots, w_n$  are the *formal parameters* of the method, and  $w_{n+1}, \dots, w_{n+m}$  are its *local variables*. The method can also use a variable `out` of type  $\kappa_0$  which holds its *return value*. We define  $\text{body}(\kappa.m) = \text{com}$  and  $\text{returnType}(\kappa.m) = \kappa_0$ . Overriding methods cannot change the formal parameters but may specialize the return type.

Given a type environment  $\tau$ , with an abuse of notation we denote with  $\tau(\text{exp})$  the *static type* of an expression  $\text{exp}$ , defined as follows:

$$\begin{aligned} \tau(v.f) &= \tau(v).f & \tau(\text{new } \kappa) &= \tau(\text{null } \kappa) = \tau((\kappa) v) = \kappa \\ \tau(v.m(v_1, \dots, v_n)) &= \text{returnType}(\tau(v).m). \end{aligned}$$

Note that the static type of a field of class  $\kappa$  is recovered from the definition of  $\kappa$ , while the static type of the return value of a method call is the return type of the method.

We require expressions, commands, and methods to be well-typed, according to the standard definition in Java. We also require that all casts are explicit, so that in any assignment  $v := \text{exp}$  (resp.  $v.f := \text{exp}$ ) the types of  $v$  (resp.  $v.f$ ) and  $\text{exp}$  coincide. The same is required for formal and actual parameters. This is not a limitation since we allow upward and downward casts.

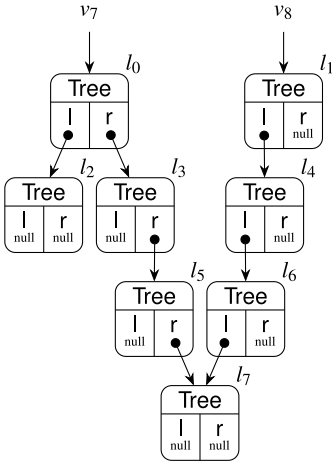


Figure 10. A concrete state with variables  $v_7, v_8$ .

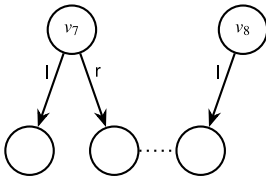


Figure 11. Abstraction of the concrete state in Figure 10.

2.2.2 Semantics

The semantics of the language is defined by means of *frames*, *objects*, and *memories* defined as follows:

$$Frame_\tau = \{\phi \mid \phi \in \text{dom}(\tau) \rightarrow Loc \cup \{\text{null}\}\}$$

$$Obj = \{\kappa \star \phi \mid \kappa \in \mathcal{K}, \phi \in Frame_\kappa\}$$

$$Memory = \{\mu \in Loc \rightarrow Obj \mid \text{dom}(\mu) \text{ is finite}\}$$

where *Loc* is an infinite set of *locations*. A frame binds identifiers to locations or null. A memory binds such locations to objects, which contain a class tag and the frame for their fields. A new object of class  $\kappa$  is  $\text{new}(\kappa) = \kappa \star \phi$ , with  $\phi(v) = \text{null}$  for each  $v \in \text{dom}(\kappa)$ .

**Example 2.** Let  $\tau = [v_7 \mapsto \text{Tree}, v_8 \mapsto \text{Tree}]$  and consider the state depicted in Figure 10, whose abstraction is in Figure 11. We have that  $\phi = [v_7 \mapsto l_0, v_8 \mapsto l_1]$  and

$$\begin{aligned} \mu = & [l_0 \mapsto \text{Tree} \star [l \mapsto l_2, r \mapsto l_3], l_1 \mapsto \text{Tree} \star [l \mapsto l_4, r \mapsto \text{null}], \\ & l_2 \mapsto \text{Tree} \star [l \mapsto \text{null}, r \mapsto \text{null}], l_3 \mapsto \text{Tree} \star [l \mapsto \text{null}, r \mapsto l_5], \\ & l_4 \mapsto \text{Tree} \star [l \mapsto l_6, r \mapsto \text{null}], l_5 \mapsto \text{Tree} \star [l \mapsto \text{null}, r \mapsto l_7], \\ & l_6 \mapsto \text{Tree} \star [l \mapsto l_7, r \mapsto \text{null}], l_7 \mapsto \text{Tree} \star [l \mapsto \text{null}, r \mapsto \text{null}]] . \end{aligned}$$

Our language is strongly typed, which means that the static type of an expression should be consistent with its runtime type. We formalize here the notion of *type correctness* for a frame and a memory.

**Definition 3** (Types of locations). Let  $\phi \in Frame_\tau, \mu \in Memory$ , and  $l \in \text{dom}(\mu)$ , we write  $\tau(l) = \mu(l).\kappa$  for the runtime type of location  $l$ .

If  $v$  is a variable, the object associated with  $v$  should be of a subtype of the static type of  $v$ . This leads to the definition of weak correctness:



**Definition 4** (Weak  $\tau$ -correctness). Let  $\phi \in \text{Frame}_\tau$  and  $\mu \in \text{Memory}$ . We say that  $\phi \star \mu$  is weakly  $\tau$ -correct if for every  $v \in \text{dom}(\phi)$  such that  $\phi(v) \neq \text{null}$  we have  $\phi(v) \in \text{dom}(\mu)$  and  $\tau(\phi(v)) \leq \tau(v)$ .

We strengthen the correctness notion of Definition 4 by requiring that it also holds for the fields of the objects in memory.

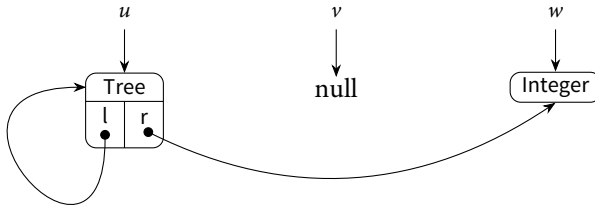
**Definition 5** ( $\tau$ -correctness). Let  $\phi \in \text{Frame}_\tau$  and  $\mu \in \text{Memory}$ . We say that  $\phi \star \mu$  is  $\tau$ -correct and write  $\phi \star \mu : \tau$ , if

- (1)  $\phi \star \mu$  is weakly  $\tau$ -correct and,
- (2) for every  $o \in \text{rng}(\mu)$ ,  $o.\phi \star \mu$  is weakly  $o.\kappa$ -correct.

We call *state* a pair  $\phi \star \mu$  which is  $\tau$ -correct for some type environment  $\tau$ . The set of  $\tau$ -correct states is

$$\Sigma_\tau = \{\phi \star \mu \mid \phi \in \text{Frame}_\tau, \mu \in \text{Memory}, \phi \star \mu : \tau\}.$$

**Example 6 (Noncorrect state).** Let  $\tau \in \text{TypEnv} = [u \mapsto \text{Tree}, v \mapsto \text{Tree}, w \mapsto \text{Integer}]$ ,  $\phi \in \text{Frame}_\tau = [u \mapsto l_0, v \mapsto \text{null}, w \mapsto l_1]$  and  $\mu \in \text{Memory}$  such that  $\mu(l_1) = \text{Integer} \star []$  and  $\mu(l_0) = \text{Tree} \star [l \mapsto l_0, r \mapsto l_1]$ . This may be depicted graphically as follows:



It turns out that  $\phi \star \mu$  is weakly  $\tau$ -correct, but it is not  $\tau$ -correct, since the right child of  $u$  points to an integer node instead of a tree node.

The semantics of an expression is a partial map  $\mathcal{E}_\tau^I[\![\text{exp}]\!] : \Sigma_\tau \rightarrow \Sigma_{\tau+\text{exp}}$  from an initial to a final state, containing a distinguished variable  $\text{res}$  holding the value of the expression, where  $\tau + \text{exp} = \tau[\text{res} \mapsto \tau(\text{exp})]$ .

**Definition 7** (Semantics of expressions). Let  $\tau$  describe the variables in scope and  $I$  be an interpretation. The semantics for expressions  $\mathcal{E}_\tau^I[\![\text{exp}]\!] : \Sigma_\tau \rightarrow \Sigma_{\tau+\text{exp}}$  is defined as

$$\begin{aligned} \mathcal{E}_\tau^I[\![\text{null } \kappa]\!](\phi \star \mu) &= \phi[\text{res} \mapsto \text{null}] \star \mu \\ \mathcal{E}_\tau^I[\![\text{new } \kappa]\!](\phi \star \mu) &= \phi[\text{res} \mapsto l] \star \mu[l \mapsto \text{new}(\kappa)] \quad \text{with } l \in \text{Loc} \setminus \text{dom}(\mu) \\ \mathcal{E}_\tau^I[\![v]\!](\phi \star \mu) &= \phi[\text{res} \mapsto \phi(v)] \star \mu \\ \mathcal{E}_\tau^I[\![v.f]\!](\phi \star \mu) &= \begin{cases} \phi[\text{res} \mapsto \phi(v).f] \star \mu & \text{if } \phi(v) \neq \text{null} \\ \perp & \text{otherwise} \end{cases} \\ \mathcal{E}_\tau^I[\![\kappa v]\!](\phi \star \mu) &= \begin{cases} \phi[\text{res} \mapsto \phi(v)] \star \mu & \text{if } \phi(v) = \text{null or} \\ & \phi(v) \neq \text{null and } \{\tau(\phi(v)), \kappa\} \text{ is a chain} \\ \perp & \text{otherwise} \end{cases} \\ \mathcal{E}_\tau^I[\![v.m(v_1, \dots, v_n)]\!](\phi \star \mu) &= \begin{cases} \phi[\text{res} \mapsto \phi'(\text{out})] \star \mu' & \text{if } \phi(v) \neq \text{null} \\ \perp & \text{otherwise} \end{cases} \end{aligned}$$

with  $\phi' \star \mu' = I(\tau(v).m)(\sigma^\dagger)$  and  $\sigma^\dagger = [\text{this} \mapsto \phi(v), w_1 \mapsto \phi(v_1), \dots, w_n \mapsto \phi(v_n)] \star \mu$ .

The semantics of a command is a partial map from an initial to a final state:  $\mathcal{C}_\tau^I[\![com]\!] : \Sigma_\tau \dashrightarrow \Sigma_\tau$ . We assume that  $\tau$  in both  $\mathcal{E}_\tau^I[\![\_]\!]$  and  $\mathcal{C}_\tau^I[\![\_]\!]$  does not contain the variable `res`.

**Definition 8** (Semantics of commands). *Let  $\tau$  describe the variables in scope,  $I$  be an interpretation and*

$$\begin{aligned} \text{setVar}_\tau^v &= \lambda(\phi \star \mu) \in \Sigma_\tau. \phi|_{-\text{res}}[v \mapsto \phi(\text{res})] \star \mu \\ \text{setField}_\tau^{v.f} &= \lambda(\phi \star \mu) \in \Sigma_\tau. \begin{cases} \phi|_{-\text{res}} \star \mu[l \mapsto \mu(l)[f \mapsto \phi(\text{res})]] & \text{if } \phi(v) = l \neq \text{null} \\ \perp & \text{otherwise.} \end{cases} \end{aligned}$$

The semantics for commands  $\mathcal{C}_\tau^I[\![com]\!] : \Sigma_\tau \dashrightarrow \Sigma_\tau$  is defined as

$$\begin{aligned} \mathcal{C}_\tau^I[\![v := exp]\!] &= \text{setVar}_{\tau+exp}^v \circ \mathcal{E}_\tau^I[\![exp]\!] \\ \mathcal{C}_\tau^I[\![v.f := exp]\!] &= \text{setField}_{\tau+exp}^{v.f} \circ \mathcal{E}_\tau^I[\![exp]\!] \\ \mathcal{C}_\tau^I \left[ \left[ \begin{array}{l} \text{if } v = w \text{ then } com_1 \\ \text{else } com_2 \end{array} \right] \right] (\phi \star \mu) &= \begin{cases} \mathcal{C}_\tau^I[\![com_1]\!](\phi \star \mu) & \text{if } \phi(v) = \phi(w) \\ \mathcal{C}_\tau^I[\![com_2]\!](\phi \star \mu) & \text{if } \phi(v) \neq \phi(w) \end{cases} \\ \mathcal{C}_\tau^I \left[ \left[ \begin{array}{l} \text{if } v = \text{null} \text{ then } com_1 \\ \text{else } com_2 \end{array} \right] \right] (\phi \star \mu) &= \begin{cases} \mathcal{C}_\tau^I[\![com_1]\!](\phi \star \mu) & \text{if } \phi(v) = \text{null} \\ \mathcal{C}_\tau^I[\![com_2]\!](\phi \star \mu) & \text{if } \phi(v) \neq \text{null} \end{cases} \\ \mathcal{C}_\tau^I[\![\{com_1; \dots; com_p\}]\!] &= (\lambda\sigma \in \Sigma_\tau. \sigma) \circ \mathcal{C}_\tau^I[\![com_p]\!] \circ \dots \circ \mathcal{C}_\tau^I[\![com_1]\!]. \end{aligned}$$

The identity function  $\lambda\sigma \in \Sigma_\tau. \sigma$  in the semantics of the sequence of commands is needed when  $p = 0$ .

Each method  $\kappa.m$  is denoted by a partial function from input to output states and an interpretation  $I$  maps methods to partial functions on states, such that  $I(\kappa.m) : \Sigma_{\text{input}(\kappa.m)} \dashrightarrow \Sigma_{\text{output}(\kappa.m)}$ , with the type environments:

$$\begin{aligned} \text{input}(\kappa.m) &= [\text{this} \mapsto \kappa, w_1 \mapsto \kappa_1, \dots, w_n \mapsto \kappa_n] \\ \text{output}(\kappa.m) &= [\text{out} \mapsto \kappa_0, w'_1 \mapsto \kappa_1, \dots, w'_n \mapsto \kappa_n] \text{ and} \\ \text{scope}(\kappa.m) &= \text{input}(\kappa.m) \cup \text{output}(\kappa.m) \\ &\quad \cup [w_{n+1} \mapsto \kappa_{n+1}, \dots, w_{n+m} \mapsto \kappa_{n+m}] \end{aligned}$$

where  $w'_1, \dots, w'_n$  are fresh variables used to keep track of the actual parameters. Each  $w'_i$  is automatically assigned to the same value of the corresponding  $w_i$  at the beginning of the method execution, and it is never changed later.

**Example 9.** Consider the method `makeTree` in Section 1.4. We have that

$$\begin{aligned} \text{input}(\text{Tree.makeTree}) &= [\text{this} \mapsto \text{Tree}, n \mapsto \text{Integer}] \\ \text{output}(\text{Tree.makeTree}) &= [\text{out} \mapsto \text{Tree}, n' \mapsto \text{Integer}] \\ \text{scope}(\text{Tree.makeTree}) &= \text{input}(\text{Tree.makeTree}) \cup \text{output}(\text{Tree.makeTree}) \cup [m \mapsto \text{Integer}]. \end{aligned}$$

The denotational semantics of a program is the least fixpoint of a transformer on interpretations which maps an interpretation  $I$  into a new interpretation  $I'$  evaluating the bodies of the methods in  $I$  from an input state where local variables are bound to `null`. If  $\kappa.m$  is defined as

$$\kappa_0 \ m(w_1:\kappa_1, \dots, w_n:\kappa_n) \text{ with } w_{n+1}:\kappa_{n+1}, \dots, w_{n+m}:\kappa_{n+m} \text{ is } com$$

we have

$$l'(\kappa.m) = (\lambda\phi \star \mu \in \Sigma_{\text{scope}(\kappa.m)} \cdot \phi|_{\text{dom}(\text{output}(\kappa.m))} \star \mu) \circ \mathcal{C}_{\text{scope}(\kappa.m)}^I \llbracket \text{body}(\kappa.m) \rrbracket \circ (\lambda\phi \star \mu \in \Sigma_{\text{input}(\kappa.m)} \cdot \phi[\text{out} \mapsto \text{null}, w'_1 \mapsto \phi(w_1), \dots, w'_n \mapsto \phi(w_n), w_{n+1} \mapsto \text{null}, \dots, w_{n+m} \mapsto \text{null}] \star \mu) .$$

### 3. Reachability, Sharing, Linearity, and Aliasing

We formalize here the concepts of reachability, sharing, linearity, and aliasing for objects. In a later section, we will use these concepts to introduce the new abstract domain ALPS. The following definition will simplify notation later.

**Definition 10** (Fields of locations). *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$ ,  $l \in \text{dom}(\mu)$ ,  $\mathfrak{f}$  an identifier and  $\bar{\mathfrak{f}} = \mathfrak{f}_1, \dots, \mathfrak{f}_n$  a possibly empty sequence of identifiers, when they exist we write*

- $l.\mathfrak{f}$  for  $\mu(l).\phi(\mathfrak{f})$ , the location reachable from  $l$  through the field  $\mathfrak{f}$ ;
- $l.\bar{\mathfrak{f}}$  for  $l.\mathfrak{f}_1 \dots \mathfrak{f}_n$ ; if  $\bar{\mathfrak{f}}$  is empty,  $l.\bar{\mathfrak{f}} = l$ .

Every time we use these notations, we implicitly require that the conditions guaranteeing their existence are satisfied. The following proposition states very simple results relating types and states.

**Proposition 11.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$ ,  $\mathfrak{f}$  an identifier and  $l, l' \in \text{dom}(\mu)$ , then:*

- (1)  $l.\mathfrak{f}$  exists iff  $\tau(l).\mathfrak{f}$  exists;
- (2) if  $l.\mathfrak{f}$  exists and  $l.\mathfrak{f} \neq \text{null}$ , then  $\tau(l.\mathfrak{f}) \leq \tau(l).\mathfrak{f}$ .

These properties essentially derive from the  $\tau$ -correctness of  $\sigma$ . In particular, (2) means that the actual type of the object pointed to by a field is a subtype of the formal type of the field.

**Definition 12** (Sharing and linearity for locations). *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $l_1, l_2 \in \text{dom}(\mu)$ , we say that:*

- (1)  $l_1$  and  $l_2$  share in  $\sigma$  when there are  $\bar{\mathfrak{f}}_1, \bar{\mathfrak{f}}_2$  such that  $l_1.\bar{\mathfrak{f}}_1 = l_2.\bar{\mathfrak{f}}_2 \neq \text{null}$ ;
- (2)  $l_1$  is nonlinear in  $\sigma$  when there are  $\bar{\mathfrak{f}}_1 \neq \bar{\mathfrak{f}}_2$  such that  $l_1.\bar{\mathfrak{f}}_1 = l_1.\bar{\mathfrak{f}}_2 \neq \text{null}$ ; otherwise,  $l_1$  is said to be linear.

Note that, for any location  $l \in \text{dom}(\mu)$ , we have that  $l$  shares with itself.

**Example 13.** Consider the state  $\sigma = \phi \star \mu$  in Example 2. We have that  $l_0$  and  $l_1$  share in  $\sigma$ , since  $l_0.r.r.r = l_1.l.l.l = l_7$ , while  $l_2$  and  $l_4$  do not share. Moreover, all the locations are linear.

If  $l.\mathfrak{f} = l'$  it means that  $l'$  is reachable from  $l$ . If we ignore the field  $\mathfrak{f}$ , we obtain the standard notion of reachability.

**Definition 14** (Reachability relations). *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $l, l' \in \text{dom}(\mu)$ , we write*

- $l \rightarrow_\sigma l'$  iff there is a field  $\mathfrak{f}$  such that  $l' = l.\mathfrak{f}$ ;
- $l \xrightarrow{*}_\sigma l'$  iff there is a sequence of fields  $\bar{\mathfrak{f}}$  such that  $l' = l.\bar{\mathfrak{f}}$ .

Moreover, we denote by  $RLoc_\sigma(l) = \{l' \in \text{dom}(\mu) \mid l \xrightarrow{*}_\sigma l'\}$  the set of locations reachable from  $l \in \text{dom}(\mu)$ . By convention, we assume  $RLoc_\sigma(l) = \emptyset$  for  $l \notin \text{dom}(\mu)$ .

**Example 15.** Consider the state  $\sigma = \phi \star \mu$  in Example 2. We have that:

$$\begin{aligned} RLoc_\sigma(l_0) &= \{l_0, l_2, l_3, l_5, l_7\} & RLoc_\sigma(l_4) &= \{l_4, l_6, l_7\} \\ RLoc_\sigma(l_1) &= \{l_1, l_4, l_6, l_7\} & RLoc_\sigma(l_5) &= \{l_5, l_7\} \\ RLoc_\sigma(l_2) &= \{l_2\} & RLoc_\sigma(l_6) &= \{l_6, l_7\} \\ RLoc_\sigma(l_3) &= \{l_3, l_5, l_7\} & RLoc_\sigma(l_7) &= \{l_7\} \end{aligned}$$

Using reachability we may give an alternative characterization of sharing between locations.

**Proposition 16.** Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and locations  $l_1, l_2 \in \text{dom}(\mu)$ , we have that  $l_1$  shares with  $l_2$  iff  $RLoc_\sigma(l_1) \cap RLoc_\sigma(l_2) \neq \emptyset$ .

An analogous characterization for linearity is not possible, since we need to discern among different ways of reaching the same location.

Using reachability, we refine our definition of interpretation, by requiring that a method does not access locations  $L$  of the input state which are *not* reachable from the actual parameters. Programming languages such as Java and that of Section 2.2 satisfy these constraints. This restriction will let us prove the correctness of method calls in the abstract semantics.

**Definition 17 (Interpretation).** An interpretation  $I$  maps methods to partial functions on state, that is,  $I(\kappa.m) : \Sigma_{\text{input}(\kappa.m)} \dashrightarrow \Sigma_{\text{output}(\kappa.m)}$ , in such a way that if  $I(\kappa.m)(\phi \star \mu) = \phi' \star \mu'$  and  $L = \text{dom}(\mu) \setminus (\bigcup \{RLoc_\sigma(\phi(v)) \mid v \in \text{dom}(\text{input}(\kappa.m))\})$  then  $\mu|_L = \mu'|_L$ ,  $\phi'(\text{out}) \notin L$  and  $\bigcup \{\text{rng}(\mu'(l).\phi) \cap L \mid l \in \text{dom}(\mu'|_{-L})\} = \emptyset$ .

Note that the transformer of interpretations in Section 2.2.2 respects the conditions in Definition 17. Interpretations could be further restricted in such a way that if  $I(\kappa.m)(\phi \star \mu) = \phi' \star \mu'$  then  $\text{rng}(\phi') \cap L = \emptyset$ . We do not enforce this condition since it is not necessary to prove correctness of the abstract semantics.

### 3.1 Reachability among identifiers

As we said before, we want to record sharing and linearity information not only for variables in the type environment but also for their fields. Therefore, we introduce some notation to treat variables and their fields as uniformly as possible.

**Definition 18 (Qualified fields and identifiers).** Given a type environment  $\tau$ , we call qualified field an expression  $v.f$  where  $v \in \text{dom}(\tau)$  and  $f \in \text{dom}(\tau(v))$  and we call qualified identifier either a variable in  $\text{dom}(\tau)$  or a qualified field. We denote by  $Q_\tau$  and  $I_\tau$  the set of qualified fields and identifiers, respectively.

It is worth noting that we only consider fields that are in the declared type of the variables, and we do not consider further fields that are in the actual type. This choice, although it may decrease the precision of the analysis, simplifies a lot the correspondence between abstract and concrete semantics and may increase the speed of the analysis. Note that since  $\text{dom}(\tau)$  is finite, we have that  $I_\tau$  is also finite.

**Example 19.** In Example 2, the qualified fields are  $Q_\tau = \{v_7.l, v_7.r, v_8.l, v_8.r\}$  and the qualified identifiers are  $I_\tau = \{v_7, v_8\} \cup Q_\tau$ .

**Definition 20** (Locations for qualified fields). *If  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $v.f \in Q_\tau$ , for uniformity of notation with variables we define  $\phi(v.f) = \text{null}$  if  $\phi(v) = \text{null}$ ,  $\phi(v.f) = \phi(v).f$  otherwise. In other words, if  $\phi(v.f) \neq \text{null}$ , then  $\phi(v.f)$  is the location pointed to by the field  $f$  in the variable  $v$ .*

The following proposition states that the runtime type of a qualified identifiers is a subtype of its declared type.

**Proposition 21.** *For each  $i \in I_\tau$  and  $\sigma = \phi \star \mu \in \Sigma_\tau$ , we have that  $\phi(i) \neq \text{null}$  implies  $\tau(\phi(i)) \leq \tau(i)$ .*

We now lift the definitions and properties of sharing and linearity from locations to identifiers.

**Definition 22.** (Sharing, linearity, and aliasing). *Let  $\sigma \in \Sigma_\tau$  and  $i_1, i_2 \in I_\tau$ . We say that:*

- $i_1$  and  $i_2$  share in  $\sigma$  when  $\phi(i_1) \neq \text{null} \neq \phi(i_2)$  and  $\phi(i_1), \phi(i_2)$  share in  $\sigma$ ;
- $i_1$  is nonlinear in  $\sigma$  when  $\phi(i_1) \neq \text{null}$  and  $\phi(i_1)$  is nonlinear in  $\sigma$ ; otherwise,  $i_1$  is said to be linear.
- $i_1$  and  $i_2$  are (weakly) aliased in  $\sigma$  when  $\phi(i_1) = \phi(i_2)$ .

Note that two identifiers that are both `null` are considered to be weakly aliased.

**Example 23.** In Figure 10, the field  $v_7.r$  shares with  $v_8.l$ . As a consequence,  $v_7$  shares with  $v_8.l$  and  $v_8$ . The field  $v_7.l$  shares only with itself and the parent. The identifiers  $v_7$  and  $v_8$  are linear, while  $v_5, v_5.r$ , and  $v_6$  in Figure 6 are not linear.

Note that a qualified identifier  $i \in I_\tau$  shares with itself if and only if it is not `null`. Moreover, each  $i \in I_\tau$  such that  $\phi(i) = \text{null}$  is linear and does not share with any other identifier.

**Definition 24** (Reachability for qualified identifiers). *Let  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $i \in I_\tau$ . We define the set of locations reachable from  $i$  in  $\sigma$  as  $RLoc_\sigma(i) = RLoc_\sigma(\phi(i))$  if  $\phi(i) \neq \text{null}$ ,  $\emptyset$  otherwise.*

Note that the reachability set for a variable is related to the reachability set of its qualified fields. This is formalized by the following result.

**Proposition 25.** *Let  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $v \in \text{dom}(\tau)$ . If  $\phi(v) \neq \text{null}$ , then*

$$RLoc_\sigma(v) \supseteq \{\phi(v)\} \cup \bigcup_{v.f \in Q_\tau} RLoc_\sigma(v.f) .$$

Equality does not hold since there is some sharing information in  $RLoc_\sigma(v)$  which is not derivable from the sharing information of its fields. This is due to the fact that we consider only fields in the declared type of a variables. Thus, further sharing relationships may exist in other fields which do not belong to the declared type.

**3.2 Class-induced reachability**

It must be observed that two qualified identifiers might *never* be able to share if their static types do not let them be bound to overlapping data structures. Analogously, certain qualified identifiers are forced to be linear.

**Example 26.** In Example 1, we have that a `Tree` is not an `Integer`, an `Integer` is not a `Tree`, and they do not have any field which can share. Therefore, any identifier of type `Tree` can never share with any identifier of type `Integer`. Moreover, any identifier of type `Integer` may only be linear.

**Example 27.** Consider the following classes:

```
class A { B b; }           class C { }
class B { C c; }           class B1 extends B { }
```

Then, every object of class `A` is linear.

Identifying pair of classes which cannot share, or that are forced to be linear, may improve the result of the analysis. This is the topic of the rest of this section.

**Definition 28** (Class reachability). *We define a reachability relation between classes given by  $\kappa \rightarrow \kappa'$  iff exists an identifier  $f$  such that  $\kappa' \leq \kappa.f$ . We denote by  $\overset{*}{\rightarrow}$  the reflexive and transitive closure of  $\rightarrow$ .*

In Definition 28, if a class  $\kappa'$  (different from  $\kappa$ ) is reachable from  $\kappa$ , then all its subclasses  $\downarrow \kappa'$  are considered reachable. This reflects the fact that we consider a language with (checked) casts. The following proposition relates reachability with class reachability: if location  $l_2$  is reachable from  $l_1$ , the type of  $l_2$  should be reachable from the type of  $l_1$ .

**Proposition 29.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $l_1, l_2 \in \text{dom}(\mu)$ , if  $l_1 \overset{*}{\rightarrow}_\sigma l_2$ , then  $\tau(l_1) \overset{*}{\rightarrow} \tau(l_2)$ .*

This notion of class reachability corresponds to the one in Secci and Spoto (2005a) if we denote by  $C(\kappa)$  the set of classes reachable by  $\downarrow \kappa$ , that is,

$$C(\kappa) = \{ \kappa' \mid \kappa'' \in \downarrow \kappa \text{ and } \kappa'' \overset{*}{\rightarrow} \kappa' \} .$$

We introduce this alternative notation since it will be convenient in the next definitions.

**Example 30.** Consider the classes in Example 1, we have that  $C(\text{Tree}) = \{\text{Tree}\}$  and  $C(\text{Integer}) = \{\text{Integer}\}$ . Therefore, any identifier of type `Tree` can never share with an identifier of type `Integer`. Given the classes in Example 27, we have  $C(A) = \{A, B, B1, C\}$ .

We will denote by *NL* the set of classes whose instances may be nonlinear and by *SH* the set of pair of classes which may share. Both *NL* and *SH* may be computed using class reachability, that is, by typing information only.

First of all, note that identifiers  $i_1$  and  $i_2$  may share only if there is a common location  $l$  which is reachable both from  $\phi(i_1)$  and  $\phi(i_2)$ . Therefore, the class  $\tau(l)$  should be class reachable from both  $\tau(\phi(i_1))$  and  $\tau(\phi(i_2))$ . This is formalized by the following:

**Definition 31** (The *SH* set). *Given the type environment  $\tau$ , we define the set of pairs of classes which may share:*

$$SH = \{ (\kappa, \kappa') \mid C(\kappa) \cap C(\kappa') \neq \emptyset \} .$$

**Proposition 32.** *Given  $i_1, i_2 \in I_\tau$ , and  $\sigma \in \Sigma_\tau$ , if  $i_1$  and  $i_2$  share in  $\sigma$ , then  $(\tau(i_1), \tau(i_2)) \in SH$ .*

We now consider the problem of linearity induced by type information. In general, an object of class  $\kappa$  may be nonlinear either if  $\kappa$  has two fields which may share, it has a field which may share with itself or if a nonlinear class  $\kappa'$  is reachable from  $\kappa$ . This is formalized by the following:

**Definition 33** (The *NL* set). *The set *NL* of nonlinear classes is the upward closure of the least solution of the equation:*

$$\mathcal{S} = \{\kappa \mid C(\kappa.f_1) \cap C(\kappa.f_2) \neq \emptyset, f_1 \neq f_2\} \cup \{\kappa \mid \kappa \in C(\kappa.f)\} \cup \{\kappa \mid \kappa \rightarrow \kappa', \kappa' \in \mathcal{S}\} .$$

Note that *NL* is upward closed by definition. If  $\kappa$  is possibly nonlinear, the same holds for any  $\kappa' \geq \kappa$  since a variable of type  $\kappa'$  may actually point to an object of class  $\kappa$ .

**Proposition 34.** *Given  $\sigma \in \Sigma_\tau$  and  $i \in I_\tau$ , if  $i$  is not linear in  $\sigma$ , then  $\tau(i) \in NL$ .*

In the following sections, we use the concepts of sharing, linearity, and aliasing introduced before to define a new abstract domain, called ALPS (Aliasing Linearity Pair Sharing), for the analysis of Java-like programs.

#### 4. Aliasing Graphs

We start by defining a basic domain encoding definite aliasing. The domain will also encode definite nullness, which is a useful and basic property of Java programs.

**Definition 35** (Pre-aliasing graphs). *A pre-aliasing graph over the type environment  $\tau$  is a directed graph  $G = N \star E \star \ell$  such that:*

- $N$  is the finite set of nodes;
- $E \subseteq N \times Ide \times N$  is the set of directed edges, each labeled by an identifier;
- $\ell : \text{dom}(\tau) \rightarrow N$  is a partial map from variables to nodes;

with the additional condition that

- $\forall n \in N, \forall f \in Ide$ , there is at most an outgoing edge from  $n$  labeled by  $f$  and
- $\forall n \in N, \forall f \in Ide$ , if  $n \xrightarrow{f} n' \in E$  then there exists  $v \in \text{dom}(\tau)$  such that  $\ell(v) = n$  and  $v.f \in Q_\tau$ .

When it is clear from the context, we denote  $G.N, G.E$ , and  $G.\ell$  just by  $N, E$ , and  $\ell$ . Moreover, we denote  $G_i.N, G_i.E$ , and  $G_i.\ell$  by  $N_i, E_i, \ell_i$ , and similarly for other typographical variants of  $G$ .

**Example 36.** Consider three classes  $A, B, C$  where  $B$  extends  $A$  and has two fields  $f$  and  $g$  of class  $B$ . We have that  $\mathcal{K} = \{\top, A, B, C\}$  with  $B \leq A$  and  $B = [f \mapsto B, g \mapsto B]$ , while  $A = C = []$ . Given variables  $a_1, a_2, b_1, b_2, c \in Ide$  with  $\tau = [a_1 \mapsto A, a_2 \mapsto A, b_1 \mapsto B, b_2 \mapsto B, c \mapsto C]$ , Figure 12A shows a pre-aliasing graph over  $\tau$ .

We can extend  $\ell$  from variables to qualified fields. Note that  $\ell(v.f)$  only depends on  $\ell(v)$  and not from  $v$  itself.

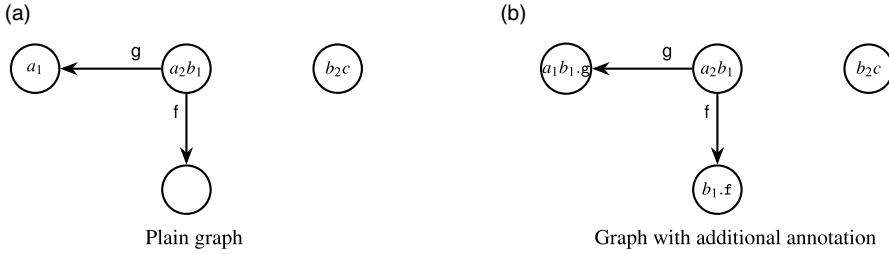


Figure 12. A pre-aliasing graph for the type environment in Example 36.

**Definition 37** (Extension of  $\ell$ ). Given a pre-aliasing graph  $G = N \star E \star \ell$ , we extend  $\ell$  on qualified fields  $v.f \in Q_\tau$  by

$$\ell(v.f) = \begin{cases} n & \text{if } \ell(v) \neq \perp \wedge \ell(v) \xrightarrow{f} n \in E \\ \perp & \text{otherwise} \end{cases}$$

When it helps readability, we will annotate nodes in aliasing graphs both with variables and qualified fields. See, for example, Figure 12B. Moreover, in the examples we will denote a node  $n$  by any identifier  $i$  such that  $\ell(i) = n$ .

The idea of a pre-aliasing graph is that, given an identifier  $i \in I_\tau$ ,  $\ell(i) = \perp$  means  $i$  is definitely null, while  $\ell(i) = \ell(j)$  means that  $i$  and  $j$  are either both null or aliased. Since we aim at designing a domain which encodes definite aliasing and nullness, we define a preorder on pre-aliasing graphs such that  $G_1 \leq G_2$  when  $G_1$  has more aliasing and nullness information than  $G_2$ .

**Definition 38** (Preordering on pre-aliasing graphs). Given two pre-aliasing graphs  $G_1$  and  $G_2$ , we say that  $G_1 \leq G_2$  iff

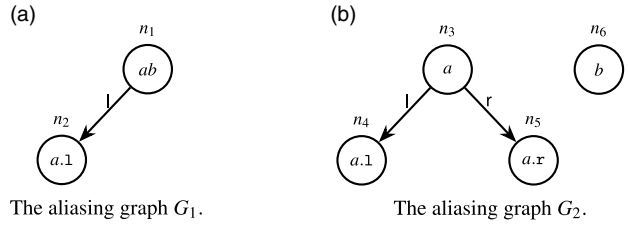
- for each  $i, i' \in I_\tau$ ,  $\ell_2(i) = \ell_2(i') \Rightarrow \ell_1(i) = \ell_1(i')$ ;
- for each  $i \in I_\tau$ ,  $\ell_2(i) = \perp \Rightarrow \ell_1(i) = \perp$ .

Note that, given their intended meaning, some pre-aliasing graphs contain redundant information. For example, nodes which are not labeled by any qualified identifiers may be removed. On the converse, two identifiers  $i_1, i_2$  of incomparable types may be (weak) aliased only if they are both null. We therefore restrict our attention to the pre-aliasing graphs which present some additional regularity conditions.

**Definition 39** (Aliasing graph). An aliasing graph is a pre-aliasing graph  $G$  such that, for all  $n \in N$ ,  $\{\tau(i) \mid i \in I_\tau \wedge \ell(i) = n\}$  is a nonempty chain. We denote by  $\mathcal{G}_\tau$  the set of aliasing graphs over the type environment  $\tau$ , by  $\tau_G(n) = \bigwedge \{\tau(i) \mid i \in I_\tau \wedge \ell(i) = n\}$  the type of the node  $n$  and by  $\psi_G(n) = \bigwedge \{\tau(w) \mid w \in \text{dom}(\tau) \wedge \ell(w) = n\}$  for the type that may be inferred by variables only, with the proviso that the meet of an empty set of classes is  $\top$ .

Note that, although the type  $\tau_G(n)$  depends on all the qualified identifiers labeling that node, the edges possibly departing from the nodes depend on  $\psi_G(n)$  only. We could have also adopted another approach for pre-aliasing graphs and allow edges  $n \xrightarrow{f} m$  if  $f$  is a field in  $\tau_G(n)$ . Although this could potentially improve precision, it comes at the cost of a greater complexity of several operations.





**Figure 13.** Comparison of aliasing graphs. We have explicitly annotated each node with its identity.

**Example 40.** The pre-aliasing graph in Figure 12A is not an aliasing graph due to the rightmost node  $b_2c$  since  $\tau(b_2) = B$ ,  $\tau(c) = C$  but  $\{B, C\}$  is not a chain. This means that the variables  $b_2$  and  $c$  can never be aliased (and thus are both null).

**4.1 Morphisms of aliasing graphs**

We give in this section a different characterization of the preordering over aliasing graphs.

**Definition 41** (Morphism of aliasing graphs). *A morphism of aliasing graphs  $h : G_1 \rightarrow G_2$  in  $\mathcal{G}_\tau$  is a partial map  $h : N_1 \rightarrow N_2$  such that, for each  $i \in I_\tau$ ,  $h(\ell_1(i)) = \ell_2(i)$ .*

This notion of morphism respects the intended meaning of aliasing graphs. If  $h : G_1 \rightarrow G_2$  and  $\ell_1(v) = \perp$ , then  $\ell_2(v) = \perp$ . Moreover, if  $\ell_1(v) = \ell_1(w)$ , then  $\ell_2(v) = \ell_2(w)$ . Aliasing morphisms enjoys many interesting properties. In particular,

**Theorem 42.** *Given two aliasing graphs  $G_1, G_2$ , there exists a morphism from  $G_2$  to  $G_1$  if and only if  $G_1 \preceq G_2$ . Moreover, the morphism, when it exists, is unique.*

It is often easier to think in terms of morphism than to check whether Definition 38 holds: most of the proofs of the properties in this section use graph morphisms and the characterization in Theorem 42. Moreover, morphisms will be pivotal in Section 5 when comparing sharing and linearity information attached to different aliasing graphs.

**Example 43.** Let us consider the following classes:

```
class B; class A extends B { B l; B r; }
```

and the type environment  $\tau = [a \mapsto A, b \mapsto B]$ . Consider the aliasing graphs  $G_1$  and  $G_2$ , respectively, in Figure 13A and B. There is a morphism  $h : G_2 \rightarrow G_1$  given by  $\{n_3 \mapsto n_1, n_6 \mapsto n_1, n_4 \mapsto n_2\}$ , since

$$\begin{aligned} h(\ell_2(a)) &= h(n_3) = n_1 = \ell_1(a) & h(\ell_2(b)) &= h(n_6) = n_1 = \ell_1(b) \\ h(\ell_2(a.l)) &= h(n_4) = n_2 = \ell_1(a.l) & h(\ell_2(a.r)) &= h(n_5) = \perp = \ell_1(a.r) \end{aligned}$$

By Theorem 42, we have that  $G_1 \preceq G_2$ .

**4.2 The lattice of aliasing graphs**

We now show that  $\preceq$  for aliasing graphs has least upper bounds and greatest lower bounds, and we show how to build them.

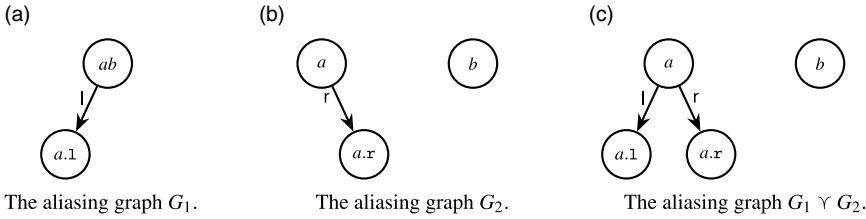


Figure 14. Least upper bound of aliasing graphs.

We begin by defining a new aliasing graph  $G_1 \vee G_2$ , which we will later prove to be the least upper bound. In the definition, we use the inverse function  $\ell^{-1}$ . Note that  $\ell^{-1}$  may be different if we consider  $\ell$  as the map  $\ell : \text{dom}(\tau) \rightarrow N$  in Definition 35 or as the map  $\ell : I_\tau \rightarrow N$  in Definition 37. When it is not specified, the latter is assumed.

**Definition 44.** Given  $G_1, G_2 \in \mathcal{G}_\tau$ , let  $X = \ell_1^{-1}(N_1) \cup \ell_2^{-1}(N_2)$  and let  $\sim \subseteq X \times X$  be the equivalence relation on  $I_\tau$  such that  $i \sim i' \iff \ell_1(i) = \ell_1(i') \wedge \ell_2(i) = \ell_2(i')$ . We define the aliasing graph  $G_1 \vee G_2 = N \star E \star \ell$  where

- $N = X/\sim$  is the set of equivalence classes of  $X$ ;
- for any  $v \in \text{dom}(\tau)$ ,  $\ell(v) = [v]_\sim$  if  $v \in X$ ,  $\ell(v) = \perp$  otherwise;
- $S_1 \xrightarrow{f} S_2 \in E$  iff there exists  $v \in S_1$  s.t.  $v.f \in S_2$ .

**Example 45.** Let us consider the same classes and type environment of Example 43. Figure 14 shows an example of lub of aliasing graphs. Note that, even without knowing the class definitions, the graphs contain enough information to justify the result of the operation. For example, in Figure 14A, the fact that the left child of the node  $ab$  does not contain  $b.l$  means that that  $l$  is not a field of  $\tau(b)$ , hence  $\tau(a)$  is a subclass of  $\tau(b)$ .

An analogous, although more complex, definition may be given for  $G_1 \wedge G_2$ , which we will later prove to be the greatest lower bound of aliasing graphs.

**Definition 46.** Given  $G_1, G_2 \in \mathcal{G}_\tau$ , let  $\sim \subseteq I_\tau \times I_\tau$  be the least equivalence relation on  $I_\tau$  such that  $\ell_1(i) = \ell_1(i') \vee \ell_2(i) = \ell_2(i') \Rightarrow i \sim i'$ . Moreover, let  $N$  be the largest subset of  $I_\tau/\sim$  such that:

- $N \subseteq \{[i]_\sim \mid i \in I_\tau, \tau([i]_\sim) \text{ is a chain and } [i]_\sim \subseteq \ell_1^{-1}(N_1) \cap \ell_2^{-1}(N_2)\}$ ;
- if  $[v]_\sim \notin N$ , then  $[v.f]_\sim \notin N$ .

We define the aliasing graph  $G_1 \wedge G_2 = N \star E \star \ell$  where

- $\ell : \text{dom}(\tau) \rightarrow N$  such that  $\ell(v) = [v]_\sim$  if  $[v]_\sim \in N$ ,  $\ell(v) = \perp$  otherwise;
- $S_1 \xrightarrow{f} S_2 \in E$  iff there exists  $v \in \text{dom}(\tau)$ ,  $v \in S_1$  s.t.  $v.f \in S_2$ .

The definition above is similar to the one for  $\vee$ : we start from defining an equivalence relation  $\sim$  which propagates weak aliasing and we define  $G_1 \wedge G_2$  whose nodes are equivalence classes of identifiers modulo  $\sim$ . However, propagation of nullness is more complex. There are several situations which may force an identifier to be null.

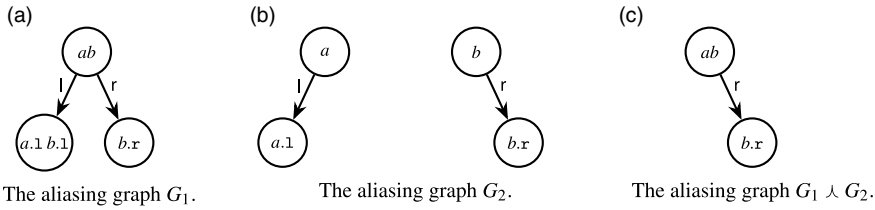


Figure 15. Greatest lower bound of aliasing graphs.

- It may be that  $i_1 \sim i_2$ , but  $i_2$  is null in either  $G_1$  or  $G_2$ . In this case, both  $i_1$  and  $i_2$  are forced to be null. This is the reason of the condition “ $[i]_{\sim} \subseteq \ell_1^{-1}(N_1) \cap \ell_2^{-1}(N_2)$ ” in the first clause of the definition for  $N$ .
- Assume  $\tau(i) = \kappa$ ,  $\tau(i_1) = \kappa_1$ , and  $\tau(i_2) = \kappa_2$ , such that  $\kappa_1 \leq \kappa$ ,  $\kappa_2 \leq \kappa$  with  $\kappa_1$  and  $\kappa_2$  incompatible. It may happen that  $\ell_1(i) = \ell_1(i_1)$  and  $\ell_2(i) = \ell_2(i_2)$ , which implies  $i \sim i_1 \sim i_2$ . However, this forces  $i$ ,  $i_1$ , and  $i_2$  to be null, since there is no object which is both an element of  $\kappa_1$  and  $\kappa_2$ . This is cared by the condition “ $\tau([i]_{\sim})$  is a chain” in the first clause of the definition for  $N$ .
- If variable  $v$  is forced to be null for one of the reasons above, then fields of  $v$  cannot exist. This is cared of by the second clause of the definition for  $N$ .

Example 47. Let us consider the following classes:

```
class A { A l; }    class B extends A { A r; }
```

and the type environment  $\tau = [a \mapsto A, b \mapsto B]$ . Figure 15 shows an example of glb of aliasing graphs.

The following theorem proves that  $\Upsilon$  and  $\wedge$  are actually the least upper bound and greatest lower bound of aliasing graphs.

Theorem 48. The preordered set  $(\mathcal{G}_\tau, \leq)$  has

- a least element  $\perp_\tau = \emptyset \star \emptyset \star \perp$  where  $\perp$  is the always undefined map;
- a greatest element  $\top_\tau = I_\tau \star E \star id$  where  $n_1 \xrightarrow{f} n_2 \in E \iff n_1 = v \in \text{dom}(\tau) \wedge n_2 = v.f \in Q_\tau$ ;
- a least upper bound  $G_1 \Upsilon G_2$  for each  $G_1, G_2 \in \mathcal{G}_\tau$ ;
- a greatest lower bound  $G_1 \wedge G_2$  for each  $G_1, G_2 \in \mathcal{G}_\tau$ .

### 4.3 Projection

Several operations may be defined on aliasing graphs. Most of them will be introduced later, since they depend on the concrete semantics of our language. We introduce here only the operation of restriction of a graph to a subset of variables.

Definition 49 (Projection). Given a pre-aliasing graph  $G$  and a set of nodes  $X$ , we denote by  $G|_X$  the tuple  $X \star E' \star \ell'$  where

$$E' = E \cap (X \times Ide \times X) \quad \ell'(v) = \begin{cases} \ell(v) & \text{if } \ell(v) \in X, \\ \perp & \text{otherwise} \end{cases}$$

It is immediate to check that  $G|_X$  is a pre-aliasing graph. Given a pre-aliasing graph  $G = N \star E \star \ell$ , a set of nodes  $X \subseteq N$  is said to be *backward closed* when, for each  $n \in X$ , if there exists  $n' \xrightarrow{f} n \in E$ , then  $n' \in X$ . Given a set of nodes  $X$ , we denote by  $\overleftarrow{X}$  the smallest backward closed set of nodes containing  $X$ . Symmetrically, we define *forward closed* sets and the forward closure operator  $\overrightarrow{X}$ .

It turns out that if  $X$  is backward closed, then  $G|_X \preceq G$ . Moreover, if  $G$  is an aliasing graph,  $G|_X$  is too. More precisely, the following hold:

**Proposition 50.** *If  $G \in \mathcal{G}_\tau$  and  $X \subseteq N$  is backward closed, then  $G|_X \in \mathcal{G}_\tau$ . Moreover, for each  $n \in X$ ,  $\tau_{G|_X}(n) = \tau_G(n)$  and  $\psi_{G|_X}(n) = \psi_G(n)$ .*

**Proposition 51.** *If  $G$  is a pre-aliasing graph and  $X \subseteq N$  is backward closed, then  $G|_X \preceq G$ .*

We will come back to this point when we introduce the abstract semantics.

#### 4.4 The domain of aliasing graphs

Given a concrete state  $\sigma \in \Sigma_\tau$ , we may abstract it into an aliasing graph which conveys the relevant information.

**Definition 52.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$ , we define the abstraction of  $\sigma$  as an aliasing graph  $\alpha_a(\sigma) = G \in \mathcal{G}_\tau$  where*

- $N = \{l \in \text{Loc} \mid \exists i \in I_\tau. \phi(i) = l\}$ ;
- for each  $v \in \text{dom}(\tau)$ ,  $\ell(v) = \phi(v)$  if  $\phi(v) \neq \text{null}$ ,  $\ell(v) = \perp$  otherwise;
- $l \xrightarrow{f} l' \in E$  iff there exists  $v \in \text{dom}(\tau)$  such that  $\ell(v) = l$ ,  $v.f \in Q_\tau$  and  $l.f = l'$ .

The abstraction of a state  $\sigma$  is essentially the representation of the environment and stores as an aliasing graph, limited to the locations reachable from a qualified identifier.

The following proposition shows that the abstraction of a concrete state is an aliasing graph.

**Proposition 53.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$ ,  $G = \alpha_a(\sigma)$  is an aliasing graph and, for each  $i \in I_\tau$ ,  $\ell(i) = \phi(i)$  if  $\phi(i) \neq \text{null}$ ,  $\ell(i) = \perp$  otherwise.*

We say that  $G \in \mathcal{G}_\tau$  is a correct abstraction of  $\sigma = \phi \star \mu \in \Sigma_\tau$  iff  $\alpha_a(\sigma) \preceq G$ . Note that if  $\alpha_a(\sigma) \preceq G$  and  $G.\ell(i) = \perp$ , then  $\phi(i) = \text{null}$ , hence  $\ell$  may actually be used to represent definite nullness. Moreover, if  $G.\ell(i_1) = G.\ell(i_2)$ , then either  $\phi(i_1) = \phi(i_2) \in \text{Loc}$  or  $\phi(i_1) = \phi(i_2) = \text{null}$ . Hence,  $\ell$  actually encodes definite weak aliasing between variables.

The following propositions show that each aliasing graph can be viewed as the abstraction of a concrete state.

**Proposition 54.** *Given  $G \in \mathcal{G}_\tau$ , there exists  $\sigma \in \Sigma_\tau$  s.t.  $\alpha_a(\sigma)$  and  $G$  are equivalent, that is,  $\alpha_a(\sigma) \sim G$ .*

The map  $\alpha_a$  of Definition 52 may be lifted to the abstraction map of a Galois insertion from  $\mathcal{P}(\Sigma_\tau)$  to  $\mathcal{G}_\tau$  given by  $\alpha_a(S) = \bigvee_{\sigma \in S} \alpha_a(\sigma)$ . The abstraction map induces the concretization map  $\gamma_a : \mathcal{G}_\tau \rightarrow \mathcal{P}(\Sigma_\tau)$ , which maps aliasing graphs to the set of concrete states they represent. Its explicit definition, below, is straightforward:

$$\gamma_a(G) = \{\sigma \in \Sigma_\tau \mid \alpha_a(\sigma) \preceq G\}.$$

**Theorem 55.** *The preorder  $\preceq$  is the same preorder induced by  $\gamma_a$ , that is, given  $G_1, G_2 \in \mathcal{G}_\tau$ ,  $G_1 \preceq G_2$  iff  $\gamma_a(G_1) \subseteq \gamma_a(G_2)$ .*

The above theorem may be considered the analogous of injectivity of  $\gamma_a$  when the abstract domain is preordered instead of partially ordered. It allows to prove that  $\gamma_a(G_1) \subseteq \gamma_a(G_2)$  by just checking  $G_1 \preceq G_2$ .

### 5. ALPS Graphs

Aliasing graphs are a very concrete representation of the part of the program state which is reachable from variables through a single field access. Pair sharing and linearity, instead, summarize global properties of the state. We want to add possible pair sharing and possible nonlinearity information to an aliasing graph.

**Definition 56** (Pre-ALPS graph). *A pre-ALPS graph  $\mathbb{G} = G \star sh \star nl$  is an aliasing graph  $G$  with a set  $sh \subseteq \{\{n, m\} \mid n, m \in N\}$  and a set  $nl \subseteq N$ .*

When it is clear from the context, we denote  $\mathbb{G}, G, \mathbb{G}.sh, \mathbb{G}.nl$  by  $G, sh, nl$  and  $\mathbb{G}_i, G, \mathbb{G}_i.sh, \mathbb{G}_i.nl$  by  $G_i, sh_i, nl_i$ . Similarly for other variants of  $\mathbb{G}$ .

The set  $sh$  in a pre-ALPS graph encodes possible pair sharing, while  $nl$  encodes possible nonlinearity. In particular, two identifiers  $i, j \in I_\tau$  may share when  $\{\ell(i), \ell(j)\} \in sh$ , while  $i$  may be nonlinear when  $\ell(i) \in nl$ . This suggests to extend the preorder on aliasing graphs to ALPS graphs as follows:

**Proposition 57.** *Pre-ALPS graphs are preordered by the relation  $\preceq$  defined as:*

$$\mathbb{G}_1 \preceq \mathbb{G}_2 \iff G_1 \preceq G_2 \text{ and } \forall i \in I_\tau. \ell_1(i) \in nl_1 \Rightarrow \ell_2(i) \in nl_2 \text{ and } \forall i, j \in I_\tau. \{\ell_1(i), \ell_1(j)\} \in sh_1 \Rightarrow \{\ell_2(i), \ell_2(j)\} \in sh_2 .$$

Not all the pre-ALPS graphs make sense, due to the way aliasing, nonlinearity, and sharing interact. In particular, some nonlinearity or sharing information is redundant, since it cannot happen in practice due to the class hierarchy under consideration: pairs  $\{n, m\} \in sh$  such that classes  $\tau_G(n)$  and  $\tau_G(m)$  cannot share, or variables  $n \in nl$  such that  $\tau_G(n) \notin NL$ . This is formalized by the following:

**Definition 58** (Graph compatibility). *Given an aliasing graph  $G \in \mathcal{G}_\tau$ , we say  $\{n, m\} \subseteq N$  is G-SH-compatible if  $(\tau_G(n), \tau_G(m)) \in SH$ . We say  $n \in N$  is G-NL-compatible if  $\tau_G(n) \in NL$ . Sets  $sh \subseteq \mathcal{P}_2(N)$  (and  $nl \subseteq N$ ) are G-SH-compatible (G-NL-compatible) if all their elements are G-SH-compatible (G-NL-compatible).*

We define a reduction operation which takes pre-ALPS graph  $\mathbb{G}$  and removes spurious sharing and linearity information.

**Definition 59.** (Reduced pre-ALPS graphs). *Given a pre-ALPS graph  $\mathbb{G}$ , let  $sh' \star nl'$  be the sharing and nonlinearity information contained in  $sh \star nl$  which is G-SH and G-NL, compatible, that is,*

$$sh' = sh \cap \{\{n, m\} \in \mathcal{P}_2(N) \mid \{n, m\} \text{ is G-SH-compatible}\} ,$$

$$nl' = nl \cap \{n \mid n \text{ is G-NL-compatible}\} .$$

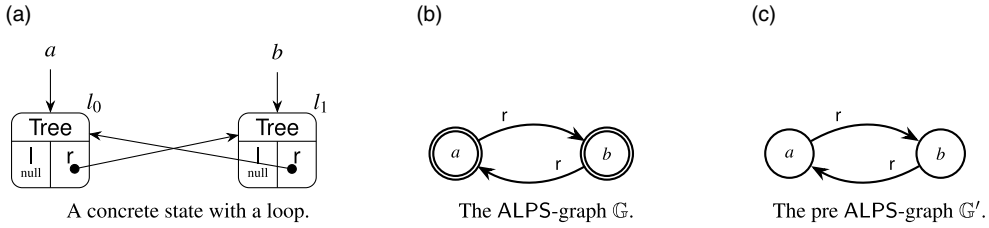


Figure 16. Two pre-ALPS graphs with a loop and a concrete state which is in the concretization of the Pre-ALPS graphs  $\mathbb{G}$  in Figure 16B.

We define the reduction of  $\mathbb{G}$  to be the pre-ALPS graph  $\text{red}(\mathbb{G}) = G \star sh' \star nl'$  and we say that a pre-ALPS graph  $\mathbb{G}$  is reduced if  $\text{red}(\mathbb{G}) = \mathbb{G}$ .

Moreover, some sharing and nonlinearity information can be derived from other information. For example, if  $n$  is a node in  $G$ , then  $\{n\}$  should be in  $sh$ , otherwise any identifier  $i$  s.t.  $\ell(i) = n$  is forced to be null and  $G$  could be simplified by removing the node  $n$ .

**Definition 60.** (Closed pre-ALPS graphs). We define a pre-ALPS graph  $\mathbb{G} = G \star sh \star nl$  closed when it satisfies all the following properties:

- (1)  $n \in N \Rightarrow \{n\} \in sh$ ;
- (2) there is a loop in  $G$  involving  $n \Rightarrow n \in nl$ ;
- (3)  $\{n, m\} \in sh \wedge n' \xrightarrow{f} n \Rightarrow \{n', m\} \in sh$ ;
- (4)  $n \xrightarrow{f_1} m_1, n \xrightarrow{f_2} m_2, f_1 \neq f_2, \{m_1, m_2\} \in sh \Rightarrow n \in nl$ ;
- (5)  $n \in nl \wedge n' \xrightarrow{f} n \Rightarrow n' \in nl$ .

Point (1) is standard in sharing domains since each non-null variable shares with itself. Point (2) expresses the fact that any variable in a loop cannot be linear. Point (5) means that  $nl$  is backward closed, while Point (3) is obvious generalization of backward closure to sharing pairs. Finally, Point (4) formalizes the fact that if two different fields of an object  $o$  share, then  $o$  is not linear.

**Example 61** (Pre-ALPS graphs with loops). Consider the concrete state in Figure 16A. Locations  $l_0$  and  $l_1$  are nonlinear, since  $l_0.r.r = l_0$  and  $l_1.r.r = l_1$ . This state is correctly abstracted by the pre-ALPS graph  $\mathbb{G}$  in Figure 16B, where nodes  $a$  and  $b$  are marked as nonlinear. On the contrary, the same concrete state is not correctly approximated by the pre-ALPS graph  $\mathbb{G}'$  in Figure 16C, since the latter does not allow for the nonlinearity of  $a$  and  $b$ . Actually, the only concrete states which are correctly abstracted by  $\mathbb{G}'$  are the ones where both  $a$  and  $b$  are null, since every other situation would violate either the aliasing constraints ( $a.r = b$  and  $b.r = a$ ) or the linearity constraint. The Pre-ALPS graph  $\mathbb{G}$  is closed according to Definition 60, while  $\mathbb{G}'$  is not and it is essentially not used, since it may be replaced by the empty pre-ALPS graph.

We want to restrict our attention only to those pre-ALPS graphs which do not contain any spurious information and where all sharing and nonlinearity information is explicit.

**Definition 62** (ALPS graph). An ALPS graph  $\mathbb{G}$  is a pre-ALPS graph which is reduced and closed. We denote by  $\text{ALPS}_\tau$  the set of ALPS graphs over the type environment  $\tau$ .

**5.1 Projection**

Analogously to aliasing graphs, we define a projection operator for ALPS graphs.

**Definition 63** (Projection of pre-ALPS graphs). *Given a pre-ALPS graph  $\mathbb{G}$  and  $X \subseteq N$  backward closed, we denote by  $\mathbb{G}|_X$  the pre-ALPS graph  $G|_X \star sh' \star nl'$  where*

$$sh' = sh \cap \mathcal{P}_2(X) \quad nl' = nl \cap X .$$

In the definition above, the hypothesis that  $X$  is backward closed is needed in order to ensure that  $G|_X$  is an aliasing graph and not just a pre-aliasing graph. It is immediate to check that projection maps ALPS graphs to ALPS graphs. More specifically:

**Proposition 64.** *If  $\mathbb{G} \in \text{ALPS}_\tau$  and  $X \subseteq N$  is backward closed, then  $\mathbb{G}|_X \in \text{ALPS}_\tau$ .*

Moreover, the following holds:

**Proposition 65.** *If  $\mathbb{G}$  is a pre-ALPS graph and  $X \subseteq N$  is backward closed, then  $\mathbb{G}|_X \preceq \mathbb{G}$ .*

**5.2 Up- and down-closures of pre-ALPS graphs**

Given a pre-ALPS graph  $\mathbb{G}$ , the up-closure of  $\mathbb{G}$  is a new pre-ALPS graph, obtained by adding derived sharing and nonlinearity information to the  $sh$  and  $nl$  components. Moreover, if  $\mathbb{G}$  is reduced (i.e., it does not contain spurious sharing and nonlinearity elements which are not G-SH-compatible and G-NL-compatible, respectively), then the up-closure of  $\mathbb{G}$  is an ALPS graph.

**Definition 66.** (Up-closure of pre-ALPS graphs). *Given a pre-ALPS graph  $\mathbb{G} = G \star sh \star nl$ , we define the up-closure of  $\mathbb{G}$  as the pre-ALPS graph  $cl^\uparrow(\mathbb{G}) = G \star sh' \star nl'$  such that  $sh' \star nl'$  is the smallest pair, under the component-wise ordering, which contains  $sh \star nl$  and  $G \star sh' \star nl'$  is closed.*

It is immediate to see that the up-closure of a pre-ALPS graph  $\mathbb{G}$  always exists and can be simply computed starting from  $sh \star nl$  and adding new elements according to the five properties in Definition 60. Note that the graph  $G$  does not change when computing the up-closure.

Symmetrically, we can define the down-closure as follows.

**Definition 67.** (Down-closure of pre-ALPS graphs). *Given a pre-ALPS graph  $\mathbb{G}$ , we define the down-closure of  $\mathbb{G}$  as the pre-ALPS graph  $cl^\downarrow(\mathbb{G})$  such that  $cl^\downarrow(\mathbb{G})$  is the greatest pre-ALPS graph smaller than or equal to  $\mathbb{G}$  and such that  $cl^\downarrow(\mathbb{G})$  is closed.*

Note that, differently from the up-closure, when computing the down-closure the graph  $G$  can possibly change, since some nodes could be removed. The next proposition shows how to compute the down-closure.

**Theorem 68.** *Given a pre-ALPS graph  $\mathbb{G} = G \star sh \star nl$ , the down-closure  $cl^\downarrow(\mathbb{G})$  can be computed as follows. Let  $sh^* \star nl^*$  be the greatest pair, under the component-wise ordering, such that*

- (1)  $nl^* = nl \setminus \{n \mid m \notin nl^* \wedge m \xrightarrow{f} n \in E\}$ ;
- (2)  $sh^* = sh \setminus \{\{m_1, m_2\} \mid n \notin nl^*, n \xrightarrow{f_1} m_1 \in E, n \xrightarrow{f_2} m_2 \in E, f_1 \neq f_2\} \setminus \{\{n, m\} \mid \{n', m\} \notin sh^* \wedge n' \xrightarrow{f} n \in E\}$ .

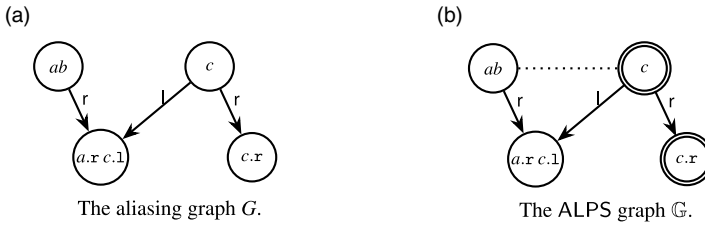


Figure 17. Example of an ALPS graph.

Then, we have that

$$cl^\downarrow(\mathbb{G}) = (G \star sh^* \star nl^*)|_{N \setminus \overrightarrow{X}}$$

where  $X = \{n \mid n \notin nl^*, \text{ there is a loop in } G \text{ such that } n \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E\} \cup \{n \mid \{n\} \notin sh^*\}$ . Moreover, if  $\mathbb{G}$  is closed w.r.t. red, then  $cl^\downarrow(\mathbb{G})$  is an ALPS graph.

The down-closure shows the interaction between the three components of aliasing (which is encoded in the graph structure), sharing, and nonlinearity. It precisely describes how linearity and sharing information propagate to the other components.

The first point states that whenever a node  $m$  is linear, then all its children are linear too. The second point explains the interaction between sharing and linearity: when a node  $n$  is linear, then its children cannot share. Moreover, when a node  $n'$  does not share with a node  $m$ , the same holds for the children of  $n'$ . Note that, whenever in a loop of the graph a node is linear, then all the nodes in the loop and all their children must be null. This is reflected in the projection on  $N \setminus \overrightarrow{X}$ .

**Example 69.** Figure 17 shows the aliasing graph  $G$  and the ALPS graph  $\mathbb{G} = G \star sh \star nl$  where

$$sh = \{\{ab\}, \{c\}, \{a.r\ c.l\}, \{c.r\}, \{ab, a.r\ c.l\}, \{c, a.r\ c.l\}, \{c, c.r\}, \{ab, c\}\}$$

$$nl = \{c, c.r\}$$

In Figure 17B, nonlinearity is represented with a double circle, while sharing information is represented as follows:

- the sharing information of the singletons  $\{ab\}, \{c\}, \{a.r\ c.l\}, \{c.r\}$  can be deduced from the existence of the nodes in the aliasing graph;
- the sharing information of a node with its field, for example,  $\{c, c.r\}$ , can be deduced from the corresponding edge in the aliasing graph  $c \xrightarrow{r} c.r$ ;
- additional sharing information is represented with a dotted line, for example, between the nodes  $ab$  and  $c$ .

### 5.3 The lattice of ALPS graphs

In order to define an abstract domain of ALPS graphs, we start by defining the least upper bound  $\mathbb{G}_1 \vee \mathbb{G}_2$  and greatest lower bound  $\mathbb{G}_1 \wedge \mathbb{G}_2$  of ALPS graphs.

We use morphisms when we need to combine sharing and nonlinearity information coming from different graphs with different sets of nodes.



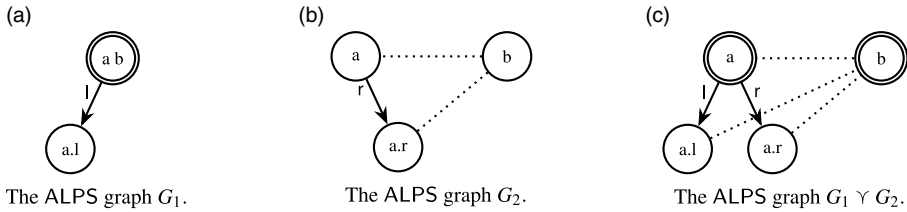


Figure 18. Least upper bound of ALPS graphs.

**Definition 70.** Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be ALPS graphs. We define  $\mathbb{G} = \mathbb{G}_1 \vee \mathbb{G}_2$  as:

- $G = G_1 \vee G_2$ , with morphisms  $h_1 : G \rightarrow G_1$  and  $h_2 : G \rightarrow G_2$ ;
- $sh = h_1^{-1}(sh_1) \cup h_2^{-1}(sh_2)$  and
- $nl = h_1^{-1}(nl_1) \cup h_2^{-1}(nl_2)$ .

**Example 71.** Consider the ALPS graphs in Figure 18A and B. For ease of notation, we assume a node to be denoted by its label. The morphisms  $h_1 : G_1 \vee G_2 \rightarrow G_1$  and  $h_2 : G_1 \vee G_2 \rightarrow G_2$  are defined as follows:

$$h_1 = [a \mapsto ab, b \mapsto ab, a.l \mapsto a.l]$$

$$h_2 = [a \mapsto a, b \mapsto b, a.r \mapsto a.r]$$

We have that:

$$\begin{aligned} sh &= h_1^{-1}(sh_1) \cup h_2^{-1}(sh_2) \\ &= h_1^{-1}(\{\{ab\}, \{a.l\}, \{ab, a.l\}\}) \cup h_2^{-1}(\{\{a\}, \{a.r\}, \{b\}, \{a, a.r\}, \{a, b\}, \{a.r, b\}\}) \\ &= \{\{a\}, \{b\}, \{a.l\}, \{a, b\}, \{a, a.l\}, \{b, a.l\}\} \cup \{\{a\}, \{a.r\}, \{b\}, \{a, a.r\}, \{a, b\}, \{a.r, b\}\} \\ &= \{\{a\}, \{b\}, \{a.l\}, \{a, b\}, \{a.r\}, \{a, a.l\}, \{a, a.r\}, \{b, a.l\}, \{a.r, b\}\} \end{aligned}$$

and

$$nl = h_1^{-1}(nl_1) \cup h_2^{-1}(nl_2) = h_1^{-1}(\{ab\}) \cup h_2^{-1}(\emptyset) = \{a, b\} .$$

**Definition 72.** Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be ALPS graphs. We define  $\mathbb{G} = \mathbb{G}_1 \wedge \mathbb{G}_2$  as  $\mathbb{G} = cl^\downarrow(\mathbb{G}')$ , where

- $G' = G_1 \wedge G_2$  with morphisms  $h_1 : G_1 \rightarrow G$  and  $h_2 : G_2 \rightarrow G$ ;
- $sh' = \{\{n, m\} \in \mathcal{P}_2(N) \mid \forall k \in \{1, 2\} h_k^{-1}(\{\{n, m\}\}) \subseteq sh_k\}$ ;
- $nl' = \{n \in N \mid \forall k \in \{1, 2\} h_k^{-1}(n) \subseteq nl_k\}$ ;

**Example 73.** Consider the ALPS graphs in Figure 19A and B. For ease of notation, we assume a node to be denoted by its label. The morphisms  $h_1 : G_1 \rightarrow G_1 \wedge G_2$  and  $h_2 : G_2 \rightarrow G_1 \wedge G_2$  are defined as follows:

$$h_1 = [ab \mapsto ab, b.r \mapsto b.r]$$

$$h_2 = [a \mapsto ab, b \mapsto ab, b.r \mapsto b.r]$$

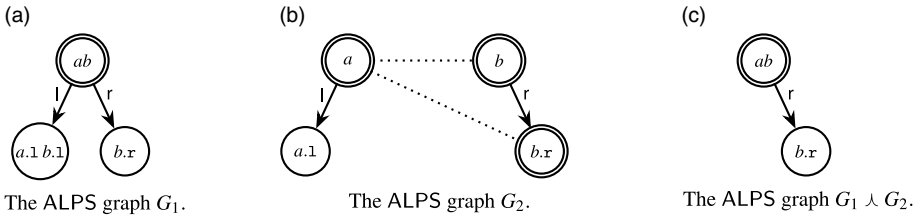


Figure 19. Greatest lower bound of ALPS graphs.

We have that:

$$\begin{aligned}
 sh &= \{ \{n, m\} \in \mathcal{P}_2(N) \mid h_1^{-1}(\{n, m\}) \subseteq sh_1, h_2^{-1}(\{n, m\}) \subseteq sh_2 \} \\
 &= \{ \{n, m\} \in \mathcal{P}_2(\{ab, b.r\}) \mid \\
 &\quad h_1^{-1}(\{n, m\}) \subseteq \{ \{ab\}, \{a.l b.l\}, \{b.r\}, \{ab, a.l b.l\}, \{ab, b.r\} \}, \\
 &\quad h_2^{-1}(\{n, m\}) \subseteq \{ \{a\}, \{b\}, \{a.l\}, \{b.r\}, \{a, a.l\}, \{b, b.r\}, \{a, b\}, \{a, b.r\} \} \} \\
 &= \{ \{ab\}, \{b.r\}, \{ab, b.r\} \}
 \end{aligned}$$

and

$$\begin{aligned}
 nl &= \{ n \in N \mid h_1^{-1}(n) \subseteq nl_1, h_2^{-1}(n) \subseteq nl_2 \} \\
 &= \{ n \in N \mid h_1^{-1}(n) \subseteq \{ab\}, h_2^{-1}(n) \subseteq \{a, b, b.r\} \} \\
 &= \{ab\} .
 \end{aligned}$$

**Theorem 74.** *The preordered set of ALPS graphs has*

- a least element  $\perp_\tau \star \emptyset \star \emptyset$ ;
- a greatest element  $\top_\tau \star sh \star nl$ , where
  - $sh = \{ \{n, m\} \in \mathcal{P}_2(I_\tau) \mid (\tau(n), \tau(m)) \in SH \}$  and
  - $nl = \{ n \in I_\tau \mid \tau(n) \in NL \}$ ;
- a least upper bound  $\mathbb{G}_1 \vee \mathbb{G}_2$  for each pair  $\mathbb{G}_1$  and  $\mathbb{G}_2$  of ALPS graphs;
- a greatest lower bound  $\mathbb{G}_1 \wedge \mathbb{G}_2$  for each pair  $\mathbb{G}_1$  and  $\mathbb{G}_2$  of ALPS graphs.

With an abuse of language, we denote the top and bottom of ALPS graphs for the domain environment  $\tau$  with  $\top_\tau$  and  $\perp_\tau$ , which are the same symbols used for aliasing graphs. We omit the index  $\tau$  when it is clear from or not relevant in the context.

### 5.4 The domain of ALPS graphs

Given a concrete state  $\sigma \in \Sigma_\tau$ , we may abstract it into an aliasing graph which conveys the relevant information.

**Definition 75** (Abstraction map on ALPS graph). *Given  $\sigma \in \Sigma_\tau$ , we define the abstraction  $\alpha : \Sigma_\tau \rightarrow \text{ALPS}$  as  $\alpha(\sigma) = \alpha_a(\sigma) \star sh \star nl$  where*

$$\begin{aligned}
 sh &= \{ \{l_1, l_2\} \subseteq N \mid l_1 \text{ and } l_2 \text{ share in } \sigma \} , \\
 nl &= \{ l \in N \mid l \text{ is not linear in } \sigma \} .
 \end{aligned}$$

We say  $\mathbb{G}$  is a correct abstraction of  $\sigma$  when  $\alpha(\sigma) \preceq \mathbb{G}$ .

The abstraction of a state  $\sigma$  is essentially the representation of the environment and stores as an ALPS graph, limited to the locations reachable from a qualified identifier.

Given  $\sigma \in \Sigma_\tau$  and  $\alpha(\sigma) \preceq \mathbb{G}$ , if  $i_1, i_2 \in I_\tau$  share in  $\sigma$ , then  $\{\mathbb{G}.l(i_1), \mathbb{G}.l(i_2)\} \in \mathbb{G}.sh$ . Moreover, if  $i \in I_\tau$  is nonlinear in  $\sigma$ , then  $\mathbb{G}.l(i) \in \mathbb{G}.nl$ . Hence,  $\mathbb{G}$  actually encodes possible sharing and nonlinearity among variables and fields.

We may define a concretization map  $\gamma : \text{ALPS}_\tau \rightarrow \mathcal{P}(\Sigma_\tau)$  which maps ALPS graphs to the set of concrete states they represent as  $\gamma(\mathbb{G}) = \{\sigma \in \Sigma_\tau \mid \alpha(\sigma) \preceq \mathbb{G}\}$ . If we lift the map  $\alpha$  in Definition 75 to an additive map  $\alpha : \mathcal{P}(\Sigma_\tau) \rightarrow \text{ALPS}_\tau$  as  $\alpha(S) = \bigvee_{\sigma \in S} \alpha(\sigma)$ , then  $\alpha$  and  $\gamma$  form a Galois connection.

**Proposition 76.** (Concretization of ALPS graphs). *The concretization map induced by the abstraction map  $\alpha$  satisfies the following property:*

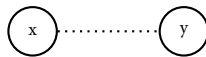
$$\begin{aligned} \gamma(\mathbb{G}) &= \{\sigma \in \Sigma_\tau \mid \sigma \in \gamma_a(\mathbb{G}), \\ &\quad \forall i \in I_\tau. i \text{ nonlinear in } \sigma \Rightarrow l(i) \in nl, \\ &\quad \forall i, i' \in I_\tau. i \text{ share with } i' \text{ in } \sigma \Rightarrow \{l(i), l(i')\} \in sh\} . \end{aligned}$$

Note that ALPS-graphs do not form a Galois insertion with concrete states, as shown in the following example.

**Example 77** (ALPS-graphs do not form a Galois insertion). Consider the following set of classes:

```
class A { C f; }      class B { C g; }      class C { }
```

Since all classes may share among them, we have  $SH = \{(\kappa, \kappa') \mid \kappa, \kappa' \in \{A, B, C\}\}$ . Let  $\tau$  be the type environment  $\tau = \{x \mapsto A, y \mapsto B\}$  and consider the ALPS-graph  $\mathbb{G}$  given by:



It turns out that there is no set of states  $S$  such that  $\alpha(S) = \mathbb{G}$ . This is because, due to the set of classes we have available,  $x$  and  $y$  may only share through the class  $C$ , that is, through the fields  $f$  and  $g$ . But according to  $\mathbb{G}$ ,  $f$  and  $g$  should be null, making sharing impossible. Therefore,  $\alpha$  is not surjective and  $\langle \alpha, \gamma \rangle$  is a Galois connection but not a Galois insertion.

Obtaining a Galois insertion would require the addition of new closure conditions on ALPS graphs. This is possible, but would make the definitions more cumbersome, and we have decided not to follow this line. While Galois insertions are more theoretical appealing since they do not contain redundant abstract elements, precise and efficient analysis can be obtained even without them. In the literature of numerical abstract domains (Amato and Scozzari 2012; Cousot and Cousot 1976) there are many examples of analysis which do not form a Galois insertion and not even a Galois connection, such as the polyhedral analysis of imperative programs (Cousot and Halbwachs 1978) and the parallelotope abstract domain (Amato et al. 2017), which nonetheless enjoy interesting completeness properties (Amato and Scozzari 2011)

In the next section, we will define the necessary operations on the domain of ALPS graphs in order to define an abstract semantics for sharing analysis. Note that, since the abstract domain is finite, we do not need a widening operator to ensure the termination of the analysis.

## 6. An Abstract Semantics on ALPS

We present the abstract semantics on the domain  $ALPS_{\tau}$ . We provide a correct abstract counterpart for each concrete operator in the standard semantics. The abstract counterpart of an interpretation is an ALPS interpretation, defined as follows.

**Definition 78.** An ALPS interpretation  $I$  maps methods to total functions such that  $I(\kappa.m) : ALPS_{input(\kappa.m)} \rightarrow ALPS_{output(\kappa.m)}$  for each method  $\kappa.m$ .

### 6.1 Auxiliary operators

First of all, we introduce some auxiliary operators which will be used later in the abstract semantics for commands and expressions.

#### 6.1.1 Pruning

Given a triple  $\mathbb{G} = G \star sh \star nl$  (not necessarily an ALPS graph), the operation *prune* removes extraneous nodes and adds inferred information:

$$\text{prune}(\mathbb{G}) = \text{cl}^{\uparrow}(\mathbb{G}|_{N'})$$

where  $N' = \{\ell(w) \mid w \in \text{dom}(\tau)\} \cup \{n \mid w \in \text{dom}(\tau), \ell(w) \xrightarrow{f} n \in E, f \in \text{dom}(\tau(w))\}$ .

#### 6.1.2 Restriction

Consider the operation on concrete states which, given  $S \subseteq \Sigma_{\tau}$  and a set of variables  $V \subseteq \text{dom}(\tau)$ , returns the set of states in  $S$  restricted to the variables in  $V$ , that is,

$$S_{\parallel V} = \{\phi|_V \star \mu \mid \phi \star \mu \in S\} \subseteq \Sigma_{\tau|_V} .$$

Starting from a correct abstraction  $\mathbb{G} \in ALPS_{\tau}$  of the set of states  $S$ , we would like to define a new abstraction  $\mathbb{G}' \in ALPS_{\tau|_V}$  which correctly approximates  $S_{\parallel V}$ .

**Definition 79** (Abstract restriction). Given  $\mathbb{G} \in ALPS_{\tau}$  and  $V \subseteq \text{dom}(\tau)$ , we define  $\mathbb{G}_{\parallel V} = \text{prune}(N \star E \star \ell|_V \star sh \star nl)$  .

Let  $W = V \cup \{v.f \in Q_{\tau} \mid v \in V\}$  and let  $\mathbb{G}_{\parallel V} = N' \star E' \star \ell' \star sh' \star nl'$ . Since  $\ell'$  is obtained by restricting  $\ell$  to the variables in  $V$ , we have that for any  $x \in \text{dom}(\tau) \setminus V$ ,  $\ell'(x) = \ell|_V(x) = \perp$ . Then for each  $n' \in N'$ ,  $\ell'^{-1}(n') \subseteq W$  and all the nodes in  $\mathbb{G}$  which are not the image of a qualified identifier in  $W$  are removed from the graph. By construction and since  $\mathbb{G} \in ALPS_{\tau}$ , we have that  $N'$  is backward closed and  $\mathbb{G}_{\parallel V} \in ALPS_{\tau|_V}$ .

**Proposition 80.** For each  $\mathbb{G} \in ALPS_{\tau}$  and  $V \subseteq \text{dom}(\tau)$ ,  $\gamma(\mathbb{G}_{\parallel V}) \subseteq \gamma(\mathbb{G})$ .

#### 6.1.3 Nullness propagation

Consider the operation on concrete states which, given  $S \subseteq \Sigma_{\tau}$  and an identifier  $i \in I_{\tau}$ , returns the subset of those states in  $S$  where  $i$  is `null`, that is,

$$S_{|i=\text{null}} = \{\sigma = \phi \star \mu \in S \mid \phi(i) = \text{null}\} .$$

If  $\mathbb{G}$  is a correct abstraction of  $S$ , it is also a correct abstraction of  $S_{|i=\text{null}}$ , but we would like to refine  $\mathbb{G}$  into a more precise abstract state which still correctly approximates  $S_{|i=\text{null}}$ . This suggests the following definition.

**Definition 81** (Abstract nullness propagation). Given  $\mathbb{G} \in \text{ALPS}_\tau$  and an identifier  $i \in I_\tau$ , we define  $\mathbb{G}_{|i=\text{null}} = \mathbb{G}|_{N_\nu}$  where  $N_\nu = N \setminus \overrightarrow{\{\ell(i)\}}$ .

In  $\mathbb{G}_{|i=\text{null}}$ , since the identifier  $i$  is forced to be `null`, all the nodes reachable from  $i$  need to be removed from the graph since they do not really exist. Given that  $N \setminus \overrightarrow{\{\ell(i)\}}$  is backward closed, we know from Propositions 64 and 65 that  $\mathbb{G}_{|i=\text{null}}$  is an ALPS graph and  $\mathbb{G}_{|i=\text{null}} \preceq \mathbb{G}$ .

Nullness propagation is a special case of greatest lower bound between ALPS graphs, as proved in the following:

**Proposition 82.** For each  $\mathbb{G} \in \text{ALPS}_\tau$  and  $i \in I_\tau$ ,  $\mathbb{G}_{|i=\text{null}} = \mathbb{G} \wedge \top_{|i=\text{null}}$ .

In turn, this allows us to prove that:

**Proposition 83.** For each  $\mathbb{G} \in \text{ALPS}_\tau$  and  $i \in I_\tau$ ,  $\gamma(\mathbb{G})_{|i=\text{null}} \subseteq \gamma(\mathbb{G}_{|i=\text{null}})$ .

In the rest of the paper, given  $i, j \in I_\tau$ , we use  $\mathbb{G}_{|i=\text{null}, j=\text{null}}$  as a short form for  $(\mathbb{G}_{|i=\text{null}})_{|j=\text{null}}$ .

#### 6.1.4 Restriction to aliasing

Consider the operation on concrete states which, given  $S \subseteq \Sigma_\tau$  and two variables  $\nu, w$ , returns the subset of those states in  $S$  where  $\nu$  is weakly aliased with  $w$ , that is,

$$S_{|\nu=w} = \{\sigma = \phi \star \mu \in S \mid \phi(\nu) = \phi(w)\} .$$

Similarly to what we have done in the previous section, we aim to determine a correct approximation of  $S_{|\nu=w}$ . Given  $\mathbb{G} \in \text{ALPS}_\tau$ , we define

$$\mathbb{G}_{|\nu=w} = \mathbb{G} \wedge \top_{\nu=w}$$

where

$$\top_{\nu=w} = \begin{cases} \text{prune}(\top.N \star \top.E \star \top.\ell[w \mapsto \nu] \star \top.sh \star \top.nl) & \text{if } \tau(\nu) \leq \tau(w); \\ \text{prune}(\top.N \star \top.E \star \top.\ell[\nu \mapsto w] \star \top.sh \star \top.nl) & \text{if } \tau(w) < \tau(\nu); \\ \top_{|\nu=\text{null}, w=\text{null}} & \text{otherwise.} \end{cases}$$

**Proposition 84.** For each  $\mathbb{G} \in \text{ALPS}_\tau$  and  $\nu, w \in \text{dom}(\tau)$ ,  $\gamma(\mathbb{G})_{|\nu=w} \subseteq \gamma(\mathbb{G}_{|\nu=w})$ .

In the rest of the paper, we will use  $\mathbb{G}_{|\nu_1=w_1, \dots, \nu_n=w_n}$  as a shorthand for  $((\mathbb{G}_{|\nu_1=w_1}) \dots)_{|\nu_n=w_n}$ .

## 6.2 Abstract semantics for expressions

The abstract semantics specifies how each expression  $exp$  transforms input abstract states  $\mathbb{G} = N \star E \star \ell \star sh \star nl$  into final abstract states  $\mathbb{G}' = N' \star E' \star \ell' \star sh' \star nl'$  where `res` holds the  $exp$ 's value. Abstract semantics for expressions (and later commands) is given compositionally on their syntax.

**Definition 85.** Let  $\tau$  describe the variables in scope and  $I$  be an ALPS interpretation. Figure 20 defines the ALPS semantics for expression (except method calls)  $\mathcal{S}_\tau^I[[exp]] : \text{ALPS}_\tau \rightarrow \text{ALPS}_{\tau+exp}$ .

$$\begin{aligned}
 \mathcal{S}^I_{\tau}[\text{null } \kappa](\mathbb{G}) &= \mathbb{G} \\
 \mathcal{S}^I_{\tau}[\text{new } \kappa](\mathbb{G}) &= N \cup \{n_{\text{new}}\} \star E \star \ell[\text{res} \mapsto n_{\text{new}}] \star sh \cup \{\{n_{\text{new}}\}\} \star nl \\
 &\quad \text{where } n_{\text{new}} \text{ is a new node not in } N \\
 \mathcal{S}^I_{\tau}[v](\mathbb{G}) &= N \star E \star \ell[\text{res} \mapsto \ell(v)] \star sh \star nl \\
 \mathcal{S}^I_{\tau}[(\kappa)v](\mathbb{G}) &= \begin{cases} \mathbb{G} & \text{if } \ell(v) = \perp \\ \mathbb{G}_{|v=\text{null}} & \text{if } \ell(v) \neq \perp \text{ and } \{\tau_G(\ell(v)), \kappa\} \text{ is not a chain} \\ \text{add}(\mathbb{G}, \ell(v), \kappa) & \text{otherwise} \end{cases} \\
 \mathcal{S}^I_{\tau}[v.f](\mathbb{G}) &= \begin{cases} \perp & \text{if } \ell(v) = \perp \\ \mathbb{G} & \text{if } \ell(v) \neq \perp \text{ and } \ell(v.f) = \perp \\ \text{add}(\mathbb{G}, \ell(v.f), \tau(v.f)) & \text{otherwise} \end{cases}
 \end{aligned}$$

Figure 20. The ALPS interpretation for expressions.

We briefly explain the behavior of the abstract semantic operators with respect to the corresponding concrete ones. The concrete semantics of `null κ` stores `null` in the variable `res`. Therefore, in the abstract semantics, we only need to add the new variable `res` into the type environment, without modifying the abstract state.

The concrete semantics of `new κ` stores in `res` a reference to a new object *o*, whose fields are `null`. The other variables do not change. Since *o* is only reachable from `res`, variable `res` shares with itself only and is clearly linear. Therefore, we only need to add a new node labeled with `res` and the corresponding sharing singleton, without affecting nonlinearity information.

The concrete semantics of `v` simply makes `res` an alias for `v`. Since the types of `v` and `res` coincide, we only need to add the variable `v` to the same node of `res`. The other variables are unchanged.

When `v = null`, then `(κ)v` behaves like `null κ`. If `κ` and the type of `v` are not compatible, then `(κ)v` only returns a nonfailed state when `v = null`. Therefore, in the abstract semantics, `(κ)v` restricts the input graph  $\mathbb{G}$  to those states where `v = null`. In the other cases, the cast `(κ)v` stores in `res` the value of `v`. We use an auxiliary operator `add(ℳ, n, κ)`, explained in the following section, which adds the label `res` to the node *n*, and possibly adds new nodes for the fields of `res` which are not fields of  $\psi_G(n)$ . In this case, we can exploit the notion of linearity. In fact, when `v` is linear, we know that fields of `res` cannot share with each other and are linear.

The concrete semantics of `v.f` stores in `res` the value of the field `f` of `v`, provided `v` is not `null`. When `v.f` is not `null`, this essentially amounts to the same procedure of the previous case.

### 6.2.1 Assignment to a node

The auxiliary operator `add(ℳ, n, κ)` adds a new variable `res` of type `κ` as an alias of the node  $n \in N$ . The operator also adds children of `res` when needed to have an ALPS graph.

$$\text{add}(\mathbb{G}, n, \kappa) = \begin{cases} \perp & \text{if } \{\tau_G(n), \kappa\} \text{ is not a chain} \\ \text{cl}^\uparrow(\mathbb{G}') & \text{otherwise} \end{cases}$$

Given  $\mathbb{G} = G \star sh \star nl$ , in order to define  $\mathbb{G}'$ , let  $\kappa' = \psi_G(n)$  be the inferred type for node *n* in  $\mathbb{G}$  (from variables only) and  $F' = \text{dom}(\kappa) \setminus \text{dom}(\kappa') = \{f_1, \dots, f_m\}$  be the set of the new fields in class `κ` which are not currently considered in  $\mathbb{G}$ ; let also  $\{n_{f_1}, \dots, n_{f_m}\}$  be a set of fresh nodes, that

$$\begin{aligned}
 \mathcal{S}\mathcal{C}_\tau^I[v := \text{exp}](\mathbb{G}) &= \text{prune}(N' \star E' \star \ell'[v \mapsto \ell'(\text{res}), \text{res} \mapsto \perp] \star \text{sh}' \star \text{nl}') \\
 &\quad \text{where } \mathbb{G}' = \mathcal{S}\mathcal{C}_\tau^I[\text{exp}](\mathbb{G}). \\
 \mathcal{S}\mathcal{C}_\tau^I[\{\text{com}_1; \dots; \text{com}_p\}] &= (\lambda s \in \text{ALPS}_\tau.s) \circ \mathcal{S}\mathcal{C}_\tau^I[\text{com}_p] \circ \dots \circ \mathcal{S}\mathcal{C}_\tau^I[\text{com}_1] \\
 \mathcal{S}\mathcal{C}_\tau^I \left[ \begin{array}{l} \text{if } v = \text{null} \\ \text{then } \text{com}_1 \\ \text{else } \text{com}_2 \end{array} \right] (\mathbb{G}) &= \begin{cases} \mathcal{S}\mathcal{C}_\tau^I[\text{com}_1](\mathbb{G}) & \text{if } \ell(v) = \perp \\ \mathcal{C}_\tau^I[\text{com}_1](\mathbb{G}_{|v=\text{null}}) \curlywedge \mathcal{S}\mathcal{C}_\tau^I[\text{com}_2](\mathbb{G}) & \text{otherwise} \end{cases} \\
 \mathcal{S}\mathcal{C}_\tau^I \left[ \begin{array}{l} \text{if } v = w \\ \text{then } \text{com}_1 \\ \text{else } \text{com}_2 \end{array} \right] (\mathbb{G}) &= \begin{cases} \mathcal{S}\mathcal{C}_\tau^I[\text{com}_1](\mathbb{G}) & \text{if } \ell(v) = \ell(w) \\ \mathcal{S}\mathcal{C}_\tau^I[\text{com}_1](\mathbb{G}_{|v=w}) \curlywedge \mathcal{S}\mathcal{C}_\tau^I[\text{com}_2](\mathbb{G}) & \text{otherwise} \end{cases}
 \end{aligned}$$

Figure 21. The ALPS interpretation for simple commands.

is, such that  $\{n_{f_1}, \dots, n_{f_m}\} \cap N = \emptyset$ ; then:

$$\begin{aligned}
 G' &= N \cup \{n_{f_1}, \dots, n_{f_m}\} \star E \cup \{n \xrightarrow{f_1} n_{f_1}, \dots, n \xrightarrow{f_m} n_{f_m}\} \star \ell[\text{res} \mapsto n] \\
 \text{sh}' &= \begin{cases} \text{sh} \cup \{\{n_{f_i}, n_{f_j}\} \mid f_i, f_j \in F', (\kappa.f_i, \kappa.f_j) \in SH\} \cup \\ \quad \{\{n_{f_i}, n'\} \mid f_i \in F', \{n, n'\} \in \text{sh}, (\kappa.f_i, \tau_G(n')) \in SH\} & \text{if } n \in \text{nl} \\ \text{sh} \cup \{\{n_{f_i}, n'\} \mid f_i \in F', \{n, n'\} \in \text{sh}, (\kappa.f_i, \tau_G(n')) \in SH, \\ \quad \nexists k > 0, f'_1, \dots, f'_k. n \xrightarrow{f'_1} n_1 \dots n_{k-1} \xrightarrow{f'_k} n' \in E\} & \text{if } n \notin \text{nl} \end{cases} \\
 \text{nl}' &= \begin{cases} \text{nl} \cup \{n_{f_i} \mid f_i \in F', \kappa.f_i \in NL\} & \text{if } n \in \text{nl} \\ \text{nl} & \text{if } n \notin \text{nl} \end{cases}
 \end{aligned}$$

Definition of  $G'$  is straightforward: we just add the new nodes as children of node  $n$ , each one pointed through the corresponding field in  $F'$ . If  $n$  is nonlinear, then all new nodes are nonlinear, all new nodes share among them and with other nodes  $n'$  sharing with  $n$ . This procedure might add spurious sharing or nonlinearity information: we solve this problem by filtering with  $SH$  and  $NL$ .

When  $n$  is linear, then all new nodes are linear, do not share among them, and do not share with any node  $n'$  reachable from  $n$  through a nonempty sequence of fields  $f'_1, \dots, f'_k$  in  $E$ , but share with all the other nodes sharing with  $n$ .

### 6.3 Abstract semantics for commands

In the concrete semantics, each command  $\text{com}$  transforms an initial state into a final state. On the abstract domain, it transforms an initial ALPS graph  $G \star \text{sh} \star \text{nl}$  into an ALPS graph  $G' \star \text{sh}' \star \text{nl}'$ .

**Definition 86.** Let  $\tau$  describe the variables in scope and  $I$  be an ALPS interpretation. Figures 21–23 show the ALPS semantics for commands  $\mathcal{S}\mathcal{C}_\tau^I[\text{com}] : \text{ALPS}_\tau \rightarrow \text{ALPS}_\tau$ .

The concrete semantics of  $v := \text{exp}$  evaluates  $\text{exp}$  and stores its result into  $v$ . Thus, the final abstract state is obtained by first computing  $\mathcal{S}\mathcal{C}_\tau^I[\text{exp}]$  and then renaming  $\text{res}$  into  $v$ . Some of the nodes may become unlabeled and must be removed. This is accomplished by the auxiliary operation  $\text{prune}$  which removes unnecessary information, in particular unlabeled nodes and fields which are not in the declared type of the variables.

Given command  $v.f := exp$  and  $\mathbb{G}' = \mathcal{S}\mathcal{E}_\tau^I[[exp]](\mathbb{G}) = N' \star E' \star \ell' \star sh' \star nl'$ , we define

$$\begin{aligned}
 V_{comp} &= \{x \in \text{dom}(\tau) \mid \ell'(x) \neq \ell'(v), \{\ell'(x), \ell'(v)\} \in sh', \{\tau_{G'}(\ell'(x)), \tau_{G'}(\ell'(v))\} \\
 &\quad \text{is a chain}\} \\
 N_{new} &= \{n_{\ell'(x)} \mid x \in V_{comp}, \mathbf{f} \in \text{dom}(\psi_{G'}(\ell'(x))), \ell'(x.f) \neq \ell'(\text{res})\} \\
 &\quad \text{a set of new distinct nodes} \\
 E_{del} &= \{\ell'(v) \xrightarrow{\mathbf{f}} \ell'(v.f)\} \cup \{\ell'(x) \xrightarrow{\mathbf{f}} \ell'(x.f) \in E' \mid x \in V_{comp}, \ell'(x.f) \neq \ell'(\text{res})\} \\
 E_{new} &= \{\ell'(x) \xrightarrow{\mathbf{f}} n_{\ell'(x)} \mid x \in V_{comp}, n_{\ell'(x)} \in N_{new}\} \\
 sh_{new} &= \{\{n_{\ell'(x)}, n'\} \mid n_{\ell'(x)} \in N_{new}, \{\ell'(x.f), n'\} \in sh'\} \cup \\
 &\quad \{\{n_{\ell'(x)}, n_{\ell'(y)}\} \mid n_{\ell'(x)}, n_{\ell'(y)} \in N_{new}, \{\ell'(x.f), \ell'(y.f)\} \in sh\} \\
 E'_{new} &= E_{new} \cup \{\ell'(v) \xrightarrow{\mathbf{f}} \ell'(\text{res})\} \\
 sh'_{new} &= \{\{n, n'\} \mid \{\ell'(v), n\} \in sh' \text{ and } \{\ell'(\text{res}), n'\} \in sh'\} \cup \\
 &\quad \{\{n_{\ell'(x)}, n'\} \mid n_{\ell'(x)} \in N_{new}, \{\ell'(\text{res}), n'\} \in sh'\} \cup \\
 &\quad \{\{n_{\ell'(x)}, n'\} \mid n_{\ell'(x)} \in N_{new}, \{\ell'(x.f), n'\} \in sh'\} \cup \\
 &\quad \{\{n_{\ell'(x)}, n_{\ell'(y)}\} \mid n_{\ell'(x)}, n_{\ell'(y)} \in N_{new}\} \\
 nl'_{new} &= \begin{cases} \{n \in N' \mid \{n, \ell'(v)\} \in sh'\} \cup \{n_{\ell'(x)} \mid n_{\ell'(x)} \in N_{new}\} & \text{if } \{\ell'(\text{res}), \ell'(v)\} \in sh' \text{ or } \ell'(\text{res}) \in nl' \\ \{n \in N' \mid \{n, \ell'(v)\} \in sh', \{\ell'(\text{res}), n\} \in sh'\} & \\ \text{otherwise} & \end{cases}
 \end{aligned}$$

and, finally,

$$\mathcal{S}\mathcal{E}_\tau^I[[v.f := exp]](\mathbb{G}) = \begin{cases} \perp & \text{if } \ell'(v) = \perp \\ \text{prune}(N' \cup N_{new} \star E' \setminus E_{del} \cup E_{new} \star \\ \ell' \star sh' \cup sh_{new} \star \\ nl' \cup \{n_{\ell'(x)} \mid n_{\ell'(x)} \in N_{new}, \ell'(x.f) \in nl'\}) & \text{if } \ell'(v) \neq \perp \text{ and } \ell'(\text{res}) = \perp \\ \text{prune}(N' \cup N_{new} \star E' \setminus E_{del} \cup E'_{new} \star \\ \ell'[\text{res} \mapsto \perp] \star sh' \cup sh'_{new} \star \\ nl' \cup nl'_{new} \cup \{n_{\ell'(x)} \mid n_{\ell'(x)} \in N_{new}, \ell'(x.f) \in nl'\}) & \\ \text{otherwise} & \end{cases}$$

Figure 22. The ALPS interpretation for assignment to field.

To determine a correct approximation of the conditional “if  $v = \text{null}$ ” we check whether  $\ell(v) = \perp$ . If this is the case, then we know that  $v$  is null and we evaluate  $com_1$ . Otherwise, we evaluate both branches and compute the lub. When evaluating the first branch, we may improve precision by using the auxiliary operator  $\mathbb{G}_{|v=\text{null}}$  (Section 6.1.3) which returns a correct approximation of the program states  $\{\phi \star \mu \mid \alpha(\phi \star \mu) \leq \mathbb{G} \wedge \phi(v) = \text{null}\}$ , that is, the states correctly approximated by  $\mathbb{G}$  where  $v$  is null. Note that, since our domain does not model definite non-nullness, there is no way to define a projection  $\mathbb{G}_{|v \neq \text{null}}$  to improve the input for  $\mathcal{S}\mathcal{E}_\tau^I[[com_2]]$  as in  $\mathcal{S}\mathcal{E}_\tau^I[[com_1]](\mathbb{G}_{|v=\text{null}})$ .



$$\mathcal{S}_{\tau}^{\ell} \llbracket v.m(v_1, \dots, v_n) \rrbracket (\mathbb{G}) = \begin{cases} \perp & \text{if } \ell(v) = \perp \\ \bigvee \{ \text{match}_{v.m}(\mathbb{G}, I(\kappa.m)(\mathbb{G}')) \mid \kappa \leq \tau_{\mathbb{G}}(\ell(v)) \} & \text{otherwise} \end{cases}$$

where  $\ell^{input} = [\text{this} \mapsto \ell(v), w_1 \mapsto \ell(v_1), \dots, w_n \mapsto \ell(v_n)]$  and  $\mathbb{G}' = \text{prune}(N \star E \star \ell^{input} \star sh \star nl)$ .

**Figure 23.** The ALPS interpretation for method calls. The auxiliary function  $\text{match}_{v.m}$  is defined later in Figure 24.

Similarly for the conditional “if  $v = w$ ” where we use another auxiliary operator  $\mathbb{G}_{|v=w}$  (Section 6.1.4) which returns a correct approximation of the set of program states  $\{\phi \star \mu \mid \alpha(\phi \star \mu) \leq \mathbb{G} \wedge \phi(v) = \phi(w)\}$ . Again, since our domain does not model definite not aliasing, we cannot improve the input for the second branch.

The composition of commands is denoted by functional composition over ALPS, where the identity map  $\lambda s \in \text{ALPS}_{\tau.s}$  is needed when  $p = 0$ . The evaluation of  $v.f := exp$  in Figure 22 is the most complex operation of the abstract semantics. Although it may seem similar to the assignment  $v := exp$ , we must take into account that  $v$  might be aliased to many different nodes. The candidates are those variables, denoted by  $V_{comp}$ , which share with  $\ell(v)$  and have compatible types. For each node labeled by a variable in  $V_{comp}$ , we add a new fresh node in  $N_{new}$  pointed by an edge (labeled by the field  $f$ ) in  $E_{new}$ . Finally, all possible sharing and nonlinearity are added. A slightly different treatment is devoted to the special case when the result of the expression is definitely null.

### 6.4 Abstract semantics of method call

The concrete semantics of the method call  $v.m(v_1, \dots, v_n)$  builds an input state containing the local variables  $w_1, \dots, w_n$  and the special variable *this*, and executes the method body. In order to improve the precision of methods call, we also use a copy of the local variables  $w'_1, \dots, w'_n$  which will be used when returning from the method call to match the original variables  $v_1, \dots, v_n$ . This allows a change to the object pointed by  $w_i$  to be distinguished from a change to  $w_i$  itself.

When a method  $v.m$  is called, the class of  $v$  is inspected and the correct overloaded method for  $m$  is selected. The abstract domain contains only a partial information on the runtime class of  $v$ , since we only know that the class of  $v$  must be a superclass of the class of any variable aliased with  $v$ , namely a superclass of  $\tau_{\mathbb{G}}(\ell(v))$ . We exploit this information in computing the abstract semantics of a method call in Figure 23. In practice, we conservatively assume that every method  $m$  in any subclass of  $\tau_{\mathbb{G}}(\ell(v))$  may be called. Note that methods defined only in superclasses of  $\kappa$  are already considered in  $\kappa$ .

When exiting from a method call, we need to rename out into *res* since, from the point of view of the caller, the returned value of the callee (out) is the value of the method call expression (*res*). We use an auxiliary operation  $\text{match}_{v.m}$  which, given an initial and final state, updates the initial state trying to guess a possible matching of variables in the abstract states. The definition of  $\text{match}_{v.m}$  is given in Figure 24.

**Theorem 87.** *The abstract semantics formalized in Figures 20–24 is correct wrt the concrete semantics in Section 2.2.2.*

Analogously to the concrete case, we may define an abstract transformer which, given a ALPS interpretation  $I$ , returns a new ALPS interpretation  $I'$  such that

We define  $\text{match}_{v,m}(\mathbb{G}^1, \mathbb{G}^2) = \text{cl}^\uparrow(\mathbb{G})$  as follows:

$$\begin{aligned}
 V_m &= \{v, v_1, \dots, v_n, \text{res}\} \\
 V_{alias} &= \{x \in \text{dom}(\tau) \mid \exists u \in V_m. \ell^1(x) = \ell^1(u)\} \setminus V_m \\
 V_{comp} &= \{x \in \text{dom}(\tau) \mid \{\ell^1(x), \ell^1(u)\} \in sh^1, u \in V_m\} \setminus (V_m \cup V_{alias}) \\
 V_{other} &= \text{dom}(\tau) \setminus (V_m \cup V_{alias} \cup V_{comp}) \\
 \mathbb{G}_m &= \text{prune}(N^2 \star E^2 \star [v \mapsto \ell^2(\text{this}), v_1 \mapsto \ell^2(w'_1), \dots, v_n \mapsto \ell^2(w'_n), \text{res} \mapsto \ell^2(\text{out})] \star sh^2 \star nl^2) \\
 \mathbb{G}_{comp} &= (\top_{\parallel V_{comp}})_{|\{x=y \mid \ell^1(x)=\ell^1(y), x, y \in V_{comp}\}} \\
 \mathbb{G}_{other} &= \mathbb{G}_{\parallel V_{other}}^1 \\
 sh &= sh_m \cup sh_{comp} \cup sh_{other} \cup \{\{n_1, n_2\} \mid n_1 \in N_{comp}, n_2 \in N_m\} \\
 nl &= nl_{other} \cup nl_{comp} \cup nl_m \\
 \ell_{alias} &= [y \mapsto \ell_m(x) \mid x \in V_{alias}, y \in V_m, \ell^1(x) = \ell^1(y)] \\
 \mathbb{G} &= (N_{other} \cup N_{comp} \cup N_m) \star (E_{other} \cup E_{comp} \cup E_m) \star (\ell_{other} \cup \ell_{comp} \cup \ell_m \cup \ell_{alias}) \star sh \star nl
 \end{aligned}$$

Figure 24. The  $\text{match}_{v,m}$  auxiliary operation.

$$\begin{aligned}
 I'(\kappa.m) &= (\lambda \mathbb{G} \in \text{ALPS}_{\text{scope}(\kappa.m)}. \mathbb{G}_{\parallel \text{dom}(\text{output}(\kappa.m))}) \circ \\
 &\quad \mathcal{S}\mathcal{E}^1_{\text{scope}(\kappa.m)} \llbracket \text{body}(\kappa.m) \rrbracket \circ \\
 &\quad (\lambda \mathbb{G} \in \text{ALPS}_{\text{input}(\kappa.m)}. N \star E \star \ell' \star sh \star nl)
 \end{aligned}$$

where  $\ell' = \ell[w'_1 \mapsto \ell(w_1), \dots, w'_n \mapsto \ell(w_n)]$ .

The new interpretation returned by the abstract transformer is computed by first adding primed variables which are used to hold a copy of the original actual parameters, then evaluating the body of the method and finally restricting the graph to the output variables. The abstract denotational semantics is the least fixpoint of this transformer.

**Theorem 88.** *The abstract denotational semantics is correct wrt the concrete one.*

**Example 89.** Consider the method `Tree.makeTree` in Section 1.4, where

$$\text{scope}(\text{Tree.makeTree}) = [\text{this} \mapsto \text{Tree}, n \mapsto \text{Integer}, n' \mapsto \text{Integer}, m \mapsto \text{Integer}, \text{out} \mapsto \text{Tree}].$$

We can compute a new ALPS interpretation from the least informative ALPS interpretation  $I_\perp(\text{Tree.makeTree}) = \lambda \mathbb{G}. \perp_{\text{out}(\text{Tree.makeTree})}$ :

$$I^1(\text{Tree.makeTree})(\mathbb{G}) = N \cup \{n_{out}\} \star E \star \ell[n' \mapsto \ell(n), \text{out} \mapsto n_{out}] \star sh \cup \{\{n_{out}\}\} \star nl$$

Now, starting from  $I^1(\text{Tree.makeTree})$ , we can compute a new interpretation as follows:

$$\begin{aligned}
 I^2(\text{Tree.makeTree})(\mathbb{G}) &= N \cup \{n_{out}, n_{out.l}, n_{out.r}\} \star \\
 &\quad E \cup \{n_{out} \xrightarrow{l} n_{out.l}, n_{out} \xrightarrow{r} n_{out.r}\} \star \\
 &\quad \ell[n' \mapsto \ell(n), \text{out} \mapsto n_{out}] \star \\
 &\quad sh \cup \{\{n_{out}\}, \{n_{out.l}\}, \{n_{out.r}\}, \{n_{out}, n_{out.l}\}, \{n_{out}, n_{out.r}\}\} \star nl
 \end{aligned}$$

which is the least fixpoint. Relatively to the case  $\ell(n) \neq \text{null}$ , the abstract states  $I^1(\text{Tree.makeTree})(\mathbb{G})$  and  $I^2(\text{Tree.makeTree})(\mathbb{G})$  are depicted in Figure 25. From this graph, it appears that the tree generated by this method is linear and does not share with `this` (the object on which we call the `makeTree` method). Moreover since  $n$  and  $n'$  are aliased we know that the method does not modify the variable  $n$ .

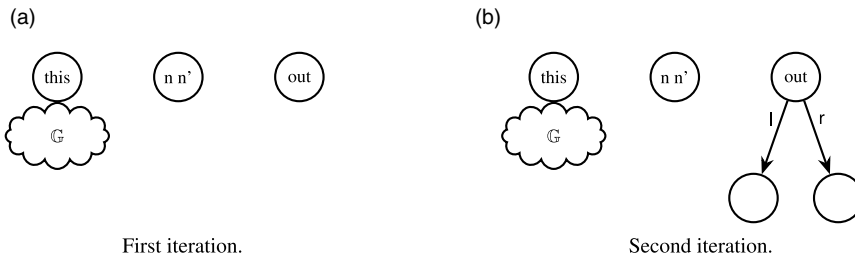


Figure 25. ALPS interpretations for the `makeTree` method.

## 7. Related Work

Sharing properties has been deeply studied for logic programs, see for instance: Jacobs and Langen (1992), Hans and Winkler (1992), Muthukumar and Hermenegildo (1992), Codish *et al.* (1999), Bagnara *et al.* (2002), and Amato and Scozzari (2009). The large literature on this topic has been the starting point for designing our enhanced abstract domain for sharing analysis. In particular, the use of a linearity property, for example, Codish *et al.* (1991), Hans and Winkler (1992), Muthukumar and Hermenegildo (1992), King (1994), and Amato and Scozzari (2010, 2014), has proved to be very useful when dealing with sharing information (see Bagnara *et al.* 2005 for a comparative evaluation).

Outside of the logic programming community, sharing information is generally regarded as a by-product of shape analysis. One of the first papers that explicitly deals with sharing information is Jones and Muchnick (1979), which presents a combined intra-procedural analysis of aliasing, reachability, and cyclicity for imperative and functional languages with records. The focus of their analysis is not pair sharing, but detecting the set of *shared nodes*, that is, heap cells which may be reached by variables using at least two different paths. Shared nodes and cyclicity are used to optimize memory management.

The property of sharing for object-oriented languages has been studied in a few works. Secci and Spoto (2005a) propose a simple domain of pair sharing for a simple Java-like language and Méndez-Lojo and Hermenegildo (2008) extend this domain proposing a combined analysis of set sharing, nullness, and classes for exactly the same language. The main differences of our paper w.r.t. these proposals are

- the ALPS abstract domain encodes linearity and aliasing information, in addition to sharing;
- the analysis is field sensitive, that is, information is encoded at the level of the fields of the objects.

The analysis in Secci and Spoto (2005a) has also been refined by Zanardini (2018) which proposes a field-sensitive sharing analysis for a very similar language. Linearity information is not exploited in this paper either.

Pollet *et al.* (2001) propose a framework for the analysis of object-oriented languages and introduce two abstract domains for definite aliasing and possible aliasing, respectively. The abstract objects of these domains are similar to aliasing graphs, but without being restricted to two levels. Termination is guaranteed by widening. These domains may be enriched by providing type information for the leaves of the graphs. However, they do not consider sharing or linearity properties.

In the context of pure functional programming with no destructive updates, sharing is just an implementation detail and has not impact on the behavior of the program. Nonetheless, sharing analysis may help in optimizing program execution. For example, the first part of Jones and Muchnick (1979) deals with sharing in pure functional programs in order to avoid the use of garbage collection and reference counting.

More recently, Peña-Marí et al. (2006) present a pair-sharing analysis for SAFE, which is a functional language with explicit control for copy and destruction of data structures. The result of the analysis is used during type checking. Due to the particular features of the programming language, the sharing analysis distinguishes between generic sharing and sharing of recursive substructures (e.g., two lists share a recursive substructure when they have a common tail). The analysis is field insensitive and cannot represent neither definite aliasing nor definite nullness and linearity. This analysis has been vastly improved in Montenegro et al. (2015), where each pair of sharing variables is annotated with the set of possible paths through which the sharing may happen. Although definite aliasing and definite nullness are still not representable in the new domain, definite linearity is inferable by the extended pair-sharing information: if  $x$  shares with itself only through the empty path, then  $x$  is linear.

## 8. Conclusions

We propose the new abstract domain ALPS which combines aliasing, linearity, and sharing analysis for an object-oriented language and provide all the necessary abstract operations.

The combination with linearity information allows us to improve the precision of the analysis in blocks of assignments, method calls, and thus on recursion. This is a fundamental issue that has not been considered in any previous analysis. We have shown in Section 1.4 a simple example where linearity plays a fundamental role in proving that two subtrees do not share. More generally, an important point is that linearity information allows us to distinguish a tree from a direct acyclic graph (DAG). For instance, the result of `makeTree` in Figure 8 is a tree. However, the abstract representation of a tree in any abstract domain containing only information about reachability, sharing, acyclicity, nullness, and aliasing (and the corresponding negated properties such as cyclicity, non-nullness, etc.) cannot be distinguished from the abstraction of a DAG. For example, the data structure `t2`:

```
Tree t2 = new Tree()
t2.l = new Tree()
t2.r = new Tree()
t2.l.l = new Tree()
t2.l.l.r = t2.l.l
```

has the same sharing, acyclicity, nullness, etc. properties of a tree, but `t2` is not linear, while `t` in Figure 8 is linear. At the end of the method `useTree`, `left` is equal to `t.l.l`, `right` is `t.l.r`, but they do not share since `t` is a tree. However, `t2.l.l` and `t2.l.r` share. Linearity allows to distinguish these two situations. None of the works discussed in Section 7 can distinguish a DAG from a tree, with the exception of Montenegro et al. (2015) which cannot be directly applied to a Java-like language since it does not support updates.

Regarding modularity of the analysis, although the most precise results are obtained by analyzing the entire program as a whole, it is possible to analyze single libraries (or even single methods) by assuming the every external method returns the largest possible abstract object. In this way, when a real object will be plugged in the final program, its behavior will be correctly abstracted.

From the point of view of performance, note that in a domain which tracks possible sharing information, it usually happens that the more precise an abstract object is, the smaller its representation is. For example, the best correct abstraction of a concrete state where no sharing happens is an ALPS graph without edges. However, if our analysis is not precise enough to compute the best correct abstraction, some edges will be included in the abstract object, which may negatively impact the performance of succeeding operations. Therefore, improving the precision of the analysis, from the one side, increases the computational cost, but from the other side may lead to smaller abstract objects which partially compensates this cost. A detailed evaluation of the trade-off between precision and performance may only be conducted experimentally. We plan to carry

on this evaluation once we implement ALPS as an abstract domain for the Jandom static analyzer (Amato *et al.* 2013).

Although we have presented ALPS graphs in the context of object-oriented programs, the same domain can be immediately applied to functional programs. In this regard, note that the example program in Figure 8 may be rewritten in functional style and the benefits of linearity are the same already discussed for object-oriented programs.

The domain of ALPS graphs may be easily extended by annotating nodes with additional information in a modular way. For example, class analysis might be integrated by adding a set of possible classes for each node, while nodes representing numerical entities may be annotated with intervals for integrating range analysis. Obviously, if we want to maximize the benefits of the integration, at the very least, new operators should be devised which use the information of one domain to improve the precision of the other.

To the best of our knowledge, this is the first attempt to combine sharing with linearity for imperative or object-oriented languages.

**Conflicts of Interests.** The authors declare none.

## References

- Amato, G., Di Nardo Di Maio, S. and Scozzari, F. (2013). Numerical static analysis with Soot. In: *Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis, SOAP'13*, New York, NY, USA. ACM.
- Amato, G., Meo, M. C. and Scozzari, F. (2015). Exploiting linearity in sharing analysis of object-oriented programs. In Crescenzi, P. and Loreti, M. (eds.) *Proceedings of ICTCS 2015, the 16th Italian Conference on Theoretical Computer Science*, Electronic Notes in Theoretical Computer Science, vol. 322, Elsevier, 3–18.
- Amato, G., Rubino, M. and Scozzari, F. (2017). Inferring linear invariants with parallelotopes. *Science of Computer Programming* **148** 161–188.
- Amato, G. and Scozzari, F. (2009). Optimality in goal-dependent analysis of sharing. *Theory and Practice of Logic Programming* **9** (5) 617–689.
- Amato, G. and Scozzari, F. (2010). On the interaction between sharing and linearity. *Theory and Practice of Logic Programming* **10** (1) 49–112.
- Amato, G. and Scozzari, F. (2011). Observational completeness on abstract interpretation. *Fundamenta Informaticae* **106** (2–4) 149–173.
- Amato, G. and Scozzari, F. (2012). Random: R-based analyzer for numerical domains. In: Bjørner, N. and Voronkov, A. (eds.) *Logic for Programming, Artificial Intelligence, and Reasoning 18th International Conference, LPAR-18, Mérida, Venezuela, March 11–15, 2012. Proceedings*, Lecture Notes in Computer Science, vol. 7180, Berlin, Heidelberg, Springer, 375–382.
- Amato, G. and Scozzari, F. (2014). Optimal multibinding unification for sharing and linearity analysis. *Theory and Practice of Logic Programming* **14** 379–400.
- Bagnara, R., Hill, P. M. and Zaffanella, E. (2002). Set-sharing is redundant for pair-sharing. *Theoretical Computer Science* **277** (1–2) 3–46.
- Bagnara, R., Zaffanella, E. and Hill, P. M. (2005). Enhanced sharing analysis techniques: A comprehensive evaluation. *Theory and Practice of Logic Programming* **5** (1–2) 1–43.
- Codish, M., Dams, D. and Yardeni, E. (1991). Derivation and safety of an abstract unification algorithm for groundness and aliasing analysis. In Furukawa, K. (ed.) *Logic Programming, Proceedings of the Eighth International Conference*, Logic Programming, Cambridge, MA, USA, The MIT Press, 79–93.
- Codish, M., Søndergaard, H. and Stuckey, P. J. (1999). Sharing and groundness dependencies in logic programs. *ACM Transactions on Programming Languages and Systems* **21** (5) 948–976.
- Cousot, P. and Cousot, R. (1976). Static determination of dynamic properties of programs. In: *Proceedings of the Second International Symposium on Programming*, Paris, France. Dunod, 106–130.
- Cousot, P. and Cousot, R. (1977). Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: *POPL'77: Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, NY, USA, ACM Press, 238–252.
- Cousot, P. and Cousot, R. (1992). Abstract interpretation and applications to logic programs. *Journal of Logic Programming* **13** (2 & 3) 103–179.
- Cousot, P. and Halbwachs, N. (1978). Automatic discovery of linear restraints among variables of a program. In: *POPL'78: Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, New York, NY, USA, ACM Press, 84–97.

Hans, W. and Winkler, S. (1992). Aliasing and groundness analysis of logic programs through abstract interpretation and its safety. Technical Report 92–27, Technical University of Aachen (RWTH Aachen). Available from <http://sunsite.informatik.rwth-aachen.de/Publications/AIB>. Last accessed March 14, 2013.

Jacobs, D. and Langen, A. (1992). Static analysis of logic programs for independent AND parallelism. *The Journal of Logic Programming* **13** (2–3) 291–314.

Jones, N. D. and Muchnick, S. S. (1979). Flow analysis and optimization of LISP-like structures. In: *POPL'79: Proceedings of the 6th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 244–256.

King, A. (1994). A synergistic analysis for sharing and groundness which traces linearity. In: Sannella, D. (ed.) *Programming Languages and Systems — ESOP'94, 5th European Symposium on Programming Edinburg, UK, April 11–13, 1994, Proceedings*, Lecture Notes in Computer Science, vol. 788, Berlin, Heidelberg, Springer, 363–378.

Méndez-Lojo, M. and Hermenegildo, M. (2008). Precise set sharing analysis for java-style programs. In: Logozzo, F., Peled, D. and Zuck, L. (eds.) *Verification, Model Checking, and Abstract Interpretation*, Lecture Notes in Computer Science, vol. 4905, Berlin, Heidelberg, Springer, 172–187.

Montenegro, M., Peña, R. and Segura, C. (2015). Shape analysis in a functional language by using regular languages. *Science of Computer Programming* **111** 51–78.

Muthukumar, K. and Hermenegildo, M. V. (1992). Compile-time derivation of variable dependency using abstract interpretation. *The Journal of Logic Programming* **13** (2–3) 315–347.

Peña-Marí, R., Segura, C. and Montenegro, M. (2006). A sharing analysis for SAFE. In: Nilsson, H. (ed.) *Trends in Functional Programming*, vol. 7, Bristol, UK. Intellect Books, 109–127.

Pollet, I., Le Charlier, B. and Cortesi, A. (2001). Distinctness and sharing domains for static analysis of Java programs. In: *Proceedings of the 25th European Conference on Object-Oriented Programming (ECOOP)*, Lecture Notes in Computer Science, vol. 2072, Budapest, Hungary, 77–98.

Secci, S. and Spoto, F. (2005a). Pair-sharing analysis of object-oriented programs. In: Hankin, C. (ed.) *Proceedings of Static Analysis Symposium (SAS)*, Lecture Notes in Computer Science, vol. 3672, London, UK, Springer, 320–335.

Secci, S. and Spoto, F. (2005b). Pair-Sharing Analysis of Object-Oriented Programs (long version). Personal communication. Also available at <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.79.2459>.

Zanardini, D. (2018). Field-sensitive sharing. *Journal of Logical and Algebraic Methods in Programming* **95** 103–127.

## Appendix A. Proofs

### A.1 Reachability, sharing, linearity, and aliasing

#### A.1.1 Reachability among locations

**Proposition 90.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$ ,  $l \in \text{dom}(\mu)$ ,  $\bar{f}$  a possible empty sequence of identifiers, if  $l.\bar{f}$  exists, then either  $l.\bar{f} = \text{null}$  or  $l.\bar{f} \in \text{dom}(\mu)$ .*

*Proof.* Assume that  $l.\bar{f}$  exists. The proof is by induction on the length of  $\bar{f}$ . If  $\bar{f}$  is empty, then  $l.\bar{f} = l$ , which is in  $\text{dom}(\mu)$  by hypothesis. Otherwise  $\bar{f} = \bar{f}_1 f_2$  and  $l.\bar{f} = (l.\bar{f}_1).f_2$ . By inductive hypothesis,  $l' = l.\bar{f}_1$  is either  $\text{null}$  or an element of  $\text{dom}(\mu)$ . If  $l' = \text{null}$ , then  $l'.f_2 = l.\bar{f}$  does not exist, which contradicts the hypothesis, hence  $l' \in \text{dom}(\mu)$ . Since  $l'.f_2$  exists, then  $f_2 \in \text{dom}(\mu(l').\phi)$ . By  $\tau$ -correctness of  $\sigma$  we have that  $\mu(l').\phi \star \mu$  is weakly  $\mu(l').\kappa$ -correct. This implies that either  $l.\bar{f} = l.\bar{f}_1 f_2 = l'.f_2 = \text{null}$ , or  $l.\bar{f} = l.\bar{f}_1 f_2 = l'.f_2 \in \text{dom}(\mu)$ .  $\square$

**Proposition 11.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$ ,  $f$  an identifier and  $l, l' \in \text{dom}(\mu)$ , then:*

- (1)  $l.f$  exists iff  $\tau(l).f$  exists;
- (2) if  $l.f$  exists and  $l.f \neq \text{null}$ , then  $\tau(l.f) \leq \tau(l).f$ .

*Proof.* Let us prove the first property. It is the case that  $l.f$  exists iff  $f \in \text{dom}(\mu(l).\phi)$ . By  $\tau$ -correctness,  $\text{dom}(\mu(l).\phi) = \text{dom}(\mu(l).\kappa) = \text{dom}(\tau(l))$ . Hence  $l.f$  exists whenever  $f \in \text{dom}(\tau(l))$  iff  $\tau(l).f$  exists.

We now prove the second property. Since  $l.f$  exists, then  $l$  exists and  $\mu(l) \in \text{rng}(\mu)$ . By this reason and  $\tau$ -correctness of  $\sigma$ , we have that  $\mu(l).\phi \star \mu$  is weakly  $\mu(l).\kappa$ -correct. If we consider what this means for the identifier  $f$ , and since  $l.f = \mu(l).\phi(f) \neq \text{null}$  by hypothesis, we have that

$\mu(\mu(l).\phi(\mathbf{f})).\kappa \leq \mu(l).\kappa(\mathbf{f})$ . By just applying the definitions, we have that  $\mu(\mu(l).\phi(\mathbf{f})).\kappa = \tau(l.\mathbf{f})$  while  $\mu(l).\kappa(\mathbf{f}) = \tau(l).\mathbf{f}$ .  $\square$

**Proposition 16.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and locations  $l_1, l_2 \in \text{dom}(\mu)$ , we have that  $l_1$  shares with  $l_2$  iff  $RLoc_\sigma(l_1) \cap RLoc_\sigma(l_2) \neq \emptyset$ .*

*Proof.* If  $l_1$  and  $l_2$  share, then there are sequences of identifiers  $\bar{f}_1$  and  $\bar{f}_2$  such that  $l_1.\bar{f}_1 = l_2.\bar{f}_2 \neq \text{null}$ . Let  $l$  be  $l_1.\bar{f}_1$ , we have that  $l \in RLoc_\sigma(l_1)$  and  $l \in RLoc_\sigma(l_2)$ .  $\square$

A.1.2 Reachability among identifiers

**Proposition 91.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $v.\mathbf{f} \in Q_\tau$  then  $\phi(v.\mathbf{f})$  exists.*

*Proof.* If  $\phi(v) = \text{null}$ , then  $\phi(v.\mathbf{f}) = \text{null}$  by definition. If  $\phi(v) \neq \text{null}$ , then  $\phi(v) \in \text{dom}(\mu)$ . By  $\tau$ -correctness, the runtime class of  $v$  is a subtype of the declared class, that is,  $\tau(\phi(v)) \leq \tau(v)$ . By properties of subtyping, if  $\tau(v).\mathbf{f}$  exists, then also  $\tau(\phi(v)).\mathbf{f}$  exists and, by Proposition 11 (1),  $\phi(v).\mathbf{f} = \phi(v.\mathbf{f})$  exists.  $\square$

**Proposition 21.** *For each  $i \in I_\tau$  and  $\sigma = \phi \star \mu \in \Sigma_\tau$ , we have that  $\phi(i) \neq \text{null}$  implies  $\tau(\phi(i)) \leq \tau(i)$ .*

*Proof.* If  $i$  is a variable in  $\text{dom}(\tau)$  and  $\phi(i) \neq \text{null}$ , then  $\tau(\phi(i)) \leq \tau(i)$  by weakly  $\tau$ -correctness. If  $i = v.\mathbf{f}$  is a qualified field, since  $\phi(i) \neq \text{null}$  then  $\phi(v) \neq \text{null}$  and, by Proposition 91,  $\phi(v.\mathbf{f}) = \phi(v).\mathbf{f}$ . Hence  $\tau(\phi(v.\mathbf{f})) = \tau(\phi(v).\mathbf{f})$ . By Proposition 11 (2),  $\tau(\phi(v).\mathbf{f}) \leq \tau(\phi(v)).\mathbf{f}$ . Since  $\phi(v) \neq \text{null}$ , by weak  $\tau$ -correctness, we have  $\tau(\phi(v)) \leq \tau(v)$  and, by property of subtyping,  $\tau(\phi(v)).\mathbf{f} = \tau(v).\mathbf{f} = \tau(v.\mathbf{f})$ , concluding the proof.  $\square$

**Proposition 25.** *Let  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $v \in \text{dom}(\tau)$ . If  $\phi(v) \neq \text{null}$ , then*

$$RLoc_\sigma(v) \supseteq \{\phi(v)\} \cup \bigcup_{v.\mathbf{f} \in Q_\tau} RLoc_\sigma(v.\mathbf{f}) .$$

*Proof.* If  $\phi(v) \neq \text{null}$ , then  $\phi(v) \in \text{dom}(\mu)$ . Moreover, by Proposition 90, if  $\phi(v).\mathbf{f}$  exists and  $\phi(v).\mathbf{f} \neq \text{null}$ , then  $\phi(v).\mathbf{f} \in \text{dom}(\mu)$ . Hence:

$$\begin{aligned} RLoc_\sigma(v) &= \\ &= \{l \mid \phi(v) \xrightarrow{*}_\sigma l\} \\ &= \{\phi(v)\} \cup \{l \mid \phi(v).\mathbf{f} \text{ exists} \wedge \phi(v).\mathbf{f} \neq \text{null} \wedge \phi(v).\mathbf{f} \xrightarrow{*}_\sigma l\} \\ &\supseteq \{\phi(v)\} \cup \{l \mid v.\mathbf{f} \in Q_\tau \wedge \phi(v).\mathbf{f} \neq \text{null} \wedge \phi(v.\mathbf{f}) \xrightarrow{*}_\sigma l\} \\ &= \{\phi(v)\} \cup \bigcup_{\substack{v.\mathbf{f} \in Q_\tau \\ \phi(v).\mathbf{f} \neq \text{null}}} \{l \mid \phi(v.\mathbf{f}) \xrightarrow{*}_\sigma l\} \\ &= \{\phi(v)\} \cup \bigcup_{\substack{v.\mathbf{f} \in Q_\tau \\ \phi(v).\mathbf{f} \neq \text{null}}} RLoc_\sigma(v.\mathbf{f}) \\ &= \{\phi(v)\} \cup \bigcup_{v.\mathbf{f} \in Q_\tau} RLoc_\sigma(v.\mathbf{f}) \end{aligned}$$

which concludes the proof.  $\square$

A.1.3 Class-induced reachability

**Proposition 29.** Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $l_1, l_2 \in \text{dom}(\mu)$ , if  $l_1 \xrightarrow{*}_\sigma l_2$ , then  $\tau(l_1) \xrightarrow{*} \tau(l_2)$ .

*Proof.* If  $l_1 = l_2$ , then  $\tau(l_1) \xrightarrow{*}_\sigma \tau(l_2)$ . Otherwise, it is enough to prove that  $l_1 \rightarrow_\sigma l_2$  implies  $\tau(l_1) \rightarrow \tau(l_2)$ . If  $l_1 \rightarrow_\sigma l_2$ , then  $l_2 = l_1.f$  for some  $f$ ,  $\tau(l_2) = \tau(l_1.f) \leq \tau(l_1).f$ , hence  $\tau(l_1) \rightarrow \tau(l_2)$ . □

The operator  $C$  for class reachability is downward closed and monotone w.r.t. subtyping.

**Lemma 92.** The following properties hold:

- (1)  $\downarrow C(\kappa) = C(\kappa)$  for each class  $\kappa$ ;
- (2)  $C(\kappa') \subseteq C(\kappa)$  for each  $\kappa' \leq \kappa$ .

*Proof.* The first property immediately descends from the fact that  $C(\kappa) = \bigcup_{\kappa'' \leq \kappa} \{\kappa' \mid \kappa'' \xrightarrow{*} \kappa'\}$  is the union of downward closed sets. The second property is an immediate consequence of the definition of  $C$ . □

**Lemma 93.** Given  $\sigma = \phi \star \mu \in \Sigma_\tau$  and  $l_1, l_2 \in \text{dom}(\mu)$ , if  $l_1$  and  $l_2$  share, then  $C(\tau(l_1)) \cap C(\tau(l_2)) \neq \emptyset$ .

*Proof.* If  $l_1$  and  $l_2$  share, then by Proposition 16  $RLoc_\sigma(l_1) \cap RLoc_\sigma(l_2) \neq \emptyset$ , that is, there is  $l \in Loc$  s.t.  $l_1 \xrightarrow{*}_\sigma l$  and  $l_2 \xrightarrow{*}_\sigma l$ . By Proposition 29, this implies  $\tau(l_1) \xrightarrow{*} \tau(l)$  and  $\tau(l_2) \xrightarrow{*} \tau(l)$ , which implies  $\tau(l) \in C(\tau(l_1)) \cap C(\tau(l_2))$ . □

**Proposition 32.** Given  $i_1, i_2 \in I_\tau$ , and  $\sigma \in \Sigma_\tau$ , if  $i_1$  and  $i_2$  share in  $\sigma$ , then  $(\tau(i_1), \tau(i_2)) \in SH$ .

*Proof.* If  $i_1$  and  $i_2$  share, then  $\phi(i_1) \neq \text{null} \neq \phi(i_2)$  and  $\phi(i_1)$  shares with  $\phi(i_2)$ . By Lemma 93,  $C(\tau(\phi(i_1))) \cap C(\tau(\phi(i_2))) \neq \emptyset$ . By Proposition 21,  $\tau(\phi(i_1)) \leq \tau(i_1)$  and  $\tau(\phi(i_2)) \leq \tau(i_2)$ . Then by Lemma 92,  $C(\tau(i_1)) \cap C(\tau(i_2)) \neq \emptyset$ . □

**Lemma 94.** Given  $\phi \star \mu \in \Sigma_\tau$  and  $l \in \text{dom}(\mu)$ , if  $l$  is not linear, then  $\tau(l) \in NL$ .

*Proof.* If  $l$  is not linear, by Definition 14, there are two sequences  $\bar{f}_1 \neq \bar{f}_2$  such that  $l.\bar{f}_1 = l.\bar{f}_2 \neq \text{null}$ . We prove by induction on the shortest among  $\bar{f}_1$  and  $\bar{f}_2$  that  $\tau(i) \in NL$ . There are several cases:

- $\bar{f}_1$  is empty. Then  $\bar{f}_2 = f \cdot \bar{f}'_2$  for some  $f$ , and  $l.f.\bar{f}'_2 = l$ . This means  $l.f \xrightarrow{*}_\sigma l$ , hence by Proposition 29,  $\tau(l.f) \xrightarrow{*} \tau(l)$ , that is,  $\tau(l) \in C(\tau(l.f)) \subseteq C(\tau(l).f)$ , where the last inclusion follows by Proposition 11 (2) and Lemma 92 (2). Hence, we have  $\tau(l) \in NL$ .
- $\bar{f}_2$  is empty. The proof is the same as for the previous point.
- $\bar{f}_1 = f_1 \cdot \bar{f}'_1$  and  $\bar{f}_2 = f_2 \cdot \bar{f}'_2$  with  $f_1 \neq f_2$ . Then  $l.f_1$  and  $l.f_2$  share, hence by Lemma 93  $C(\tau(l.f_1)) \cap C(\tau(l.f_2)) \neq \emptyset$ . By Proposition 11 (2) and by monotonicity of  $C$ , that is, Lemma 92 (2),  $C(\tau(l).f_1) \cap C(\tau(l).f_2) \neq \emptyset$ , hence  $\tau(l) \in NL$ .
- $\bar{f}_1 = f \cdot \bar{f}'_1$  and  $\bar{f}_2 = f \cdot \bar{f}'_2$ . Then  $\bar{f}'_1 \neq \bar{f}'_2$  and  $l.f$  is not linear. By inductive hypothesis,  $\tau(l.f) \in NL$ . Since  $l \rightarrow l.f$  then, by Proposition 29,  $\tau(l) \rightarrow \tau(l.f)$ , hence  $\tau(l) \in NL$ .

This concludes the proof. □



**Proposition 34.** Given  $\sigma \in \Sigma_\tau$  and  $i \in I_\tau$ , if  $i$  is not linear in  $\sigma$ , then  $\tau(i) \in NL$ .

*Proof.* If  $i$  is not linear, then  $\phi(i) \neq \text{null}$  is not linear, hence, by Lemma 94  $\tau(\phi(i)) \in NL$ . Since by Proposition 21,  $\tau(\phi(i)) \leq \tau(i)$  and  $NL$  is upward closed, then  $\tau(i) \in NL$ .  $\square$

**A.2 Aliasing graphs and aliasing morphisms**

The following proposition relates edges in aliasing graphs with reachability among classes.

**Proposition 95.** Given an aliasing graph  $G$  and nodes  $n, m \in N$ , we have that:

- if  $n \xrightarrow{f} m$ , then  $\psi_G(n) \rightarrow \tau_G(m)$ ;
- if there is a path from  $n$  to  $m$ , then  $\tau_G(m) \in C(\tau_G(n))$ .

*Proof.* For the first point, if  $n \xrightarrow{f} m$  there is  $v \in \text{dom}(\tau)$  such that  $\ell(v) = n$  and  $\ell(v.f) = m$ . By definition of  $\tau$ , we have  $\tau_G(m) \leq \tau(v.f) = \tau(v).f$ . Moreover,  $\psi_G(n) \leq \tau(v)$ . Since fields cannot be redefined in subclasses,  $\tau(v).f = \psi_G(n).f$ , hence the thesis. The second point is an immediate corollary.  $\square$

In the following, we show that  $\ell(v.f)$  only depends on  $\ell(v)$  and not on  $v$  itself.

**Lemma 96.** Given two aliasing graphs  $G_1, G_2 \in \mathcal{G}_\tau$ , the following properties are true for every  $h : G_1 \rightarrow G_2$ :

- if  $n \xrightarrow{f} m \in E_1$  and  $h(m) \neq \perp$ , then  $h(n) \xrightarrow{f} h(m) \in E_2$ ;
- if  $n \in N_1$  and  $h(n) \neq \perp$ , then  $\tau_{G_1}(n) \geq \tau_{G_2}(h(n))$  and  $\psi_{G_1}(n) \geq \psi_{G_2}(h(n))$ ;
- for each  $n_2 \in N_2$ , there is  $n_1 \in N_1$  s.t.  $h(n_1) = n_2$ ,  $\tau_{G_1}(n_1) = \tau_{G_2}(n_2)$  and  $\psi_{G_1}(n_1) = \psi_{G_2}(n_2)$ .

*Proof.* For the first point, let  $n \xrightarrow{f} m \in E_1$ . There exists  $v \in \text{dom}(\tau)$  such that  $\ell_1(v) = n$  and  $\ell_1(v.f) = m$ . Composing with  $h$ , we have  $\ell_2(v) = h(n)$  and  $\ell_2(v.f) = h(m)$ . By definition of  $\ell$  for qualified fields, since  $h(m) \neq \perp$  then  $h(n) \neq \perp$ , too. In turn, this means  $h(n) \xrightarrow{f} h(m) \in E_2$ .

For the second point,  $\{i \in I_\tau \mid \ell_1(i) = n\} \subseteq \{i \in I_\tau \mid h(\ell_1(i)) = h(n)\} = \{i \in I_\tau \mid \ell_2(i) = h(n)\}$ , and therefore  $\tau_{G_2}(h(n)) \leq \tau_{G_1}(n)$  and  $\psi_{G_1}(n) \geq \psi_{G_2}(h(n))$ .

For the last point, since  $\tau(\ell_2^{-1}(n_2))$  is a chain, then  $\tau_{G_2}(n_2) = \tau(i)$  for some  $i \in I_\tau$  such that  $\ell_2(i) = n_2$ . Let  $n_1 = \ell_1(i)$ , so that  $h(n_1) = n_2$ . Obviously,  $\tau_{G_1}(n_1) \leq \tau(i) = \tau_{G_2}(n_2)$ . The converse inequality comes from the previous point. The same holds for  $\psi_{G_1}$  and  $\psi_{G_2}$ .  $\square$

**Proposition 97.** Given two aliasing graphs  $G_1, G_2 \in \mathcal{G}_\tau$ , the following properties are true for every  $h : G_1 \rightarrow G_2$ :

- $h$  is the unique morphism from  $G_1$  to  $G_2$ ;
- $h$  is surjective;
- $h$  is an isomorphism iff it is total and injective; the inverse morphism is  $h^{-1}$ .

*Proof.* For the first point, assume  $h' : G_1 \rightarrow G_2$ . Then, for each node  $n \in N_1$ , there is an identifier  $i \in I_\tau$  such that  $\ell_1(i) = n$ . Therefore,  $h'(n) = h'(\ell_1(i)) = \ell_2(i) = h(\ell_1(i)) = h(n)$ , that is,  $h = h'$ . For the second point, given  $n \in N_2$ , there is  $i \in I_\tau$  such that  $n = \ell_2(i)$ . Since  $\ell_2(i) = h(\ell_1(i))$ , then  $n$  is in the range of  $h$ , hence  $h$  is surjective. The third point is straightforward.  $\square$

Since by Proposition 97, there exists a single morphism  $h$  between aliasing graphs, it is natural to define a preorder  $\sqsubseteq$  on  $\mathcal{G}_\tau \times \mathcal{G}_\tau$  by

$$G_1 \sqsubseteq G_2 \iff \exists h : G_2 \rightarrow G_1 \text{ in } \mathcal{G}_\tau$$

**Lemma 98.** *Given two aliasing graphs  $G_1, G_2 \in \mathcal{G}_\tau$ ,  $G_2 \sqsubseteq G_1$ , and  $i \in I_\tau$ , if  $\ell_2(i) \neq \perp$  then  $\ell_2^{-1}(\ell_2(i)) \supseteq \ell_1^{-1}(\ell_1(i))$ .*

*Proof.* Since  $G_2 \sqsubseteq G_1$ , there exists a morphism  $h : G_1 \rightarrow G_2$ . Assume  $i \in I_\tau$  such that  $\ell_2(i) \neq \perp$ . If  $\ell_1(i) = n$ , then  $\ell_2(i) = h(n)$ . Therefore,  $\ell_2^{-1}(\ell_2(i)) = \ell_2^{-1}(h(n)) = \ell_1^{-1}(h^{-1}(h(n))) \supseteq \ell_1^{-1}(n) = \ell_1^{-1}(\ell_1(i))$ , and then the thesis.  $\square$

Finally, it is possible to characterize the preorder between aliasing graphs without using the concept of morphism of graphs.

**Theorem 99.** *Given  $G_1, G_2 \in \mathcal{G}_\tau$ , we have  $G_1 \sqsubseteq G_2$  iff  $G_1 \leq G_2$ .*

*Proof.* Assume  $G_1 \sqsubseteq G_2$ . By definition there exists  $h : G_2 \rightarrow G_1$ . Then, if  $\ell_2(i) = \ell_2(i')$ , then  $\ell_1(i) = h(\ell_2(i)) = h(\ell_2(i')) = \ell_1(i')$ . Moreover, if  $\ell_2(i) = \perp$ , then  $\ell_1(i) = h(\ell_2(i)) = \perp$ .

On the other side, assume  $G_1 \leq G_2$ . For each node  $n \in N_2$ , there is  $i \in I_\tau$  s.t.  $\ell_2(i) = n$ . We let  $h(n) = \ell_1(i)$ . This is well defined since if  $\ell_2(j) = n$ , then  $\ell_1(j) = \ell_1(i)$ . Moreover, it is a graph morphism. Given  $i \in I_\tau$ , if  $\ell_1(i) = n$ , then  $h(\ell_2(i)) = \ell_1(i)$  by the definition above. If  $\ell_2(i) = \perp$ , then  $\ell_1(i) = \perp$ , hence  $h(\ell_2(i)) = \ell_1(i)$  again.  $\square$

**Theorem 42.** *Given two aliasing graphs  $G_1, G_2$ , there exists a morphism from  $G_2$  to  $G_1$  if and only if  $G_1 \leq G_2$ . Moreover, the morphism, when it exists, is unique.*

*Proof.* The proof is straightforward by Proposition 97 and Theorem 99.  $\square$

### A.2.1 The lattice of aliasing graphs

First of all, we prove a characterization for least upper bounds.

**Lemma 100.** *Assume given  $G_1, G_2, G \in \mathcal{G}_\tau$  such that  $G_1 \leq G, G_2 \leq G$ , with morphisms  $h_1 : G \rightarrow G_1$  and  $h_2 : G \rightarrow G_2$ . Then,  $G$  is the least upper bound of  $G_1$  and  $G_2$  if:*

- $\forall n, m \in N. (h_1(n) = h_1(m) \wedge h_2(n) = h_2(m)) \Rightarrow n = m$ ;
- $\nexists n \in N. h_1(n) = h_2(n) = \perp$ .

*Proof.* First of all, observe that the existence of  $h_1$  and  $h_2$  follows by Theorem 99. Moreover, note that the two conditions are equivalent to  $\forall n, m \in N \cup \{\perp\}. (h_1(n) = h_1(m) \wedge h_2(n) = h_2(m)) \Rightarrow n = m$ . If the original condition holds and there are  $n, m \in N \cup \{\perp\}$  with  $h_1(n) = h_1(m) \wedge h_2(n) = h_2(m)$ , we have that: either  $n, m \in N$ , and the results follow immediately, or one among  $n$  and  $m$  is  $\perp$ . Assume without loss of generality that  $n = \perp$ . Then  $h_1(m) = h_2(m) = \perp$  which implies  $m = \perp$  by hypothesis, hence  $n = m$ . On the converse, if the condition stated above holds and  $h_1(n) = h_2(n) = \perp$ , then  $h_1(n) = h_1(\perp)$  and  $h_2(n) = h_2(\perp)$ , hence  $n = \perp$ .

Now, assume this property hold, and we prove that  $G$  is the least upper bound. Let  $G'$  such that  $G_1 \leq G'$  and  $G_2 \leq G'$  with corresponding morphisms  $h'_1$  and  $h'_2$ . We need to build a morphism  $h : G' \rightarrow G$ . For each  $n \in N'$ , consider a qualified identifier  $i \in I_\tau$  such that  $\ell'(i) = n$  and we define  $h(n) = \ell(i)$ . We prove that  $h$  is well defined and that it is a morphism. Assume there are  $i', i$  with  $i' \neq i, \ell'(i) = \ell'(i') = n$ . Then  $\ell_1(i) = \ell_1(i')$  and  $\ell_2(i) = \ell_2(i')$ , that is,  $h_1(\ell(i)) = h_1(\ell(i'))$

and  $h_2(\ell(i)) = h_2(\ell(i'))$ . By the alternative formulation of hypothesis, this implies  $\ell(i) = \ell(i')$ . Therefore,  $h$  is well defined.

Now, we prove  $\ell(i) = h(\ell'(i))$  for each  $i \in I_\tau$ . If  $\ell'(i) \in N'$ , this is immediate, since  $h(\ell'(i))$  is  $\ell(i')$  for some identifier  $i'$  such that  $\ell(i') = \ell(i)$ . Obviously  $i' = i$  is a good choice, hence  $\ell(i) = h(\ell'(i))$ . If  $\ell'(i) = \perp$ , then  $\ell_1(i) = \ell_2(i) = \perp$  hence  $h_1(\ell(i)) = h_2(\ell(i)) = \perp$ . By hypothesis,  $\ell(i) = \perp = h(\ell'(i))$ .  $\square$

Then, using this characterization, we prove that  $G_1 \curlywedge G_2$  in Definition 44 is the least upper bound of  $G_1$  and  $G_2$ .

**Theorem 101.** *The aliasing graph  $G_1 \curlywedge G_2$  is the least upper bound of  $G_1$  and  $G_2$ .*

*Proof.* **G is a pre-aliasing graph).** First of all, we prove that  $G = G_1 \curlywedge G_2$  is a pre-aliasing graph.

Given  $S_1 \in N$  and  $f \in Ide$ , assume that there are  $S_2, S'_2 \in N$  such that  $S_1 \xrightarrow{f} S_2$  and  $S_1 \xrightarrow{f} S'_2$ . This means there are  $v, w \in S_1 \cap Ide$  such that  $v \sim w$ ,  $v.f \in S_2$  and  $w.f \in S'_2$ . We need to prove that  $S_2 = S'_2$ . Since  $v \sim w$  then  $\ell_1(v) = \ell_1(w)$  and  $\ell_2(v) = \ell_2(w)$ , which implies  $\ell_1(v.f) = \ell_1(w.f)$ ,  $\ell_2(v.f) = \ell_2(w.f)$  and in turn,  $v.f \sim w.f$ , that is,  $S_2 = S'_2$ . Now assume there are  $S_1, S_2 \in N$  and  $f \in Ide$  with  $S_1 \xrightarrow{f} S_2$ . By construction of  $G$ , there is  $v \in S_1$  s.t.  $v.f \in S_2$ . This implies  $v \in Ide$ ,  $v.f \in Q_\tau$  and  $\ell(v) = [v]_\sim = S_1$ , proving that the second condition in Definition 35 holds.

**A small lemma).** Note that  $\ell(i) = [i]_\sim$  for any  $i \in X$  and  $\ell(i) = \perp$  otherwise. This happens by definition when  $i \in Ide$ , hence we only need to prove the property for  $i = v.f \in Q_\tau$ . When  $v.f \in X$ , then  $v \in X$  and  $[v]_\sim \xrightarrow{f} [v.f]_\sim$ , hence  $\ell(v) = [v]_\sim$  and  $\ell(v.f) = [v.f]_\sim$ . When  $v \notin X$ , then  $\ell(v) = \perp$  and  $\ell(v.f) = \perp$ , too. If  $v \in X$  but  $v.f \notin X$ , we should prove  $\ell(v.f) = \perp$ . For the sake of contradiction, assume  $\ell(v.f) = S$ , that is, there is an arrow  $\ell(v) = [v]_\sim \xrightarrow{f} S$ . This means there is  $w \in Ide$  such that  $w \sim v$ ,  $w \neq v$  and  $w.f \in S$ . Since  $w.f \in X$ , then  $\ell_1(w.f) \neq \perp \neq \ell_2(w.f)$ . But since  $v \sim w$ , then  $\ell_1(v) = \ell_1(w)$  and  $\ell_1(v.f) = \ell_1(w.f) \neq \perp$ . The same holds for  $\ell_2(v.f)$ . Therefore,  $v.f \in X$ , which is a contradiction.

$G \in \mathcal{G}_\tau$ ). Now we prove that  $G$  is an aliasing graph. Given  $S \in N$ , we need to check that  $\{\tau(i) \mid i \in I_\tau \wedge \ell(i) = n\}$  is a chain. Let  $i_1, i_2 \in I_\tau$  such that  $\ell(i_1) = \ell(i_2) = n$ . Since by the previous small lemma  $\ell(i_1) = [i_1]_\sim$  and  $\ell(i_2) = [i_2]_\sim$ , we have  $\ell_1(i_1) = \ell_1(i_2) \in N_1$ , which means that  $\tau(i_1)$  and  $\tau(i_2)$  are comparable, since  $G_1$  is an aliasing graph.

**G is the least upper bound of  $G_1$  and  $G_2$ .** Finally, we prove that  $G$  is the least upper bound of  $G_1$  and  $G_2$ . Consider the map  $h_1 : N \rightarrow N_1$  such that  $h_1([i]_\sim) = \ell_1(i)$  for any  $[i]_\sim \in N$ . If  $i \in X$ , then  $h_1(\ell(i)) = h_1([i]_\sim) = \ell_1(i)$ , otherwise  $i \notin \ell_1^{-1}(N_1)$ , hence  $h_1(\ell(i)) = \perp = \ell_1(i)$ . This means that  $h_1 : G \rightarrow G_1$ . The same holds for  $h_2 : G \rightarrow G_2$ . Now, if  $h_1([i]_\sim) = h_1([i']_\sim)$  and  $h_2([i]_\sim) = h_2([i']_\sim)$ , then  $\ell_1(i) = \ell_1(i')$  and  $\ell_2(i) = \ell_2(i')$  hence  $i \sim i'$  and  $[i]_\sim = [i']_\sim$ . Moreover, for each  $[i]_\sim$ , either  $h_1([i]_\sim) = \ell_1(i) \in N_1$  or  $h_2([i]_\sim) = \ell_2(i) \in N_2$ . By Lemma 100 follows that  $G$  is the least upper bound of  $G_1$  and  $G_2$ .  $\square$

Now, we prove that  $G_1 \curlywedge G_2$  in Definition 46 is the greatest lower bound of  $G_1$  and  $G_2$ .

**Theorem 102.** *The aliasing graph  $G_1 \curlywedge G_2$  is the greatest lower bound of  $G_1$  and  $G_2$ .*

*Proof.* Let  $G = G_1 \curlywedge G_2$ . The following holds:

**G is a pre-aliasing graph).** First, we have to prove that  $\forall S \in N, \forall f \in Ide$ , there is at most one outgoing edge from  $S$  labeled by  $f$ . Let  $S_1 \in N$  and assume there are edges  $S_1 \xrightarrow{f} S_2$  and  $S_1 \xrightarrow{f} S_3 \in E$ . By construction there are  $v, w \in S_1$  s.t.  $S_2 = [v.f]_\sim$  and  $S_3 = [w.f]_\sim$ . By definition

of  $G$ , there is a sequence  $v = i_1 \sim \dots \sim i_t = w$  such that  $\ell_1(i_s) = \ell_1(i_{s+1})$  or  $\ell_2(i_s) = \ell_2(i_{s+1})$  for each  $s \in \{1, \dots, t-1\}$ . Then, for each  $s \in \{1, \dots, t-1\}$ , either  $\ell_1(i_s.f) = \ell_1(i_{s+1}.f)$  or  $\ell_2(i_s.f) = \ell_2(i_{s+1}.f)$ . By definition of  $\sim$ , we have that  $v.f \sim w.f$ . Therefore,  $S_2 = [v.f]_{\sim} = [w.f]_{\sim} = S_3$ .

Now, we have to prove that  $\forall S_1, S_2 \in N, \forall f \in Ide$ , if  $S_1 \xrightarrow{f} S_2$  there exists  $v \in \text{dom}(\tau)$  such that  $\ell(v) = S_1$  and  $v.f \in Q_\tau$ . Let  $S_1, S_2 \in N$  and let  $f \in Ide$  such that  $S_1 \xrightarrow{f} S_2$ . By construction, there exists  $v \in S_1$  s.t.  $v.f \in S_2$ . Obviously,  $S_1 = [v]_{\sim} = \ell(v)$ . Moreover  $S_2 = [v.f]_{\sim} \in N$  and therefore  $v.f \in [v.f]_{\sim} \subseteq \ell_1^{-1}(N_1) \cap \ell_2^{-1}(N_2) \subseteq Q_\tau$ .

**A small lemma.** Note that  $\ell(i) = [i]_{\sim}$  for any  $i \in N$  and  $\ell(i) = \perp$  otherwise. The proof is analogous to the one for  $\Upsilon$ , hence it is omitted here.

$G \in \mathcal{G}_\tau$ ). We have to prove that for each  $S \in N$ , the set  $\{\tau(i) \mid i \in I_\tau \wedge \ell(i) = S\}$  is a nonempty chain. By construction for each  $S \in N$ , there exists  $j \in I_\tau$  such that  $S = [j]_{\sim}$  and  $\tau([j]_{\sim})$  is a chain. Note that  $\tau([j]_{\sim}) = \{\tau(i) \mid i \in I_\tau \wedge i \sim j\}$ . By the previous small lemma  $\tau([j]_{\sim}) = \{\tau(i) \mid i \in I_\tau \wedge \ell(i) = \ell(j)\} = \{\tau(i) \mid i \in I_\tau \wedge \ell(i) = S\}$  which is the require property.

**G is a lower bound of  $G_1$  and  $G_2$ .** Let us consider the map  $h_1 : N_1 \rightarrow N$  such that  $h_1(n_1) = [i]_{\sim}$  if  $i \in I_\tau, \ell_1(i) = n_1$  and  $[i]_{\sim} \in N, h_1(n_1) = \perp$  otherwise. First, we prove that  $h_1$  is well defined. Let  $j \in I_\tau$  such that  $\ell_1(j) = n_1$ . By definition,  $i \sim j$  and therefore  $[i]_{\sim} = [j]_{\sim} \in N$ . Moreover, note that  $h_1(\ell_1(i)) = [i]_{\sim} = \ell(i)$  both when  $[i]_{\sim} \in N$  and  $[i]_{\sim} \notin N$ . Therefore,  $h_1 : G_1 \rightarrow G$ . The same holds for  $h_2 : G \rightarrow G_2$ .

**G is the greatest lower bound of  $G_1$  and  $G_2$ .** Let  $G' \in \mathcal{G}_\tau$  and assume that  $G' \preceq G_1$  and  $G' \preceq G_2$  with the corresponding morphisms  $h'_1$  and  $h'_2$  (by Theorem 99).

First of all, observe that for each  $i, j \in I_\tau$

$$[i]_{\sim} = [j]_{\sim} \Rightarrow \ell'(i) = \ell'(j) . \tag{A1}$$

In fact, if  $[i]_{\sim} = [j]_{\sim}$ , there is a sequence  $i = i_1 \sim \dots \sim i_t = j$  such that  $\ell_1(i_s) = \ell_1(i_{s+1})$  or  $\ell_2(i_s) = \ell_2(i_{s+1})$  for each  $s \in \{1, \dots, t-1\}$ . If  $\ell_1(i_s) = \ell_1(i_{s+1})$ , then  $\ell'(i_s) = h'_1(\ell_1(i_s)) = h'_1(\ell_1(i_{s+1})) = \ell'(i_{s+1})$ . Analogously if  $\ell_2(i_s) = \ell_2(i_{s+1})$ . As a result, we have  $\ell'(i) = \ell'(j)$ . Note that this implies:

$$\{j \mid j \in I_\tau \wedge \ell(j) = \ell(i) \neq \perp\} \subseteq \{j \mid j \in I_\tau \wedge \ell'(j) = \ell'(i)\} . \tag{A2}$$

We need to build a morphism  $h' : G \rightarrow G'$ . For each  $S \in N$ , consider a qualified identifier  $i \in I_\tau$  such that  $\ell(i) = [i]_{\sim} = S$  and we define  $h'(S) = \ell'(i)$ . We only need to prove that  $h'$  is well defined and that it is a morphism. Assume there is  $j \in I_\tau$ , with  $j \neq i$  and  $\ell(j) = [j]_{\sim} = S$ . By (A1), we have  $\ell'(i) = \ell'(j)$  and therefore  $h'$  is well defined.

Now, we prove that  $\ell'(i) = h'(\ell(i))$ . If  $\ell(i) \in N$ , this is immediate, since  $h'(\ell(i))$  is  $\ell'(i')$  for some identifier  $i'$  such that  $\ell(i) = \ell(i')$ . Obviously  $i' = i$  is a good choice, hence  $\ell'(i) = h'(\ell(i))$ . If  $\ell(i) = \perp$ , the proof is by contradiction. Assume that  $\ell'(i) = n' \neq \perp$ .

Since  $\ell(i) = \perp$ , we have one of the following possibilities:

- $\tau([i]_{\sim})$  is not a chain. In this case, by (A2),  $\tau([i]_{\sim}) \subseteq \{\tau(j) \mid j \in I_\tau \wedge \ell'(j) = n'\}$ . Therefore,  $\{\tau(j) \mid j \in I_\tau \wedge \ell'(j) = n'\}$  is not a chain and this contradicts the fact that  $G'$  is an aliasing graph.
- $[i]_{\sim} \notin \ell_1^{-1}(N_1) \cap \ell_2^{-1}(N_2)$ . In this case, by (A2),  $\{j \mid j \in I_\tau \wedge \ell'(j) = n'\} \not\subseteq \ell_1^{-1}(N_1) \cap \ell_2^{-1}(N_2)$ . This means that there exists  $j \in I_\tau$  such that  $\ell'(j) = n'$  and  $\ell_1(j) = \perp$  or  $\ell_2(j) = \perp$ . Now, we have a contradiction, since by definition of morphism and by Theorem 99 either  $G' \not\preceq G_1$  or  $G' \not\preceq G_2$ .
- $[v.f]_{\sim} = [i]_{\sim}$  and  $[v]_{\sim} \notin N$ . We define
  - $X_0 = \{[i]_{\sim} \mid i \in I_\tau, \tau([i]_{\sim}) \text{ is a chain and } [i]_{\sim} \subseteq \ell_1^{-1}(N_1) \cap \ell_2^{-1}(N_2)\}$ ;
  - $X_{n+1} = X_n \setminus \{[v.f]_{\sim} \in X_n \mid [v]_{\sim} \notin X_n\}$ .

By definition and since  $I_\tau$  is finite, there exists  $m \geq 0$  such that  $N = \bigcap_{n \leq m} X_n$ . Since  $[v]_\sim \notin N$ , there exists a least  $n \leq m$  such that  $[v]_\sim \notin X_n$ . Now the proof is by induction on  $n$ .

- $[v]_\sim \notin X_0$ . By previous points  $\ell'(v) = \perp$  and therefore  $\ell'(v.f) = \perp$ .
- $[v]_\sim \notin X_{n+1}$ . Since  $[v]_\sim \in X_n$  there exists  $w \in \text{dom}(\tau)$  such that  $[w.f']_\sim = [v]_\sim$  and  $[w]_\sim \notin X_n$ . By inductive hypothesis  $\ell'(w) = \perp$  and therefore  $\ell'(w.f') = \perp$ . Then by (A1),  $\ell'(v) = \ell'(w.f') = \perp$ ,  $\ell'(i) = \ell'(v.f) = \perp$  and we have a contradiction.  $\square$

**Theorem 48.** *The preordered set  $(\mathcal{G}_\tau, \leq)$  has*

- a least element  $\perp_\tau = \emptyset \star \emptyset \star \perp$  where  $\perp$  is the always undefined map;
- a greatest element  $\top_\tau = I_\tau \star E \star \text{id}$  where  $n_1 \xrightarrow{f} n_2 \in E \iff n_1 = v \in \text{dom}(\tau) \wedge n_2 = v.f \in Q_\tau$ ;
- a least upper bound  $G_1 \vee G_2$  for each  $G_1, G_2 \in \mathcal{G}_\tau$ ;
- a greatest lower bound  $G_1 \wedge G_2$  for each  $G_1, G_2 \in \mathcal{G}_\tau$ .

*Proof.* Immediately follows by Theorems 101 and 102.  $\square$

A.2.2 Projection and propagation of nullness for aliasing graphs

**Lemma 103.** *If  $G$  is a pre-aliasing graph and  $X \subseteq N$ , then  $G|_X$  is a pre-aliasing graph.*

*Proof.* Assume  $G$  is a pre-aliasing graph. In  $G|_X$ , the first condition for pre-aliasing graph is trivially respected, since projection does not introduce any new edge. The second condition is also respected, since no new edges are introduced and labels for nodes in  $X$  are preserved. Therefore,  $G|_X$  is a pre-aliasing graph.  $\square$

**Proposition 50.** *If  $G \in \mathcal{G}_\tau$  and  $X \subseteq N$  is backward closed, then  $G|_X \in \mathcal{G}_\tau$ . Moreover, for each  $n \in X$ ,  $\tau_{G|_X}(n) = \tau_G(n)$  and  $\psi_{G|_X}(n) = \psi_G(n)$ .*

*Proof.* Let  $G|_X = X \star E' \star \ell'$ . First of all,  $G|_X$  is a pre-aliasing graph by Lemma 103. Now consider  $n \in X$  and the set  $Y = \{\tau(i) \mid i \in I_\tau \wedge \ell'(i) = n\}$ . Note that, for each  $i \in I_\tau$ ,  $\ell(i) = n$  iff  $\ell'(i) = n$ . This is obvious when  $i$  is a variable simply by definition of  $\ell'$ . If  $i = v.f$  and  $\ell(i) = n$ , then there is  $m \in N$  such that  $\ell(v) = m$  and  $m \xrightarrow{f} n \in E$ . Since  $X$  is backward closed, then  $m \in X$ , hence  $\ell'(v) = m$ ,  $m \xrightarrow{f} n \in E'$  and  $\ell'(i) = n$ . On the converse, if  $\ell'(i) = n$ , then  $\ell(i) = n$  follows trivially by definition of  $\ell'$ .

Therefore,  $Y = \{\tau(i) \mid i \in I_\tau \wedge \ell(i) = n\}$  and it is a nonempty chain since  $G$  is an aliasing graph. Then  $G|_X$  is an aliasing graph and  $\tau_{G|_X}(n) = \tau_G(n)$  for each  $n \in X$ . In the same way,  $\psi_{G|_X}(n) = \psi_G(n)$  for each  $n \in X$ .  $\square$

**Proposition 51.** *If  $G$  is a pre-aliasing graph and  $X \subseteq N$  is backward closed, then  $G|_X \leq G$ .*

*Proof.* Let  $G|_X = X \star E' \star \ell'$ . Consider the partial map  $h : N \rightarrow X$  which is the identity on  $X$  and undefined on  $N \setminus X$ . We show  $h : G \rightarrow G|_X$  is a morphism.

Let  $i$  be an identifier. If  $i$  is a variable  $v$  and  $\ell'(i) = \perp$ , then  $\ell(i) \notin X$ , hence  $h(\ell(i)) = \perp = \ell'(i)$ . If  $\ell'(i) = n$ , then  $n \in X$ , and  $h(\ell(i)) = h(n) = n = \ell'(i)$ . If  $i$  is the qualified identifier  $v.f$ , consider the following cases:

- $\ell(v) \notin X$ : Since  $X$  is backward closed,  $\ell(i) \notin X$ . Therefore,  $h(\ell(i)) = \perp = \ell'(v).f = \ell'(i)$ .

- $\ell(v) \in X$  and  $\ell(v.f) \notin X$ : Then  $\ell(v) = \ell'(v) = n \in X$ ,  $\ell(v.f) = m \notin X$  and  $n \xrightarrow{f} m \in E$ . By definition of  $E'$ , there is no outgoing edge from  $n$  labeled by  $f$  in  $G|_X$ . Hence,  $h(\ell(v.f)) = h(m) = \perp = \ell'(v.f) = \ell'(v).f = n$ .
- $\ell(v) \in X$  and  $\ell(v.f) \in X$ : Then  $\ell(v) = \ell'(v) = n \in X$ ,  $\ell(v.f) = m \in X$  and  $n \xrightarrow{f} m \in E$ . By definition of  $E'$ ,  $n \xrightarrow{f} m \in E'$ , hence  $\ell'(v.f) = m$ . Therefore,  $h(\ell(v.f)) = h(m) = m = \ell'(v.f)$ .  $\square$

### A.2.3 The domain of aliasing graphs

The following propositions show that the abstraction of a concrete state is an aliasing graph and that each aliasing graph can be viewed as the abstraction of a concrete state.

**Proposition 53.** *Given  $\sigma = \phi \star \mu \in \Sigma_\tau$ ,  $G = \alpha_a(\sigma)$  is an aliasing graph and, for each  $i \in I_\tau$ ,  $\ell(i) = \phi(i)$  if  $\phi(i) \neq \text{null}$ ,  $\ell(i) = \perp$  otherwise.*

*Proof.* First of all, we prove  $G$  is a pre-aliasing graph. If  $l \xrightarrow{f} l'$  and  $l \xrightarrow{f} l''$ , then  $l.f = l' = l''$ , hence  $l' = l''$ . Moreover, there exists  $v \in \text{dom}(\tau)$  s.t.  $\ell(v) = l$  and  $v.f \in Q_\tau$ .

Now we prove  $\ell(i) = \phi(i)$  for each  $i \in I_\tau$  (modulo the fact that  $\text{null}$  corresponds to  $\perp$ ). If  $i \in \text{dom}(\tau)$ , the thesis follows directly by definition. Otherwise  $i = v.f$ . If  $\phi(v) = \text{null}$ , we have  $\phi(v.f) = \text{null}$ ,  $\ell(v) = \ell(v.f) = \perp$ . If  $\phi(v) = l \neq \text{null}$ , then  $\phi(v.f) = \phi(v).f = l.f$ . Then  $\ell(v) = l$ ,  $l \xrightarrow{f} l.f \in E$ , and  $\ell(v.f) = l.f = \phi(v.f)$  if and only if  $\phi(v.f) \neq \text{null}$ .

Finally, we prove that  $G$  is an aliasing graph. Given a node  $l$ , we consider the set  $\tau(\ell^{-1}(l)) = \tau(\{i \in I_\tau \mid \ell(i) = l\}) = \tau(\{i \in I_\tau \mid \phi(i) = l\})$ . By Proposition 21, we have that  $\tau(i) \geq \tau(\phi(i))$  for each  $i$  such that  $\phi(i) \neq \text{null}$ . Therefore,  $\kappa \in \tau(\ell^{-1}(l))$  implies  $\kappa \geq \tau(l)$ . Since we do not allow multiple inheritance, this means  $\tau(\ell^{-1}(l))$  is a chain.  $\square$

**Proposition 54.** *Given  $G \in \mathcal{G}_\tau$ , there exists  $\sigma \in \Sigma_\tau$  s.t.  $\alpha_a(\sigma)$  and  $G$  are equivalent, that is,  $\alpha_a(\sigma) \sim G$ .*

*Proof.* Let  $L$  be a set of locations of the same cardinality of  $N$ , and  $\iota : N \rightarrow L$  be a bijective map. Consider the state  $\sigma = \phi \star \mu$  such that  $\text{dom}(\mu) = L$  and:

- for each  $v \in \text{dom}(\tau)$ , if  $\ell(v) = n$ , then  $\phi(v) = \iota(n)$ , otherwise  $\phi(v) = \text{null}$ ;
- for each  $n \in N$ ,  $\mu(\iota(n)) = \kappa_n \star \phi_n$  where  $\kappa_n = \tau_G(n)$
- for each  $n \in N$  and  $f \in \text{dom}(\kappa_n)$ , if there is  $n' \in N$  such that  $n \xrightarrow{f} n' \in E$ , then  $\phi_n(f) = \iota(n')$ , otherwise  $\phi_n(f) = \text{null}$ .

It is easy to check that  $\sigma \in \Sigma_\tau$ . Let  $G' = \alpha_a(\sigma)$  and we prove that  $\iota : G \rightarrow G'$  is an isomorphism. Since  $\iota$  is total and injective, by Proposition 97 it is enough to prove that  $\iota$  is a graph morphism. In turn, this means proving that, for each identifier  $i \in I_\tau$ ,  $\iota(\ell(i)) = \ell'(i) = \phi(i)$  when  $\phi(i) \neq \text{null}$  and  $\iota(\ell(i)) = \perp$  when  $\phi(i) = \text{null}$ .

- for each  $v \in \text{dom}(\tau)$ , if  $\ell(v) = n$ , then  $\phi(v) = \iota(n)$ , hence  $\iota(\ell(v)) = \phi(v)$ . Otherwise, if  $\ell(v) = \perp$  then  $\phi(v) = \text{null}$  and  $\iota(\ell(v)) = \perp$ ;
- for each  $v.f \in Q_\tau$  such that  $\ell(v.f) = n' \neq \perp$ , then  $\ell(v) = n \neq \perp$  and  $n \xrightarrow{f} n' \in E$ . Since both  $n, n' \in N$  and  $\phi(v) = \iota(n)$ , we have  $\phi_n(f) = \iota(n')$ , that is,  $\phi(v.f) = \mu(\iota(n)).\phi(f) = \phi_n(f) = \iota(n') = \iota(\ell(v.f))$ ;
- for each  $v.f \in Q_\tau$  such that  $\ell(v.f) = \perp$ , either  $\ell(v) = \perp$  or  $\ell(v) = n \neq \perp$ . In the first case,  $\phi(v) = \text{null}$ , hence  $\phi(v.f) = \text{null}$ . Otherwise, there is no  $n'$  such that  $n \xrightarrow{f} n'$  which means  $\phi(v.f) = \phi_n(f) = \text{null}$ .  $\square$

**Theorem 55.** *The preorder  $\preceq$  is the same preorder induced by  $\gamma_a$ , that is, given  $G_1, G_2 \in \mathcal{G}_\tau$ ,  $G_1 \preceq G_2$  iff  $\gamma_a(G_1) \subseteq \gamma_a(G_2)$ .*

*Proof.* That  $G_1 \preceq G_2$  implies  $\gamma_a(G_1) \subseteq \gamma_a(G_2)$  is trivial. On the converse, assume  $\gamma_a(G_1) \subseteq \gamma_a(G_2)$  and consider a state  $\sigma$  such that  $\alpha_a(\sigma) \sim G_1$  (it exists by Proposition 54). Since  $\alpha_a(\sigma) \preceq G_1$  we have  $\sigma \in \gamma_a(G_1) \subseteq \gamma_a(G_2)$ , that is,  $\alpha_a(\sigma) \preceq G_2$ . By  $G_1 \preceq \alpha_a(\sigma)$ , we have the required result  $G_1 \preceq G_2$ .  $\square$

**A.3 ALPS graphs**

We begin by proving that  $\preceq$  is a preorder on ALPS-graphs.

**Proposition 57.** *Pre-ALPS graphs are preordered by the relation  $\preceq$  defined as:*

$$\mathbb{G}_1 \preceq \mathbb{G}_2 \iff G_1 \preceq G_2 \text{ and } \forall i \in I_\tau. \ell_1(i) \in nl_1 \Rightarrow \ell_2(i) \in nl_2 \text{ and } \forall i, j \in I_\tau. \{\ell_1(i), \ell_1(j)\} \in sh_1 \Rightarrow \{\ell_2(i), \ell_2(j)\} \in sh_2 .$$

*Proof.* Reflexivity is trivial. Assume now  $\mathbb{G}_1 \preceq \mathbb{G}_2$  and  $\mathbb{G}_2 \preceq \mathbb{G}_3$ . By definition  $G_1 \preceq G_2$  and  $G_2 \preceq G_3$  and since  $\preceq$  is a preorder on aliasing graphs, we have that  $G_1 \preceq G_3$ . Moreover,  $\ell_1(i) \in nl_1 \Rightarrow \ell_2(i) \in nl_2, \ell_2(i) \in nl_2 \Rightarrow \ell_3(i) \in nl_3$  and therefore  $\ell_1(i) \in nl_1 \Rightarrow \ell_3(i) \in nl_3$ . Analogously, we have  $\{\ell_1(i), \ell_1(j)\} \in sh_1 \Rightarrow \{\ell_3(i), \ell_3(j)\} \in sh_3$ .  $\square$

We now prove some properties of  $cl^\uparrow$  and  $red$ , and of their interaction.

**Proposition 104.** (Closures of pre-ALPS graphs). *The operators  $cl^\uparrow$  and  $red$  are an upper and lower closure operator, respectively. Moreover, if  $\mathbb{G}$  is closed w.r.t.  $red$ , then  $cl^\uparrow(\mathbb{G})$  is closed w.r.t.  $red$ , too.*

*Proof.* The fact that  $red$  and  $cl^\uparrow$  are closure operators is immediate from their definition. It remains to prove that  $cl^\uparrow$  preserves closedness w.r.t.  $red$ .

First of all, every set  $\{n\}$  is G-SH-compatible with any type environment, since  $\kappa \in C(\kappa)$ . Moreover, if there is a nonempty loop involving node  $n$ , then  $n \xrightarrow{f} m$  and there is a path from  $m$  to  $n$ . By Proposition 95, this means that  $\tau_G(n) \in C(\tau_G(m))$  and by definition of  $\tau_G$ ,  $\tau_G(m) \leq \tau_G(n).f$ . Hence,  $\tau_G(n) \in C(\tau_G(n).f)$ ,  $\tau_G(n) \in NL$  and  $n$  is G-NL-compatible.

Assume now that  $\{n, m\}$  is G-SH-compatible and  $n' \xrightarrow{f} n$ . Then  $C(\tau_G(n)) \cap C(\tau_G(m)) \neq \emptyset$  and by Proposition 95,  $\tau_G(n) \in C(\tau_G(n'))$ , hence  $C(\tau_G(n')) \supseteq C(\tau_G(n))$ . Therefore,  $C(\tau_G(n)) \cap C(\tau_G(m)) \neq \emptyset$  implies  $C(\tau_G(n')) \cap C(\tau_G(m)) \neq \emptyset$ , that is,  $\{n', m\}$  is G-SH-compatible.

Now, assume  $\{m_1, m_2\}$  is G-SH-compatible,  $n \xrightarrow{f_1} m_1$  and  $n \xrightarrow{f_2} m_2$  with  $f_1 \neq f_2$ . Then,  $(\tau_G(m_1), \tau_G(m_2)) \in SH$  and therefore  $C(\tau_G(m_1)) \cap C(\tau_G(m_2)) \neq \emptyset$ . Moreover, by definition of  $\tau_G$ ,  $\tau_G(m_1) \leq \tau_G(n).f_1$ ,  $\tau_G(m_2) \leq \tau_G(n).f_2$ . Therefore, since  $C$  is downward closed, we have  $C(\tau_G(n).f_1) \cap C(\tau_G(n).f_2) \neq \emptyset$ , i.e.,  $\tau_G(n) \in NL$ .

Finally, assume that  $n$  is G-NL-compatible. This means  $\tau_G(n) \in NL$ . In particular, there exists  $\kappa$  such that  $\kappa \leq \tau_G(n)$  and  $\kappa$  is in the least solution of the equation in Definition 33 of NL. When  $n' \xrightarrow{f} n$  then  $\kappa \leq \tau_G(n) \leq \tau_G(n').f$ , i.e.,  $\tau_G(n') \rightarrow \kappa$  which implies  $\tau_G(n') \in NL$ .  $\square$

**A.3.1 Projections and propagation of nullness**

**Proposition 64.** *If  $\mathbb{G} \in \text{ALPS}_\tau$  and  $X \subseteq N$  is backward closed, then  $\mathbb{G}|_X \in \text{ALPS}_\tau$ .*

*Proof.* It is immediate to check that  $\mathbb{G}|_X$  is closed and that  $red(\mathbb{G}|_X) = \mathbb{G}|_X$  (this holds even if  $X$  is not backward closed). Moreover, since  $X$  is backward closed, then  $\mathbb{G}|_X$  is an aliasing graph and not just a pre-aliasing graph.  $\square$

**Proposition 65.** *If  $\mathbb{G}$  is a pre-ALPS graph and  $X \subseteq N$  is backward closed, then  $\mathbb{G}|_X \preceq \mathbb{G}$ .*

*Proof.* Let  $\mathbb{G}' = \mathbb{G}|_X$ . We know that  $\mathbb{G}|_X \preceq \mathbb{G}$  by Proposition 51. Now, given the identifier  $i$ , assume  $\ell'(i) \in nl'$ . This means  $\ell'(i) = n \in X$  and  $n \in nl'$ . By definition,  $\ell(i) = n$  and  $n \in nl$ . Similarly for the sharing component.  $\square$

A.3.2 Up- and down-closures of Pre-ALPS graphs

We introduce a new preorder on ALPS graphs which is apparently different from the one defined in the main part of the paper. The new definition is based on the concept of morphisms of aliasing graphs, and it is easier to use than the one in Proposition 57, although we will prove them to be equivalent.

**Proposition 105.** *Pre-ALPS graphs are preordered by*

$$\mathbb{G}_1 \widetilde{\preceq} \mathbb{G}_2 \iff \exists h : G_2 \rightarrow G_1 \text{ s.t. } h^{-1}(sh_1) \subseteq sh_2 \text{ and } h^{-1}(nl_1) \subseteq nl_2,$$

where

$$h^{-1}(nl_1) = \{n \in N_2 \mid h(n) \in nl_1\},$$

$$h^{-1}(sh_1) = \{\{n, m\} \in \mathcal{P}_2(N_2) \mid \{h(n), h(m)\} \in sh_1\}.$$

*Proof.* We show that  $\widetilde{\preceq}$  is reflexive. Given a pre-ALPS graph  $\mathbb{G}$ , the only morphism  $h : G \rightarrow G$  is the identity. Then  $h^{-1}(sh) = sh$  and  $h^{-1}(nl) = nl$ , hence  $\mathbb{G} \widetilde{\preceq} \mathbb{G}$ . Assume now  $\mathbb{G}_1 \widetilde{\preceq} \mathbb{G}_2$  and  $\mathbb{G}_2 \widetilde{\preceq} \mathbb{G}_3$  with  $h_2 : G_3 \rightarrow G_2$  and  $h_1 : G_2 \rightarrow G_1$ . We have that  $h = h_1 \circ h_2 : G_3 \rightarrow G_1$ . Moreover,  $h^{-1}(sh_1) = h_2^{-1}(h_1^{-1}(sh_1)) \subseteq h_2^{-1}(sh_2) \subseteq sh_3$ . Analogously, we have  $h^{-1}(nl_1) \subseteq nl_3$ .  $\square$

**Proposition 106.** *Given pre-ALPS graphs  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , we have  $\mathbb{G}_1 \widetilde{\preceq} \mathbb{G}_2$  iff  $\mathbb{G}_1 \preceq \mathbb{G}_2$ .*

*Proof.* We prove the two implications of the equivalence separately.

$\Rightarrow$ ) By Proposition 105 and Definition 41 there exists  $h : G_2 \rightarrow G_1$  such that  $\ell_1 = h \circ \ell_2$ ,  $h^{-1}(sh_1) \subseteq sh_2$  and  $h^{-1}(nl_1) \subseteq nl_2$ . Applying  $\ell_2^{-1}$  to both sides of set inequalities, we have  $\ell_1^{-1}(sh_1) = \ell_2^{-1}(h^{-1}(sh_1)) \subseteq \ell_2^{-1}(sh_2)$  and  $\ell_1^{-1}(nl_1) = \ell_2^{-1}(h^{-1}(nl_1)) \subseteq \ell_2^{-1}(nl_2)$ .

$\Leftarrow$ ) By Propositions 57 and 42, there exists  $h : G_2 \rightarrow G_1$  such that  $\ell_1 = h \circ \ell_2$ . Note that  $\ell_1^{-1}(sh_1) = \ell_2^{-1}(h^{-1}(sh_1)) \subseteq \ell_2^{-1}(sh_2)$ . Since  $\ell_2$  is surjective,  $\ell_2 \circ \ell_2^{-1} = id$ . Therefore, by applying  $\ell_2$  to both sides, we have  $h^{-1}(sh_1) \subseteq sh_2$ . Analogously for  $nl_1$  and  $nl_2$ .  $\square$

**Theorem 68.** *Given a pre-ALPS graph  $\mathbb{G} = G \star sh \star nl$ , the down-closure  $cl^\downarrow(\mathbb{G})$  can be computed as follows. Let  $sh^* \star nl^*$  be the greatest pair, under the component-wise ordering, such that*

- (1)  $nl^* = nl \setminus \{n \mid m \notin nl^* \wedge m \xrightarrow{f} n \in E\}$ ;
- (2)  $sh^* = sh \setminus \{\{m_1, m_2\} \mid n \notin nl^*, n \xrightarrow{f_1} m_1 \in E, n \xrightarrow{f_2} m_2 \in E, f_1 \neq f_2\} \setminus \{\{n, m\} \mid \{n', m\} \notin sh^* \wedge n' \xrightarrow{f} n \in E\}$ .

Then, we have that

$$cl^\downarrow(\mathbb{G}) = (G \star sh^* \star nl^*)|_{N \setminus X}$$

where  $X = \{n \mid n \notin nl^*, \text{ there is a loop in } G \text{ such that } n \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E\} \cup \{n \mid \{n\} \notin sh^*\}$ . Moreover, if  $\mathbb{G}$  is closed w.r.t. red, then  $cl^\downarrow(\mathbb{G})$  is an ALPS graph.



*Proof.* The following holds.

- (1) First of all, observe that  $nl^* = \bigcap_{i \geq 0} nl_i$ , where  $nl_0 = nl$  and for  $i \geq 0$ ,  $nl_{i+1} = nl_i \setminus \{n \mid m \notin nl_i \wedge m \xrightarrow{f} n \in E\}$ . We first show that  $nl^* = nl^\dagger$ , where  $nl^\dagger = N \setminus \{n \mid m \notin nl \wedge m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E, k \geq 0\}$ . We prove the two inclusions separately.

$nl^\dagger \subseteq nl^*$ ) The proof is by contradiction. Let us assume that  $nl^\dagger \not\subseteq nl^*$ . Since  $nl^\dagger \subseteq nl$ , there exists  $j \geq 0$  such that  $nl^\dagger \not\subseteq nl_{j+1}$  and  $nl^\dagger \subseteq nl_j$ . Therefore, there exist  $n \in nl^\dagger$  and  $j \geq 0$  such that  $n \in nl_j$  and  $n \notin nl_{j+1}$ . By definition of  $nl_{j+1}$ , there exists  $m \notin nl_j$  such that  $m \xrightarrow{f} n \in E$ . Since by hypothesis  $nl^\dagger \subseteq nl_j$ , we have that  $m \notin nl^\dagger$  and, by definition of  $nl^\dagger$ , there exists  $m' \xrightarrow{f_1} \dots \xrightarrow{f_k} m \in E$  such that  $m' \notin nl$ . Therefore, there exists  $m' \xrightarrow{f_1} \dots \xrightarrow{f_k} m \xrightarrow{f} n \in E$  such that  $m' \notin nl$ . By definition of  $nl^\dagger$ ,  $n \notin nl^\dagger$  and this contradicts the hypothesis.

$nl^* \subseteq nl^\dagger$ ) Let  $n \notin nl^\dagger$ . By definition there exists  $m \notin nl$  such that  $m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E$ . We prove by induction on  $k$  that  $n \notin nl^*$ .

-  $k = 0$ ). In this case  $n = m \notin nl$  and therefore  $n \notin nl^*$ .

-  $k > 0$ ). In this case  $m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E$  and  $m \notin nl$ . By inductive hypothesis  $n' \notin nl^*$  and  $n' \xrightarrow{f_k} n \in E$ . Therefore, by definition of  $nl^*$ ,  $n \notin nl^*$  and then the thesis.

- (2) First of all, observe that by the previous result, it holds that

$$sh^* = S \setminus \{\{n, m\} \mid \{n', m\} \notin sh^* \wedge n' \xrightarrow{f} n \in E\}$$

$$S = sh \setminus \{\{m_1, m_2\} \mid m \notin nl \wedge m \xrightarrow{g_1} \dots \xrightarrow{g_k} n \in E, k \geq 0,$$

$$n \xrightarrow{f_1} m_1 \in E, n \xrightarrow{f_2} m_2 \in E, f_1 \neq f_2\}$$

and therefore  $sh^* = \bigcap_{i \geq 0} sh_i$ , where  $sh_0 = S$  and for  $i \geq 0$ ,  $sh_{i+1} = sh_i \setminus \{\{n, m\} \mid \{n', m\} \notin sh_i \wedge n' \xrightarrow{f} n \in E\}$ . We show that  $sh^* = sh^\dagger$  where

$$sh^\dagger = sh \setminus \{\{m_1, m_2\} \mid m \notin nl, m \xrightarrow{f_1} \dots \xrightarrow{f_k} m_1 \in E,$$

$$m \xrightarrow{g_1} \dots \xrightarrow{g_h} m_2 \in E, k \leq h, f_1 \dots f_k \neq g_1 \dots g_k\}$$

$$\setminus \{\{n, m\} \mid \{n', m\} \notin sh \wedge n' \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E\}$$

We prove the two inclusions separately.

$sh^\dagger \subseteq sh^*$ ) The proof is by contradiction. Let us assume that  $sh^\dagger \not\subseteq sh^*$ . Since by definition  $sh^\dagger \subseteq S = sh_0$  there exists  $j \geq 0$  such that  $sh^\dagger \subseteq sh_j$  and  $sh^\dagger \not\subseteq sh_{j+1}$ . Therefore, there exist  $\{n, m\} \in sh^\dagger$  and  $j \geq 0$  such that  $\{n, m\} \in sh_j$  and  $\{n, m\} \notin sh_{j+1}$ . By definition of  $sh_{j+1}$ , there exists  $\{n', m\} \notin sh_j$  such that  $n' \xrightarrow{f} n \in E$ . Since by hypothesis  $sh^\dagger \subseteq sh_j$ , we have that  $\{n', m\} \notin sh^\dagger$ . By definition of  $sh^\dagger$ , we have the following possibilities:

- a.  $p \notin nl$ ,  $p \xrightarrow{f_1} \dots \xrightarrow{f_k} n' \in E$  and  $p \xrightarrow{g_1} \dots \xrightarrow{g_h} m \in E$  and  $f_1 \dots f_y \neq g_1 \dots g_y$ , where  $y$  is the minimum between  $k$  and  $h$ . Since  $n' \xrightarrow{f} n \in E$ , we have that  $p \xrightarrow{f_1} \dots \xrightarrow{f_k} n' \xrightarrow{f} n \in E$ . Moreover,  $f_1 \dots f_x \neq g_1 \dots g_x$ , where  $x = y$  if  $k \geq h$  and  $x = y + 1$  if  $k < h$ . By definition  $\{n, m\} \notin sh^\dagger$  and then we have a contradiction.

- b. there exists  $\{p, m\} \notin sh$  such that  $p \xrightarrow{f_1} \dots \xrightarrow{f_k} n' \in E$ . In this case  $p \xrightarrow{f_1} \dots \xrightarrow{f_k} n' \xrightarrow{f} n \in E$ . Therefore,  $\{n, m\} \notin sh^\dagger$  and this contradicts the hypothesis.

$sh^* \subseteq sh^\dagger$ ) Let  $\{n, m\} \notin sh^\dagger$ . By definition we have the following possibilities:

- a.  $\{n, m\} \notin sh$ . In this case  $\{n, m\} \notin sh^*$ .

- b. there exists  $\{n', m\} \notin sh$  such that  $n' \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E$ . We prove by induction on  $k$  that  $\{n, m\} \notin sh^*$ .  
 $k = 0$ ) In this case  $n' = n$  and  $\{n, m\} \notin sh$ . Therefore  $\{n, m\} \notin sh^*$ .  
 $k > 1$ )  $n' \xrightarrow{f_1} \dots \xrightarrow{f_{k-1}} n'' \xrightarrow{f_k} n \in E$ . By inductive hypothesis  $\{n'', m\} \notin sh^*$  and then, by definition of  $sh^*$ ,  $\{n, m\} \notin sh^*$ .
- c. there exists  $m' \notin nl$ , such that  $m' \xrightarrow{f_1} \dots \xrightarrow{f_k} m \in E$ ,  $m' \xrightarrow{g_1} \dots \xrightarrow{g_h} n \in E$ ,  $k \leq h$   
 $f_1 \dots f_k \neq g_1 \dots g_k$ . Let  $i \geq 1$  be the first index such that  $f_i \neq g_i$ . Then  $m' \xrightarrow{f_1} \dots \xrightarrow{f_{i-1}} n_{i-1} \in E$ ,  $n_{i-1} \xrightarrow{f_i} n_i \dots \xrightarrow{f_k} m \in E$ ,  $n_{i-1} \xrightarrow{g_i} n'_i \dots \xrightarrow{g_h} n \in E$  and  $f_i \neq g_i$ .  
 By previous result  $n_{i-1} \notin nl^{\dagger} = nl^*$  and therefore, by definition of  $sh^*$ ,  $\{n_i, n'_i\} \notin sh^*$ . Now, the proof follows by a straightforward inductive argument.

By previous results, it holds that

$$\begin{aligned}
 X &= \{n \mid m \notin nl \wedge m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E, k \geq 0 \\
 &\quad \text{and there is a loop in } G \text{ such that } n \xrightarrow{g_1} \dots \xrightarrow{g_h} n \in E\} \\
 &\cup \{n \in N \mid \{n\} \notin sh\} \\
 &= \{n \mid m \notin nl, m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E, \\
 &\quad m \xrightarrow{g_1} \dots \xrightarrow{g_h} n \in E, k \leq h, f_1 \dots f_k \neq g_1 \dots g_k\} \\
 &\cup \{n \mid \{n', n\} \notin sh \wedge n' \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E\}
 \end{aligned}$$

Now, observe that since  $\mathbb{G}$  is a pre-ALPS graph, then  $G \star sh^* \star nl^*$  is also a pre-ALPS graph and by construction,  $(G \star sh^* \star nl^*) \leq \mathbb{G}$ . Moreover, since  $N \setminus \overrightarrow{X} \subseteq N$  backward closed, we have that  $\mathbb{G}'$  is a pre-ALPS graph. Then, by Proposition 65  $\mathbb{G}' \leq (G \star sh^* \star nl^*) \leq \mathbb{G}$ .

Now, we prove that  $\mathbb{G}'$  is closed. We prove that all conditions in the Definition 60 are respected.

$n \in N' \Rightarrow \{n\} \in sh'$ ) The proof is by contradiction. Let us assume that there exists  $n \in N'$  such that  $\{n\} \notin sh'$ . Then  $n \notin \overrightarrow{X}$  and  $\{n\} \notin sh^*$ . Now, we have a contradiction, since  $X \supseteq \{n \mid \{n\} \notin sh^*\}$ .  
**There is a loop in G involving  $n \Rightarrow n \in nl$ .** Let us assume that there is a nonempty loop in  $G'$  involving  $n$  such that  $n \notin nl'$ . In this case, there is a nonempty loop in  $G$  involving  $n$  such that  $n \notin \overrightarrow{X}$  and  $n \notin nl^*$ . Now, we have a contradiction, since  $X \supseteq \{n \mid n \notin nl^*\}$ , there is a loop in  $G$  such that  $n \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E$ .

$\mathbb{G}'$  is partially closed. We check the three conditions in the definition of partial closure.

$\{n, m\} \in sh' \wedge n' \xrightarrow{f} n \in E' \Rightarrow \{n', m\} \in sh'$ ) Assume that  $\{n, m\} \in sh'$  and  $n' \xrightarrow{f} n \in E'$ . Then  $n, m, n' \notin \overrightarrow{X}$ ,  $\{n, m\} \in sh^*$  and  $n' \xrightarrow{f} n \in E$ . By definition of  $sh^*$ ,  $\{n', m\} \in sh^*$  and since  $n', m \notin \overrightarrow{X}$ , we have that  $\{n', m\} \in sh'$ .

$n \xrightarrow{f_1} m_1 \in E', n \xrightarrow{f_2} m_2 \in E', f_1 \neq f_2, \{m_1, m_2\} \in sh' \Rightarrow n \in nl'$ ) Assume that  $n \xrightarrow{f_1} m_1 \in E'$ ,  $n \xrightarrow{f_2} m_2 \in E'$ ,  $f_1 \neq f_2$ ,  $\{m_1, m_2\} \in sh'$ . Then  $n, m_1, m_2 \notin \overrightarrow{X}$  and  $n \xrightarrow{f_1} m_1 \in E$ ,  $n \xrightarrow{f_2} m_2 \in E$ ,  $f_1 \neq f_2$ ,  $\{m_1, m_2\} \in sh^*$ . By definition of  $sh^*$ ,  $n \in nl^*$  and since  $n \notin \overrightarrow{X}$ , we have that  $n \in nl'$ .

$n \in nl' \wedge n' \xrightarrow{f} n \in E' \Rightarrow n' \in nl'$ ) Assume that  $n \in nl'$  and  $n' \xrightarrow{f} n \in E'$ . Then  $n' \notin \overrightarrow{X}$ ,  $n \in nl^*$  and  $n' \xrightarrow{f} n \in E$ . By definition of  $nl^*$ ,  $n' \in nl^*$  and since  $n' \notin \overrightarrow{X}$ , we have that  $n' \in nl'$ .

By Proposition 65, since  $\mathbb{G}$  is a pre-ALPS graph and  $N \setminus \overrightarrow{X} \subseteq N$  is backward closed, we have that  $\mathbb{G}' \leq \mathbb{G}$ .

Now, we have to prove that  $\mathbb{G}'$  is the greatest pre-ALPS graph smaller than  $\mathbb{G}$  and such that  $\mathbb{G}'$  is closed. Let  $\mathbb{G}_1$  be a pre-ALPS graph smaller than  $\mathbb{G}$  such that  $\mathbb{G}_1$  is closed. We have to prove that  $\mathbb{G}_1 \preceq \mathbb{G}'$ .

Since  $\mathbb{G}_1 \preceq \mathbb{G}$ , by Propositions 106 and 105, there exists  $h_1 : G \rightarrow G_1$  such that  $h_1^{-1}(sh_1) \subseteq sh$  and  $h_1^{-1}(nl_1) \subseteq nl$ .

Let  $h'_1 : G' \rightarrow G_1$  such that for each  $n \in N'$   $h'_1(n) = h_1(n)$ . We prove that  $h'_1$  is a morphism from  $G'$  to  $G_1$ ,  $h'_1{}^{-1}(sh_1) \subseteq sh^\dagger$  and  $h'_1{}^{-1}(nl_1) \subseteq nl'$ . Therefore, the thesis follows by Propositions 105 and 106.

**$h'_1$  is a morphism from  $G'$  to  $G_1$ .** Let  $i \in I_\tau$ . We have to prove that  $h'_1(\ell'(i)) = \ell_1(i)$ . The following holds:

- $\ell'(i) \neq \perp$ . In this case  $h'_1(\ell'(i)) = h_1(\ell(i)) = \ell_1(i)$ .
- $\ell'(i) = \perp$  and  $\ell_1(i) = \perp$ . In this case  $h'_1(\ell'(i)) = \perp = \ell_1(i)$  and then the thesis.
- $\ell'(i) = \perp$  and  $\ell_1(i) = h_1(\ell(i)) \neq \perp$ . In this case, since  $\mathbb{G}_1 \preceq \mathbb{G}$ , we have that  $\ell(i) \neq \perp$  and therefore  $\ell(i) \in \bar{X}$ . Therefore there exists  $j \in X$  such that  $\ell'(j) = \perp$ ,  $\ell(j) \neq \perp$  and  $\ell_1(j) = h_1(\ell(j)) \neq \perp$ . By definition of  $X$  one of the following holds.
  - $\ell(j) = n$ , there exists  $m \notin nl$  such that  $m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E$ ,  $k \geq 0$  and there is a loop in  $G$  such that  $n \xrightarrow{g_1} \dots \xrightarrow{g_h} n \in E$ . In this case, since  $\mathbb{G}_1 \preceq \mathbb{G}$ ,  $\ell_1(j) = h_1(\ell(j)) \neq \perp$  and by Lemma 96, we have that  $h_1(m) \notin nl_1$ ,  $h_1(m) \xrightarrow{f_1} \dots \xrightarrow{f_k} h_1(n) \in E_1$ ,  $k \geq 0$  and there is a loop in  $G_1$  such that  $h_1(n) \xrightarrow{g_1} \dots \xrightarrow{g_h} h_1(n) \in E_1$ . Since by hypothesis  $\mathbb{G}_1$  is closed, by Point 5 of Definition 60 and by a straightforward inductive argument, we have that  $h_1(n) \notin nl_1$ . Therefore, by Point 2 of Definition 60, we have a contradiction, since there is a loop in  $\mathbb{G}_1$  involving  $h_1(n)$  and  $h_1(n) \notin nl_1$ .
  - $\ell(j) \in \{n \in N \mid \{n\} \not\subseteq sh\}$ . In this case,  $h_1(\{\ell(j)\}) \not\subseteq sh_1$ . By Point 1 of Definition 60,  $h_1(\ell(j)) = \ell_1(j) \notin N_1$  and this contradicts the hypothesis.
  - $\ell(j) = n$ , and there exists  $m \notin nl$ , such that  $m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E$ ,  $m \xrightarrow{g_1} \dots \xrightarrow{g_h} n \in E$ ,  $k \leq h$  and  $f_1 \dots f_k \neq g_1 \dots g_k$ . In this case, since  $\mathbb{G}_1 \preceq \mathbb{G}'$ ,  $h_1(n) \neq \perp$ ,  $h_1(m) \notin nl_1$  and by Lemma 96,  $h_1(m) \xrightarrow{f_1} \dots \xrightarrow{f_k} h_1(n) \in E_1$ ,  $h_1(m) \xrightarrow{g_1} \dots \xrightarrow{g_h} h_1(n) \in E_1$ ,  $k \leq h$  and  $f_1 \dots f_k \neq g_1 \dots g_k$ . Let  $i \geq 1$  be the first index such that  $f_i \neq g_i$ . Therefore, we have that  $h_1(m) \xrightarrow{f_1} \dots \xrightarrow{f_{i-1}} h_1(n_{i-1}) \in E_1$ ,  $h_1(n_{i-1}) \xrightarrow{f_i} h_1(n_i) \dots \xrightarrow{f_k} h_1(n) \in E_1$ ,  $h_1(n_{i-1}) \xrightarrow{g_i} h_1(n'_i) \dots \xrightarrow{g_h} h_1(n) \in E_1$  and  $f_i \neq g_i$ . Since  $\mathbb{G}_1$  is closed, by Point 2 of Definition 60 and by a straightforward inductive argument, we have that  $h_1(n_{i-1}) \notin nl_1$ . Therefore, since  $\mathbb{G}_1$  is closed and by Point 4 of Definition 60, we have that  $\{n_i, n'_i\} \not\subseteq sh_1$ . Now, since  $\mathbb{G}_1$  is closed, by Point 3 of Definition 60 and by a straightforward inductive argument,  $\{h_1(n)\} \not\subseteq sh_1$ . By Point 1 of Definition 60  $h_1(n) = \ell_1(j) \notin N_1$  and this contradicts the hypothesis.
  - $\ell(j) = n$  and there exists  $\{n', n\} \not\subseteq sh$  such that  $n' = n_0 \xrightarrow{f_1} n_1 \xrightarrow{f_2} \dots n_{k-1} \xrightarrow{f_k} n_k = n \in E$ . In this case, since  $\mathbb{G}_1 \preceq \mathbb{G}'$ ,  $\{h_1(n'), h_1(n)\} \not\subseteq sh_1$ . Moreover, since  $h_1(n) \neq \perp$  and by Lemma 96,  $h_1(n') = h_1(n_0) \xrightarrow{f_1} h_1(n_1) \dots h_1(n_{k-1}) \xrightarrow{f_k} h_1(n_k) = h_1(n) \in E_1$ . Therefore, there exists  $l \in [1, n]$  such that  $\{h_1(n_{l-1}), h_1(n_l)\} \not\subseteq sh_1$  and  $h_1(n_{l-1}) \xrightarrow{f_l} h_1(n_l)$ . Now, by Point 3 of Definition 60  $\{h_1(n_l)\} \not\subseteq sh_1$ . If  $l = k$ , then we have a contradiction, since by Point 1 of Definition 60,  $h_1(n_k) = \ell_1(j) \notin N_1$ . Otherwise  $l < k$  and  $h_1(n_l) \xrightarrow{f_{l+1}} h_1(n_{l+1}) \in E_1$ . Since  $\{h_1(n_l)\} \not\subseteq sh_1$ , by Point 3 of Definition 60,  $\{h_1(n_l), h_1(n_{l+1})\} \not\subseteq sh_1$ . Now the proof follows by a straightforward inductive argument.

$h_1^{-1}(nl_1) \subseteq n'$ ) Assume that there exists  $n \in h_1^{-1}(nl_1)$  such that  $n \notin n'$ . Since  $\mathbb{G}_1 \preceq \mathbb{G}$  and by definition of  $h_1'$ , we have that  $n \in h_1^{-1}(nl_1) \subseteq N$ ,  $n \in N'$  and  $n \notin n'$ . By definition of  $nl^\dagger$ , there exists  $m \notin nl$  such that  $m \xrightarrow{f_1} \dots \xrightarrow{f_k} n \in E$ . Then, since  $\mathbb{G}_1 \preceq \mathbb{G}$ ,  $h_1(n) \neq \perp$  and by Lemma 96, we have that  $h_1(m) \notin nl_1$  and  $h_1(m) \xrightarrow{f_1} \dots \xrightarrow{f_k} h_1(n) \in E_1$ . Since  $\mathbb{G}_1$  is closed and by a straightforward inductive argument we have that  $h_1(n) = h_1'(n) \notin nl_1$  and this contradicts the hypothesis.

$h_1^{-1}(sh_1) \subseteq sh'$ ) Assume that there exists  $\{m_1, m_2\} \in h_1^{-1}(sh_1)$  such that  $\{m_1, m_2\} \notin sh'$ . Since  $\mathbb{G}_1 \preceq \mathbb{G}$  and by definition of  $h_1'$ , we have that  $\{m_1, m_2\} \in h_1^{-1}(sh_1) \subseteq sh$ ,  $m_1, m_2 \in N'$  and  $\{m_1, m_2\} \notin sh'$ . By definition of  $sh^\dagger$ , the following holds:

- (1) there exists  $m \notin nl$ ,  $m \xrightarrow{f_1} \dots \xrightarrow{f_k} m_1 \in E$ ,  $m \xrightarrow{g_1} \dots \xrightarrow{g_h} m_2 \in E$ ,  $k \leq h$ ,  $f_1 \dots f_k \neq g_1 \dots g_h$ . Then, since  $\mathbb{G}_1 \preceq \mathbb{G}$ ,  $h_1(m_1) \neq \perp$ ,  $h_1(m_2) \neq \perp$  and by Lemma 96, we have that  $h_1(m) \notin nl_1$  and  $h_1(m) \xrightarrow{f_1} \dots \xrightarrow{f_k} h_1(m_1) \in E_1$ ,  $h_1(m) \xrightarrow{g_1} \dots \xrightarrow{g_h} h_1(m_2) \in E_1$ ,  $k \leq h$ ,  $f_1 \dots f_k \neq g_1 \dots g_h$ . Let  $i \geq 1$  be the first index such that  $f_i \neq g_i$ . Therefore, we have that  $h_1(m) \xrightarrow{f_1} \dots \xrightarrow{f_{i-1}} h_1(n_{i-1}) \in E_1$ ,  $h_1(n_{i-1}) \xrightarrow{f_i} h_1(n_i) \dots \xrightarrow{f_k} h_1(m_1) \in E_1$ ,  $h_1(n_{i-1}) \xrightarrow{g_i} h_1(n'_i) \dots \xrightarrow{g_h} h_1(m_2) \in E_1$  and  $f_i \neq g_i$ . Analogously to the previous case  $h_1(n_{i-1}) \notin nl_1$ . Therefore, since  $\mathbb{G}_1$  is closed and by Point 4 of Definition 60, we have that  $\{h_1(n_i), h_1(n'_i)\} \notin sh_1$ . Now, since  $\mathbb{G}_1$  is closed, by Point 3 of Definition 60 and by a straightforward inductive argument, we have that  $\{h_1(m_1), h_1(m_2)\} \notin sh_1$  and this contradicts the hypothesis.
- (2) there exists  $\{n', m_2\} \notin sh$  such that  $n' \xrightarrow{f_1} \dots \xrightarrow{f_k} m_1 \in E$ . Then, since  $\mathbb{G}_1 \preceq \mathbb{G}$ ,  $h_1(m_1) \neq \perp$ ,  $h_1(m_2) \neq \perp$  and by Lemma 96, we have that  $\{h_1(n'), h_1(m_2)\} \notin sh_1$  and  $h_1(n') \xrightarrow{f_1} \dots \xrightarrow{f_k} h_1(m_1) \in E_1$ . Now, since  $\mathbb{G}_1$  is closed, by Point 3 of Definition 60 and by a straightforward inductive argument, we have that  $\{h_1(m_1), h_1(m_2)\} \notin sh_1$  and this contradicts the hypothesis.

Finally, we have to prove that if  $\mathbb{G}$  is closed w.r.t. red then  $cl^\downarrow(\mathbb{G})$  is a ALPS graph. By previous result, we have only to prove that  $cl^\downarrow(\mathbb{G})$  is closed w.r.t. red. The following holds:

$sh'$  is  $G'$ -SH-compatible. Assume  $\{n, m\} \in sh'$ . By construction,  $\tau_{G'}(n) = \tau_G(n)$ ,  $\tau_{G'}(m) = \tau_G(m)$  and  $\{n, m\} \in sh$ . Since by hypothesis  $\mathbb{G} = G \star sh \star nl$  is a pre-ALPS, we have that  $G$  graph is an aliasing graph and therefore  $(\tau_{G'}(n), \tau_{G'}(m)) = (\tau_G(n), \tau_G(m)) \in SH$ .

$nl'$  is  $G'$ -NL-compatible. The proof is analogous to the previous one. □

### A.3.3 The lattice of ALPS graphs

Among the clauses defining the operation  $cl^\uparrow$  on pre-aliasing graphs, clauses 3, 4, and 5 of Definition 60 enjoy some special properties, since closure w.r.t. these clauses is preserved by counter-image of graph morphisms. This is formally stated by the following definition and lemma.

**Definition 107.** We say that a pre-ALPS graph  $\mathbb{G}$  is partially closed when

- $\{n, m\} \in sh \wedge n' \xrightarrow{f} n \Rightarrow \{n', m\} \in sh$ ;
- $n \xrightarrow{f_1} m_1, n \xrightarrow{f_2} m_2, f_1 \neq f_2, \{m_1, m_2\} \in sh \Rightarrow n \in nl$ ;
- $n \in nl \wedge n' \xrightarrow{f} n \Rightarrow n' \in nl$ .

**Lemma 108.** *Let  $G_1$  and  $G_2$  be aliasing graphs, with  $h : G_2 \rightarrow G_1$ . The following properties hold:*

- if  $\{n_1, m_1\} \subseteq N_1$  is  $G_1$ -SH-compatible, then  $h^{-1}(\{n_1, m_1\})$  is  $G_2$ -SH-compatible;
- if  $n_1 \in N_1$  is  $G_1$ -NL-compatible, then  $h^{-1}(n_1)$  is  $G_2$ -NL-compatible;
- if  $G_1 \star sh_1 \star nl_1$  is partially closed, then  $G_2 \star h^{-1}(sh_1) \star h^{-1}(nl_1)$  is partially closed, too.

*Proof.* For the first point, assume  $\{n_1, m_1\}$  is  $G_1$ -SH-compatible and  $\{n_2, m_2\} \in h^{-1}(\{n_1, m_1\})$ . By Lemma 96,  $\tau_{G_2}(n_2) \geq \tau_{G_1}(n_1)$  and  $\tau_{G_2}(m_2) \geq \tau_{G_1}(m_1)$ . This implies that  $\{n_2, m_2\}$  is  $G_2$ -SH-compatible. The proof for the second point is similar.

For the third point, the following holds:

- Assume that  $\{n, m\} \in h^{-1}(sh_1)$  and  $n' \xrightarrow{f} n \in E_2$ . Then  $\{h(n), h(m)\} \in sh_1$  and by Lemma 96,  $h(n') \xrightarrow{f} h(n) \in E_1$ . Since  $sh_1 \star nl_1$  is partially closed this implies  $\{h(n'), h(m)\} \in sh_1$ , hence  $\{n', m\} \in h^{-1}(sh_1)$ .
- Now, assume  $n \xrightarrow{f_1} m_1, n \xrightarrow{f_2} m_2 \in E_2, f_1 \neq f_2$  and  $\{m_1, m_2\} \in h^{-1}(sh_1)$ . Then  $\{h(m_1), h(m_2)\} \in sh_1$  and by Lemma 96,  $h(n) \xrightarrow{f_1} h(m_1), h(n) \xrightarrow{f_2} h(m_2) \in E_1$ . Since  $sh_1 \star nl_1$  is partially closed this implies  $h(n) \in nl_1$ , hence  $n \in h^{-1}(nl_1)$ .
- Finally, assume  $n \in h^{-1}(nl_1)$  and  $n' \xrightarrow{f} n \in E_2$ . Then  $h(n) \in nl_1$  and by Lemma 96,  $h(n') \xrightarrow{f} h(n) \in E_1$ . Since  $sh_1 \star nl_1$  is partially closed, this means  $h(n') \in nl_1$  and  $n' \in h^{-1}(nl_1)$ . □

**Lemma 109.** *Let  $G_1$  and  $G_2$  be ALPS graphs. Then  $G_1 \vee G_2$  is an ALPS graph.*

*Proof.* Let  $G = G_1 \vee G_2$ . The proof that  $G$  is an aliasing graph and that for  $k = 1, 2, G_k \leq G$  follows by Theorem 101. Let  $h_1 : G \rightarrow G_1, h_2 : G \rightarrow G_2$  be the corresponding morphisms. We prove that all conditions in the definition of ALPS graph are respected.

**sh is G-SH-compatible.** If  $\{n, m\} \in sh, \exists k \in \{1, 2\}$  such that  $\{h_k(n), h_k(m)\} \in sh_k$ . Now the proof follows by the first point of Lemma 108 and since by hypothesis  $G_k$  is an ALPS graph.

**nl is G-NL-compatible.** The proof is similar to the previous one by using the second point of Lemma 108 and hence it is omitted.

$\{\{n\} \mid n \in N\} \subseteq sh$  Let  $n \in N$ . From Lemma 100 there is  $k \in \{1, 2\}$  such that  $h_k(n) \neq \perp$ . Then  $\{h_k(n)\} \in sh_k$  since  $G_k$  is a ALPS graph, hence  $\{n\} \in sh$ .

**if there is a nonempty loop in  $G$  involving  $n$ , then  $n \in nl$ .** Assume there is a loop in  $N$  such that  $n \xrightarrow{f_1} m_1 \xrightarrow{f_2} \dots \xrightarrow{f_r} n$ . By Lemma 100, there exists  $k \in \{1, 2\}$  such that  $h_k(n) \neq \perp$ . By proceeding backward from the last edge toward the first using Lemma 96, we have that  $h_k(n) \xrightarrow{f_1} h_k(m_1) \xrightarrow{f_2} \dots \xrightarrow{f_r} h_k(n)$  is a loop in  $G_k$  involving  $h_k(n)$ . Therefore,  $h_k(n) \in nl_k$  and  $n \in nl$ .

**$G$  is partially closed.** By the third point of Lemma 108 and since  $G_1$  and  $G_2$  are ALPS graphs, we have that  $G \star h_k^{-1}(sh_k) \star h_k^{-1}(nl_k)$  is partially closed, for  $k = 1, 2$ . Now, the following holds:

- Assume that  $\{n, m\} \in sh \wedge n' \xrightarrow{f} n \in E$ . Since  $sh = h_1^{-1}(sh_1) \cup h_2^{-1}(sh_2)$  there is  $k \in \{1, 2\}$  such that  $\{n, m\} \in h_k^{-1}(sh_k)$ . Then, since  $G_k \star h_k^{-1}(sh_k) \star h_k^{-1}(nl_k)$  is partially closed, we have that  $\{n', m\} \in sh_k \subseteq sh$ ;
- Assume that  $n \xrightarrow{f_1} m_1, n \xrightarrow{f_2} m_2 \in E, f_1 \neq f_2, \{m_1, m_2\} \in sh$ . Since  $sh = h_1^{-1}(sh_1) \cup h_2^{-1}(sh_2)$  there is  $k \in \{1, 2\}$  such that  $\{m_1, m_2\} \in h_k^{-1}(sh_k)$ . Then, since  $G \star h_k^{-1}(sh_k) \star h_k^{-1}(nl_k)$  is partially closed, we have that  $n \in h_k^{-1}(nl_k) \subseteq nl$ ;

- Assume that  $n \in nl \wedge n' \xrightarrow{f} n$ . Since by definition  $nl = h_1^{-1}(nl_1) \cup h_2^{-1}(nl_2)$  there is  $k \in \{1, 2\}$  such that  $n \in h_k^{-1}(nl_k)$ . Then, since  $G \star h_k^{-1}(sh_k) \star h_k^{-1}(nl_k)$  is partially closed, we have that  $n' \in h_k^{-1}(nl_k) \subseteq nl$  and then the thesis.  $\square$

**Lemma 110.** *Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be ALPS graphs. Then  $\mathbb{G}_1 \vee \mathbb{G}_2$  is the least upper bound of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ .*

*Proof.* Let  $\mathbb{G} = \mathbb{G}_1 \vee \mathbb{G}_2$ . By Lemma 109 we have that  $\mathbb{G}_1 \vee \mathbb{G}_2$  is an ALPS graph. The following holds:

**$\mathbb{G}$  is an upper bound of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ .** By Theorem 101 for  $k = 1, 2$ ,  $G_k \leq G_1 \vee G_2 = G$  and let  $h_k : G \rightarrow G_k$  be the corresponding morphism. We have that  $h_k^{-1}(sh_k) \subseteq h_1^{-1}(sh_1) \cup h_2^{-1}(sh_2) = sh$  and  $h_k^{-1}(nl_k) \subseteq h_1^{-1}(nl_1) \cup h_2^{-1}(nl_2) = nl$ . Then the thesis follows by Propositions 105 and 106.

**$\mathbb{G}$  is the least upper bound of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ .** Let  $\mathbb{G}'$  be an ALPS graph and assume that  $\mathbb{G}_1 \leq \mathbb{G}'$  and  $\mathbb{G}_2 \leq \mathbb{G}'$ . Obviously  $G \leq G'$  since  $G = G_1 \vee G_2$ . Let  $h'_1 : G' \rightarrow G_1$ ,  $h'_2 : G' \rightarrow G_2$  be morphisms of aliasing graphs, they factor through  $h : G' \rightarrow G$  and  $h_1 : G \rightarrow G_1$ ,  $h_2 : G \rightarrow G_2$ , that is,  $h'_1 = h_1 \circ h$  and  $h'_2 = h_2 \circ h$ . Since, by Propositions 105 and 106,  $h'_k^{-1}(sh_k) \subseteq sh'$ , we have  $h^{-1}(h_k^{-1}(sh_k)) \subseteq sh'$ , hence  $h^{-1}(sh) = h^{-1}(h_1^{-1}(sh_1) \cup h_2^{-1}(sh_2)) = h^{-1}(h_1^{-1}(sh_1)) \cup h^{-1}(h_2^{-1}(sh_2)) \subseteq sh'$ . Similarly for the nonlinearity component. Therefore, the proof follows by Propositions 105 and 106.  $\square$

**Lemma 111.** *Let  $\mathbb{G}_1$  and  $\mathbb{G}_2$  be ALPS graphs. Then  $\mathbb{G}_1 \wedge \mathbb{G}_2$  is the greatest lower bound of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ .*

*Proof.* Let  $\mathbb{G} = \mathbb{G}_1 \wedge \mathbb{G}_2$ . By definition 72  $\mathbb{G} = cl^\downarrow(G')$ , where

- $G' = G_1 \wedge G_2$  with morphisms  $h_1 : G_1 \rightarrow G$  and  $h_2 : G_2 \rightarrow G$ ;
- $sh' = \{\{n, m\} \in \mathcal{P}_2(N) \mid \forall k \in \{1, 2\} h_k^{-1}(\{\{n, m\}\}) \subseteq sh_k\}$ ;
- $nl' = \{n \in N \mid \forall k \in \{1, 2\} h_k^{-1}(n) \subseteq nl_k\}$ ;

The proof that  $G'$  is an aliasing graph and that for  $k = 1, 2$ ,  $G' \leq G_k$  follows by Theorem 102. Let  $h_1 : G_1 \rightarrow G'$ ,  $h_2 : G_2 \rightarrow G'$  be the corresponding morphisms. The following holds:

**$\mathbb{G}$  is a ALPS graph.** By Theorem 68, we have only prove that  $\mathbb{G}'$  is closed wrt red. The following holds:

**$sh'$  is  $G'$ -SH-compatible.** Assume  $\{n, m\} \in sh'$ . By the third point of Lemma 96, for each  $k \in \{1, 2\}$ , there are nodes  $n_k, m_k \in N_k$  s.t.  $h_k(n_k) = n$ ,  $h_k(m_k) = m$ ,  $\tau_{G_k}(n_k) = \tau_{G'}(n)$ , and  $\tau_{G_k}(m_k) = \tau_{G'}(m)$ . Moreover, by definition of  $sh'$ ,  $\{n_k, m_k\} \in sh_k$ . Since  $\mathbb{G}_k$  is an ALPS graph, we have  $(\tau_{G'}(n), \tau_{G'}(m)) = (\tau_{G_k}(n_k), \tau_{G_k}(m_k)) \in SH$  and then the thesis.

**$nl'$  is  $G'$ -NL-compatible.** Assume  $n \in nl'$ . By the third point of Lemma 96, for each  $k \in \{1, 2\}$ , there exists  $n_k \in N_k$  s.t.  $h(n_k) = n$ ,  $\tau_{G_k}(n_k) = \tau_{G'}(n)$  and by definition of  $nl'$ ,  $n_k \in nl_k$ . Since  $\mathbb{G}_k$  is an ALPS graph,  $\tau_{G'}(n) = \tau_{G_k}(n_k) \in NL$ .

**$\mathbb{G}$  is a lower bound of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ .** By definition  $\mathbb{G} \leq \mathbb{G}'$ . Therefore, it is sufficient to prove that  $\mathbb{G}'$  is a lower bound of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ . By Theorem 102 for  $k = 1, 2$ ,  $G' = G_1 \wedge G_2 \leq G_k$ . If  $\{n, m\} \in sh'$ , then  $h_k^{-1}(\{n, m\}) \subseteq sh_k$  by definition and therefore  $h_k^{-1}(sh') \subseteq sh_k$ . Analogously, if  $n \in nl'$ , then  $h_k^{-1}(n) \subseteq nl_k$  and then  $h_k^{-1}(nl') \subseteq nl_k$ . Hence, for  $k = 1, 2$ , by Propositions 105 and 106,  $\mathbb{G}' \leq \mathbb{G}_k$ .

$\mathbb{G}$  is the greatest lower bound of  $\mathbb{G}_1$  and  $\mathbb{G}_2$ . Let  $\mathbb{G}^*$  be an ALPS graph smaller than  $\mathbb{G}_1$  and  $\mathbb{G}_2$ , with corresponding morphisms  $h_1^*$  and  $h_2^*$ , and prove that  $\mathbb{G}^* \preceq \mathbb{G}'$ . By Proposition 106, we have to prove the following facts.

$G^* \preceq G'$ . Obviously  $G^* \preceq G_1 \wedge G_2 = G'$  and therefore there is  $h^* : G' \rightarrow G^*$  s.t.  $h_k^* = h^* \circ h_k$ .  $h^{*-1}(sh^*) \subseteq sh'$ . Let  $\{n, m\} \in h^{*-1}(sh^*)$ . Since for  $k = 1, 2$ ,  $\mathbb{G}^* \preceq \mathbb{G}_k$ , by Propositions 105 and 106,  $h_k^{*-1}(sh^*) \subseteq sh_k$ . Moreover  $h_k^{*-1} = h_k^{-1} \circ h^{*-1}$  and then

$$h_k^{-1}(\{n, m\}) \subseteq h_k^{-1}(h^{*-1}(sh^*)) = h_k^{*-1}(sh^*) \subseteq sh_k.$$

By Definition 72  $sh' = \{\{n', m'\} \in \mathcal{P}_2(N) \mid \forall k \in \{1, 2\}. h_k^{-1}(\{n', m'\}) \subseteq sh_k\}$ . Therefore,  $\{n, m\} \in sh'$  and then the thesis.

$h^{*-1}(nl^*) \subseteq nl'$ . The proof is similar to the one for the previous point.

Therefore, we have that  $\mathbb{G}^* \preceq \mathbb{G}'$ . Moreover since  $\mathbb{G}^*$  is closed and  $\mathbb{G}$  is the greatest closed pre-ALPS graph smaller than  $\mathbb{G}'$ , we have that  $\mathbb{G}^* \preceq \mathbb{G}$ . □

**Theorem 74.** *The preordered set of ALPS graphs has*

- a least element  $\perp_\tau \star \emptyset \star \emptyset$ ;
- a greatest element  $\top_\tau \star sh \star nl$ , where
  - $sh = \{\{n, m\} \in \mathcal{P}_2(I_\tau) \mid (\tau(n), \tau(m)) \in SH\}$  and
  - $nl = \{n \in I_\tau \mid \tau(n) \in NL\}$ ;
- a least upper bound  $\mathbb{G}_1 \vee \mathbb{G}_2$  for each pair  $\mathbb{G}_1$  and  $\mathbb{G}_2$  of ALPS graphs;
- a greatest lower bound  $\mathbb{G}_1 \wedge \mathbb{G}_2$  for each pair  $\mathbb{G}_1$  and  $\mathbb{G}_2$  of ALPS graphs.

*Proof.* The proof that  $\perp = \perp_\tau \star \emptyset \star \emptyset$  and  $\top = \top_\tau \star sh \star nl$  are least and greatest element is straightforward. Given a generic ALPS graph  $\mathbb{G}$ , it is easy to prove that  $\perp \preceq \mathbb{G}$  and  $\mathbb{G} \preceq \top$ . For the latter, if  $h : \top_\tau \rightarrow G$ , Lemma 108 ensures that  $h^{-1}(nl) \subseteq \top.nl$  and  $h^{-1}(sh) \subseteq \top.sh$ . Now, the proof follows by Lemmas 110 and 111. □

### A.3.4 The domain of ALPS graphs

The following proposition shows that the abstraction of a concrete state is an ALPS graph.

**Proposition 112.** *If  $\sigma \in \Sigma_\tau$ , then  $\alpha(\sigma)$  is an ALPS graph.*

*Proof.* We first prove that if  $\sigma \in \Sigma_\tau$  then  $\alpha(\sigma)$  is an ALPS graph. Recall that, by Definition 52,  $\alpha_a(\sigma) = G$  where

- $N = \{l \in Loc \mid \exists i \in I_\tau \phi(i) = l\}$ ;
- $n \xrightarrow{f} n' \in E$  iff  $n.f = n' \in N$ ;
- $\ell = \phi$

By Definition 62, we have to prove that

**$sh$  is G-SH-compatible.** Assume  $\{n, m\} \in sh$ . By definition of  $\alpha(\sigma)$ ,  $n, m \in Loc$ ,  $n, m$  share in  $\sigma$  and there exist  $i, j \in I_\tau$  such that  $\phi(i) = n$ ,  $\phi(j) = m$ ,  $\tau(i) = \tau_G(n)$ ,  $\tau(j) = \tau_G(m)$ . Then by Definition 22,  $i$  and  $j$  share in  $\sigma$  and by Proposition 32,  $C(\tau(i)) \cap C(\tau(j)) \neq \emptyset$ . Therefore, by previous results,  $C(\tau_G(n)) \cap C(\tau_G(m)) \neq \emptyset$  and hence  $(n, m) \in SH$ .

$nl$  is **G-NL-compatible**. The proof is analogous to the previous one by using Proposition 34 and hence it is omitted.

$\{\{n\} \mid n \in N\} \subseteq sh$ . Straightforward by observing that, by definition of  $G$ , if  $n \in N$  there exists  $i \in I_\tau$  s.t.  $\phi(i) = n$ . Therefore,  $n \in Loc$ ,  $n$  shares with itself in  $\sigma$  and, by Definition 75,  $\{n\} \in sh$ .

**If there is a nonempty loop in  $G$  involving  $n$ , then  $n \in nl$ .** Assume there is a loop in  $N$  such that  $n \xrightarrow{f_1} n_1 \xrightarrow{f_2} \dots \xrightarrow{f_n} n$ . Therefore,  $n = n.f_1 \dots f_n$  in  $\sigma$  and then by Definition 12,  $n$  is not linear in  $\sigma$ . By Definition 75,  $n \in nl$  and hence the thesis.

$\mathbb{G}$  is **partially closed**. Assume that  $\{n, m\} \in sh$  and  $n' \xrightarrow{f} n$ . By definition of  $\alpha(\sigma)$ ,  $n$  and  $m$  share in  $\sigma$  and then, by Definition 12, there exists  $l \in \text{dom}(\mu)$  such that  $n \xrightarrow{*}_\sigma l$  and  $m \xrightarrow{*}_\sigma l$ . Moreover, by definition of  $G$ ,  $n'.f = n$ . Therefore, by Definition 12,  $n' \xrightarrow{*}_\sigma l$ ,  $n'$  shares with  $m$  in  $\sigma$  and then, by definition of  $\alpha(\sigma)$ ,  $\{n', m\} \in sh$ .

Now, assume that  $n \xrightarrow{f_1} m_1, n \xrightarrow{f_2} m_2, f_1 \neq f_2$  and  $\{m_1, m_2\} \in sh$ . By definition of  $G$ ,  $n.f_1 = m_1 \neq \text{null}, n.f_2 = m_2 \neq \text{null}$  and  $m_1, m_2$  share in  $\sigma$ . Then since  $f_1 \neq f_2$ , by Definition 12,  $n$  is nonlinear in  $\sigma$  and therefore, by definition of  $\alpha(\sigma)$ ,  $n \in nl$ .

Finally, assume that  $n \in nl$  and  $n' \xrightarrow{f} n$ . By definition of  $\alpha(\sigma)$  and by Definition 12, there are two sequence of fields  $\bar{f}_1 \neq \bar{f}_2$  such that  $n.\bar{f}_1 = n.\bar{f}_2 \neq \text{null}$ . Then, since  $n'.f = n \neq \text{null}$  by definition of  $G$ , we have that  $n'.f \cdot \bar{f}_1 = n'.f \cdot \bar{f}_2 \neq \text{null}$ , with  $f \cdot \bar{f}_1 \neq f \cdot \bar{f}_2$ . Therefore,  $n'$  is nonlinear in  $\sigma$  and then, by definition of  $\alpha(\sigma)$ ,  $n' \in nl$ . □

**Proposition 76.** (Concretization of ALPS graphs). *The concretization map induced by the abstraction map  $\alpha$  satisfies the following property:*

$$\begin{aligned} \gamma(\mathbb{G}) = \{ & \sigma \in \Sigma_\tau \mid \sigma \in \gamma_a(G), \\ & \forall i \in I_\tau. i \text{ nonlinear in } \sigma \Rightarrow \ell(i) \in nl, \\ & \forall i, i' \in I_\tau. i \text{ share with } i' \text{ in } \sigma \Rightarrow \{\ell(i), \ell(i')\} \in sh \} . \end{aligned}$$

*Proof.* It is a straightforward application of the definition of  $\alpha$ . □

**Proposition 113.** *Let  $\sigma = \phi \star \mu \in \Sigma_\tau, i_1, i_2 \in I_\tau$  and let  $\mathbb{G} \in \text{ALPS}$ , such that  $\sigma \in \gamma(\mathbb{G})$ .*

- if  $\phi(i_1) \neq \text{null}$ , then  $\tau(\phi(i_1)) \leq \tau_G(\ell(i_1))$ ,
- if  $i_1$  and  $i_2$  share in  $\sigma$ , then  $\{\ell(i_1), \ell(i_2)\} \in \mathcal{P}_2(N)$  is  $G$ -SH-compatible and  $C(\tau(i_1)) \cap C(\tau_G(\ell(i_2))) \neq \emptyset$ .
- if  $i_1$  is not linear in  $\sigma$ , then  $\ell(i_1) \in N$  is  $G$ -NL-compatible.

*Proof.* Given  $\mathbb{G}' = \alpha(\sigma)$ , we have  $\mathbb{G}' \leq \mathbb{G}$ . Moreover:

- By Proposition 21, for each  $j \in I_\tau$  such that  $\phi(j) = \phi(i_1)$ , we have that  $\tau(\phi(i_1)) = \tau(\phi(j)) \leq \tau(j)$  and therefore, by Definition 39,  $\tau(\phi(i_1)) \leq \tau_{G'}(\ell'(i_1))$ . Moreover, by Lemma 96,  $\tau_{G'}(\ell'(i_1)) \leq \tau_G(\ell(i_1))$ .
- Now, assume that  $i_1$  and  $i_2$  share in  $\sigma$ . By Lemma 93,  $C(\tau(\phi(i_1))) \cap C(\tau(\phi(i_2))) \neq \emptyset$ . By Proposition 21, we get  $C(\tau(i_1)) \cap C(\tau(\phi(i_2))) \neq \emptyset$ , and by the first point of this proposition, we get both  $C(\tau(i_1)) \cap C(\tau_G(\ell(i_2))) \neq \emptyset$  and  $C(\tau_G(\ell(i_1))) \cap C(\tau_G(\ell(i_2)))$ , the latter meaning that  $\{\ell(i_1), \ell(i_2)\}$  is  $G$ -SH-compatible.
- Finally, assume that  $i_1$  is not linear in  $\sigma$ . By Lemma 94,  $\tau(\phi(i_1)) \in NL$ . Then, by the first point of this proposition, we have  $\tau_G(\ell(i)) \in NL$ , that is,  $\ell(i)$  is  $G$ -NL-compatible. □



**A.4 An abstract semantics on ALPS**

*A.4.1 Auxiliary operators*

**Proposition 80.** For each  $\mathbb{G} \in \text{ALPS}_\tau$  and  $V \subseteq \text{dom}(\tau)$ ,  $\gamma(\mathbb{G})_{\parallel V} \subseteq \gamma(\mathbb{G}_{\parallel V})$ .

*Proof.* Let  $W = V \cup \{v.f \in Q_\tau \mid v \in V\}$  and let  $\phi \star \mu \in \gamma(\mathbb{G})_{\parallel V}$ . By definition, there exists  $\phi' \star \mu \in \gamma(\mathbb{G})$  such that  $\phi = \phi'|_V$  and  $\alpha(\phi' \star \mu) \leq \mathbb{G}$ . Let  $\alpha(\phi' \star \mu) = G_1 \star sh_1 \star nl_1$ ,  $\alpha(\phi \star \mu) = G_2 \star sh_2 \star nl_2$ ,  $\mathbb{G} = G \star sh \star nl$  and  $\mathbb{G}_{\parallel V} = G' \star sh' \star nl'$ . Since  $\alpha(\phi' \star \mu) \leq \mathbb{G}$  by Proposition 57 and Theorem 42, there exists a morphism  $h : G \rightarrow G_1$ , such that  $\forall i \in I_\tau. \ell_1(i) \in nl_1 \Rightarrow \ell(i) \in nl$  and  $\forall i, j \in I_\tau. \{\ell_1(i), \ell_1(j)\} \in sh_1 \Rightarrow \{\ell(i), \ell(j)\} \in sh$ . The following holds

- $(h' = h|_{N'} \text{ is a morphism from } G' \text{ to } G_2)$  To prove the statement it is sufficient to observe that, by Definition 52, for each  $w \in W$ ,  $h'(\ell'(w)) = h|_{N'}(\ell|_V(w)) = h(\ell(w)) = \ell_1(w) = \phi(w) = \phi'(w) = \ell_2(w)$  and then the thesis.
- $(\forall i \in W. \ell_2(i) \in nl_2 \Rightarrow \ell'(i) \in nl')$  Let  $i \in W$  such that  $\ell_2(i) \in nl_2$ . By Definitions 52 and 75,  $\ell_2(i) = \phi(i) = \phi'(i)$  and  $\phi(i)$  is not linear in  $\phi \star \mu$ . Since  $\phi = \phi'|_V$ , we have that  $\phi'(i)$  is not linear in  $\phi' \star \mu$  and therefore  $\phi(i) = \phi'(i) = \ell_1(i) \in nl_1$ . Then  $\ell(i) \in nl$  and by definition of  $\mathbb{G}_{\parallel V}$ , we have that  $\ell'(i) = \ell(i) \in nl'$  and then the thesis.
- $(\forall i \in W. \{\ell_2(i), \ell_2(j)\} \in sh_2 \Rightarrow \{\ell'(i), \ell'(j)\} \in sh')$  The proof is similar to the previous one and hence it is omitted.

By previous results  $\alpha(\phi \star \mu) \leq \mathbb{G}_{\parallel V}$ . Therefore,  $\phi \star \mu \in \gamma(\mathbb{G}_{\parallel V})$  and then the thesis. □

**Lemma 114.** Given  $i \in I_\tau$ , we have that  $\top_{|i=\text{null}}$  is the largest ALPS-graph such that  $\ell(i) = \text{null}$ .

*Proof.* Let us denote by  $N_\top$  the set of nodes in  $\top_{|i=\text{null}}$  and by  $\ell_\top$  its labeling function. Note that, if  $i \in Q_\tau$ , then  $N_\top = I_\tau \setminus \{i\}$ , otherwise  $i$  is a variable  $v$  and  $N_\top = I_\tau \setminus \{v\} \setminus \{v.f \mid v.f \in Q_\tau\}$ . Moreover, note that  $\ell_\top(i) = i$  if  $i \in N_\top$ ,  $\perp$  otherwise.

Given an ALPS graph  $\mathbb{G}$  such that  $\ell(i) = \text{null}$ , consider the map  $h : N_\top \rightarrow N$  such that  $h(j) = \ell(j)$  for each  $j \in N_\top$ . This is a morphism between aliasing graph, since for each identifier  $j$ :

- If  $j \notin N_\top$ , then  $h(\ell_\top(j)) = h(\perp) = \perp$ . Moreover,  $j$  is either  $i$  or an identifier  $i.f$ . In both cases,  $\ell(j) = \perp$ .
- If  $j \in N_\top$ , then  $h(\ell_\top(j)) = h(j) = \ell(j)$ .

As a consequence, we also have that  $\ell(j) \neq \perp$  implies  $\ell_\top(j) \neq \perp$ . By definition of the sharing and nonlinearity components of  $\top$ , it is immediate to check that  $\mathbb{G} \leq \top_{|i=\text{null}}$ . □

**Proposition 82.** For each  $\mathbb{G} \in \text{ALPS}_\tau$  and  $i \in I_\tau$ ,  $\mathbb{G}_{|i=\text{null}} = \mathbb{G} \wedge \top_{|i=\text{null}}$ .

*Proof.* We prove the two implications of the equality separately.

$\mathbb{G}_{|i=\text{null}} \leq \mathbb{G} \wedge \top_{|i=\text{null}}$ . By Proposition 65 we know that  $\mathbb{G}_{|i=\text{null}} \leq \mathbb{G}$ , while from Lemma 114 we have that  $\mathbb{G}_{|i=\text{null}} \leq \top_{|i=\text{null}}$ . The result follows since  $\wedge$  is the greatest lower bound.

$\mathbb{G} \wedge \top_{|i=\text{null}} \leq \mathbb{G}_{|i=\text{null}}$ . Let  $\mathbb{G}_1 = \mathbb{G} \wedge \top_{|i=\text{null}}$  and  $\mathbb{G}_2 = \mathbb{G}_{|i=\text{null}}$ . The following facts hold.

$\ell_1^{-1}(N_1) \subseteq \ell_2^{-1}(N_2)$ . Let  $j \in I_\tau$  such that  $\ell_2(j) = \perp$ , we show that  $\ell_1(j) = \perp$  too. By definition of  $\mathbb{G}$ , if  $\ell_2(j) = \perp$  then either  $\ell(j) = \perp$  or  $\ell(j) \in \overrightarrow{\{\ell(i)\}}$ . In the latter case, if  $\ell'$  is the labeling map of  $\top_{|i=\text{null}}$ , we have that  $\ell'(j) = \perp$ . Hence, either  $\ell(j) = \perp$  or  $\ell'(j) = \perp$ . Due to the definition of  $\wedge$ , in both cases we have  $\ell_1(j) = \perp$

$G_1 \preceq G_2$ . Since  $G_1 \preceq G$  there exists  $h_1 : N \rightarrow N_1$  such that for each  $j \in I_\tau$ ,  $h_1(\ell(j)) = \ell_1(j)$ . Consider  $h : N_2 \rightarrow N_1$  such that

$$h(\ell_2(j)) = \begin{cases} h_1(\ell(j)) & \text{if } \ell_2(j) \neq \perp \\ \perp & \text{otherwise} \end{cases}$$

This is well defined since if  $\ell_2(j) = \ell_2(j') \neq \perp$ , then by definition of  $G_{|i=\text{null}}$ ,  $\ell(j) = \ell(j') \neq \perp$ . Moreover, it is a graph morphism. Given  $j \in I_\tau$ , if  $\ell_2(j) \neq \perp$ , then  $h(\ell_2(j)) = \ell_1(j)$  by the definition above. Moreover, by the previous point, if  $\ell_2(j) = \perp$ , then  $\ell_1(j) = \perp$ . Hence  $h(\ell_2(j)) = \ell_1(j)$  again.

$\ell_1^{-1}(sh_1) \subseteq \ell_1^{-2}(sh_2)$ . First of all, observe that if  $\{j, j'\} \in \ell_1^{-1}(sh_1)$ , then by previous result and by definition of  $\preceq$  we have that  $j, j' \in N_2$  and  $\{j, j'\} \in \ell^{-1}(sh)$ . Now the thesis is immediate by definition of  $G_2$ .

$\ell_2^{-1}(nl_2) \subseteq \ell_1^{-1}(nl_1)$ . The proof is analogous to the previous one. □

**Proposition 83.** For each  $G \in \text{ALPS}_\tau$  and  $i \in I_\tau$ ,  $\gamma(G)_{|i=\text{null}} \subseteq \gamma(G_{|i=\text{null}})$ .

*Proof.* If  $\sigma \in \gamma(G)_{|i=\text{null}}$ , then  $\sigma \in \gamma(G)$  and  $\phi(i) = \text{null}$ , that is,  $\sigma \in \gamma(\top_{|i=\text{null}})$  by Lemma 114. Therefore,  $\sigma \in \gamma(G) \cap \gamma(\top_{|i=\text{null}}) \subseteq \gamma(G \wedge \top_{|i=\text{null}})$  which is equal to  $\gamma(G_{|i=\text{null}})$  by Proposition 82. □

**Proposition 115.** Let  $v \in \text{dom}(\tau)$  and let  $G, G' \in \text{ALPS}$ , such that  $G \preceq G'$ . Then  $G_{|v=\text{null}} \preceq G'_{|v=\text{null}}$ .

*Proof.* The proof is immediate, by observing that by hypothesis and by Proposition 82,  $G_{|v=\text{null}} = G \wedge \top_{|v=\text{null}} \preceq G' \wedge \top_{|v=\text{null}} = G'_{|v=\text{null}}$ . □

**Lemma 116.** Given  $v, w \in \text{dom}(\tau)$ , we have that  $\top_{|v=w}$  is the largest ALPS-graph such that  $\ell(v) = \ell(w)$ .

*Proof.* Let  $G$  be an ALPS graphs such that  $\ell(v) = \ell(w)$ . If  $\tau(v)$  and  $\tau(w)$  do not form a chain, then  $\ell(v) = \ell(w)$  implies that  $\ell(v) = \text{null} = \ell(w)$ , and we have  $G \preceq \top_{|v=w} = \top_{|v=\text{null}, w=\text{null}}$  by Lemma 114.

If  $\tau(v)$  and  $\tau(w)$  do form a chain, let us denote by  $N_\top$  the set of nodes in  $\top_{|v=w}$  and by  $\ell_\top$  its labeling function. Assume, without loss of generality, that  $\tau(v) \leq \tau(w)$ . Then  $N_\top = I_\tau \setminus \{w\} \setminus \{w.f \mid f \in \text{dom}(\tau(w))\}$  and  $\ell_\top(i)$  is equal to  $i$ , with the proviso that any possible occurrence of  $w$  is replaced by  $v$ . Consider the map  $h : N_\top \rightarrow N$  such that  $h(j) = \ell(j)$  for each  $j \in N_\top$ . This is a morphism between aliasing graphs since for each identifier  $j$ :

- if  $j = w$ , then  $h(\ell_\top(w)) = h(v) = \ell(v) = \ell(w)$ ;
- if  $j = w.f$ , then  $h(\ell_\top(w.f)) = h(v.f) = \ell(v.f) = \ell(v).f = \ell(w).f = \ell(w.f)$ ;
- if  $j \in N_\top$ , then  $h(\ell_\top(j)) = h(j) = \ell(j)$ .

By definition of the sharing and nonlinearity components of  $\top$ , it is immediate to check that  $G \preceq \top_{|v=w}$ . □

**Proposition 84.** For each  $G \in \text{ALPS}_\tau$  and  $v, w \in \text{dom}(\tau)$ ,  $\gamma(G)_{|v=w} \subseteq \gamma(G_{|v=w})$ .

*Proof.* Similar to the proof of Proposition 83. □

A.4.2 Abstract semantics

**Proposition 117** (Correctness of *add*). *Let  $\phi \star \mu \in \Sigma_\tau$ ,  $i \in I_\tau$ ,  $\tau' = \tau[\text{res} \mapsto \kappa_{\text{res}}]$  and let  $\mathbb{G} \in \text{ALPS}$  such that  $\phi(i) \neq \text{null}$ ,  $\tau(\phi(i)) \leq \kappa_{\text{res}}$  and  $\phi \star \mu \in \gamma(\mathbb{G})$ . Then  $\phi' \star \mu = \phi[\text{res} \mapsto \phi(i)] \star \mu \in \Sigma_{\tau'}$  and  $\alpha(\phi' \star \mu) \leq \text{add}(\mathbb{G}, \ell(i), \kappa_{\text{res}})$ .*

*Proof.* First of all, observe that since by hypothesis  $\tau(\phi(i)) \leq \kappa_{\text{res}}$ ,  $\phi' \star \mu \in \Sigma_{\tau'}$ . Now, let  $\mathbb{G}' = \alpha(\phi \star \mu)$ ,  $\mathbb{G}_1 = \alpha(\phi' \star \mu)$  and  $\mathbb{G}_2 = \text{add}(\mathbb{G}, \ell(i), \kappa_{\text{res}})$ . Since in a Galois connection  $\alpha \circ \gamma$  is reductive (Section 2), we have that  $\mathbb{G}' \leq \mathbb{G}$  and therefore there exists  $h' : G \rightarrow G'$ .

Let  $\kappa_1 = \psi_{G'}(\ell'(i))$ ,  $\kappa_2 = \psi_G(\ell(i))$ ,  $F_1 = \text{dom}(\kappa_{\text{res}}) \setminus \text{dom}(\kappa_1)$  and let  $F_2 = \text{dom}(\kappa_{\text{res}}) \setminus \text{dom}(\kappa_2)$ . Since  $\mathbb{G}' \leq \mathbb{G}$ , by Lemma 96,  $\kappa_1 \leq \kappa_2$ ,  $\text{dom}(\kappa_2) \subseteq \text{dom}(\kappa_1)$  and therefore  $F_1 \subseteq F_2$ .

Moreover, let  $W = \ell'^{-1}(N') \cup \{\text{res}\} \cup \{\text{res.f} \in Q_\tau \mid \text{f} \notin F_2\}$ . Since  $F_1 \subseteq F_2$ ,  $\mathbb{G}' \leq \mathbb{G}$  and by definition of *add*, for each  $w \in W$  there exists  $w^* \in \ell'^{-1}(N')$  such that the following holds:

- $\ell_1(w) = \ell_1(w^*) = \ell'(w^*) \neq \perp$ ,  $\ell_2(w) = \ell_2(w^*) = \ell(w^*) \neq \perp$  and
- $w \in \ell'^{-1}(N')$  and  $w^* = w$  or
  - $w = \text{res}$  and  $w^* = i$  or
  - $w = \text{res.f}$  and  $w^* = i.f$ .

The following holds.

(1)  $G_1 \leq G_2$ ). By definition

$$G_1 = (N' \cup \{\phi(i).f \mid f \in F_1, \phi(i).f \neq \text{null}\}) \star (E' \cup \{\phi(i) \xrightarrow{f} \phi(i).f \mid f \in F_1, \phi(i).f \neq \text{null}\}) \star \ell'[\text{res} \mapsto \ell'(i)]$$

and  $G_2 = N \cup \{n_f \mid f \in F_2\} \star E \cup \{\ell(i) \xrightarrow{f} n_f \mid f \in F_2\} \star \ell[\text{res} \mapsto \ell(i)]$ , where  $\{n_f \mid f \in F_2\} \cap N = \emptyset$ , namely, they are fresh nodes. Now, we can define  $h$  such that

$$h(n) = \begin{cases} h'(n) & \text{if } n \in N \\ \phi(i).f & \text{if } n = n_f \in N_2 \setminus N \text{ and } \phi(i).f \neq \text{null} \\ \perp & \text{otherwise.} \end{cases}$$

It is easy to check that  $h : G_2 \rightarrow G_1$  is a morphism and then the thesis.

(2)  $\ell_1^{-1}(sh_1) \subseteq \ell_2^{-1}(sh_2)$ ) First of all, observe that, by definition of  $\mathbb{G}_1$ ,

$$\text{for each } n \in N', \tau_{G_1}(n) \leq \tau_{G'}(n). \tag{A3}$$

Now, let  $\{j, j'\} \in \ell_1^{-1}(sh_1)$ . We have the following possibilities.

$j, j' \in W$ ) By construction  $\{j^*, j'^*\} \in \ell'^{-1}(sh')$  and since  $\mathbb{G}' \leq \mathbb{G}$ , we have that  $\{j^*, j'^*\} \in \ell^{-1}(sh)$ . Therefore  $\{\ell(j^*), \ell(j'^*)\} \in sh$  and then  $\{\ell(j^*), \ell(j'^*)\}$  is  $G$ -SH-compatible. Moreover, by hypothesis  $\ell_2(j) = \ell_2(j^*)$  and  $\ell_2(j') = \ell_2(j'^*)$ . Now, observe that by Lemma 93 and the first point of Proposition 113, since  $\mathbb{G}' = \alpha(\phi \star \mu) \leq \mathbb{G}$  and  $j^*$  and  $j'^*$  share in  $\phi \star \mu$ , we have that  $C(\tau(\phi(j^*))) \cap C(\tau(\phi(j'^*))) \neq \emptyset$ ,  $\tau(\phi(j^*)) \leq \tau_G(\ell(j^*))$  and  $\tau(\phi(j'^*)) \leq \tau_G(\ell(j'^*))$ . Now, since by hypothesis  $\tau(\phi(i)) \leq \kappa_{\text{res}}$  we have that  $\{\ell_2(j), \ell_2(j')\} = \{\ell(j^*), \ell(j'^*)\}$  is  $G_2$ -SH-compatible and then the thesis follows by definition of *add*.

$j \in W$  and  $j' \notin W$ . In this case  $j' = \text{res.f}$ , with  $\mathbf{f} \in F_2$ . By construction  $\{j^*, i\} \in \ell'^{-1}(sh')$  and since  $\mathbb{G}' \preceq \mathbb{G}$ , we have that  $\{j^*, i\} \in \ell^{-1}(sh)$ , namely

$$\{\ell(j^*), \ell(i)\} \in sh. \tag{A4}$$

Moreover, by hypothesis  $\ell_2(j) = \ell_2(j^*) = \ell(j^*)$  and  $\ell_2(j') = n_{\mathbf{f}}$ , where  $n = \ell(i)$ . Moreover, since  $\text{res.f}$  and  $j$  share in  $\mathbb{G}_1$ , by the second point of Proposition 113, we have that  $C(\tau(\text{res.f})) \cap C(\tau_{G_1}(\ell_1(j))) \neq \emptyset$  and therefore by (A3),  $C(\kappa_{\text{res.f}}) \cap C(\tau_{G'}(\ell'(j^*))) \neq \emptyset$ . By properties of morphisms and since  $\mathbb{G}' \preceq \mathbb{G}$ ,  $\tau_{G'}(\ell'(j^*)) \leq \tau_G(\ell(j^*))$  and therefore

$$C(\kappa_{\text{res.f}}) \cap C(\tau_G(\ell(j^*))) \neq \emptyset. \tag{A5}$$

Moreover, observe that by Lemma 93, since  $\mathbb{G}' = \alpha(\phi \star \mu) \preceq \mathbb{G}$  and  $j^*$  and  $\text{res.f}$  share in  $\phi' \star \mu$ , we have that  $C(\tau(\phi(j^*))) \cap C(\tau(\phi(i).\mathbf{f})) \neq \emptyset$ . Moreover by the first point of Proposition 113,  $\tau(\phi(j^*)) \leq \tau_G(\ell(j^*))$ . Then, since by hypothesis  $\tau(\phi(i)) \leq \kappa_{\text{res}}$  and then  $\tau(\phi(i).\mathbf{f}) \leq \kappa_{\text{res.f}}$ , we have that  $\{\ell_2(j), \ell_2(j')\} = \{\ell(j^*), n_{\mathbf{f}}\}$  is  $G_2$ -SH-compatible. Now, we have two possibilities

-  $\ell(i) \in nl$ . In this case by (A4), (A5) and definition of  $add$ ,  $\{\ell(j^*), n_{\mathbf{f}}\} = \{\ell_2(j), \ell_2(\text{res.f})\} \in sh_2$  and then the thesis.

-  $\ell(i) \notin nl$ . Since  $\mathbb{G}' \preceq \mathbb{G}$ , we have that  $\ell'(i) \notin nl'$  and therefore, by definition of  $\mathbb{G}_1$ ,  $\ell_1(i) \notin nl_1$ . If  $\exists \mathbf{f}'_1, \dots, \mathbf{f}'_k. \ell(i) \xrightarrow{\mathbf{f}'_1} n_1 \xrightarrow{\mathbf{f}'_2} n_2 \dots n_{k-1} \xrightarrow{\mathbf{f}'_k} \ell(j^*) \in E$  then, by (A4) and (A5),  $\{n_{\mathbf{f}}, \ell_2(j)\} = \{\ell_2(\text{res.f}), \ell_2(j^*)\} \in sh_2$  and then the thesis. Now, assume that  $\exists \mathbf{f}'_1, \dots, \mathbf{f}'_k. \ell(i) \xrightarrow{\mathbf{f}'_1} n_1 \xrightarrow{\mathbf{f}'_2} n_2 \dots n_{k-1} \xrightarrow{\mathbf{f}'_k} n_k = \ell(j^*) \in E$ . By definition for each  $l = 1, \dots, k - 1$ , there exists  $i_l$  such that  $\ell(i_l^*) = n_l$ . Moreover since  $\ell(i) \xrightarrow{\mathbf{f}'_1} n_1 \xrightarrow{\mathbf{f}'_2} n_2 \dots n_{k-1} \xrightarrow{\mathbf{f}'_k} n_k = \ell(j^*) \in E$ , for each  $l = 1, \dots, k$ ,  $i_l^* \neq i.\mathbf{f}'$  with  $\mathbf{f}' \in F_2$ . Then, since  $F_1 \subseteq F_2$ ,  $\ell'(j^*) \neq \text{null}$ , and by proceeding backwards from the last edge toward the first using Lemma 96,  $\ell'(i) \xrightarrow{\mathbf{f}'_1} \ell'(i_1^*) \xrightarrow{\mathbf{f}'_2} \ell'(i_2^*) \dots \ell'(i_{k-1}^*) \xrightarrow{\mathbf{f}'_k} n_k = \ell'(j^*) \in E'$ .

Moreover for each  $l = 1, \dots, k$ ,  $\ell_1(i_l^*) = \ell'(i_l^*) \in N'$  and therefore  $\ell_1(i) \xrightarrow{\mathbf{f}'_1} \ell_1(i_1^*) \xrightarrow{\mathbf{f}'_2} \ell_1(i_2^*) \dots \ell_1(i_{k-1}^*) \xrightarrow{\mathbf{f}'_k} n_k = \ell_1(j^*) \in E_1$  and therefore  $\ell_1(i) \xrightarrow{\mathbf{f}'_1} \ell_1(i_1^*) \xrightarrow{\mathbf{f}'_2} \ell_1(i_2^*) \dots \ell_1(i_{k-1}^*) \xrightarrow{\mathbf{f}'_k} \ell_1(j^*) = \ell_1(j) \in E_1$ . Moreover, since by construction

$\ell_1(i) = \ell_1(\text{res})$  and by hypothesis  $j' = \text{res.f} \in N_1$ , we have that  $\ell_1(i) \xrightarrow{\mathbf{f}} \ell_1(j') \in E_1$ .

Therefore, since by hypothesis  $\{\ell_1(j), \ell_1(j')\} \in sh_1$ , by definition of  $\text{cl}^\uparrow$  and by proceeding backward from the last edge toward the first, we have that  $\{\ell_1(i).\mathbf{f}'_1, \ell_1(i).\mathbf{f}_2\} \in sh_1$ , with  $\mathbf{f}'_1 \notin F_2$  and  $\mathbf{f}_2 \in F_2$ . Then, we have a contradiction, since by previous observations,  $\ell_1(i) \notin nl_1$ .

$j, j' \notin W$ . In this case  $j = \text{res.f}$  and  $j' = \text{res.f}'$ , with  $\mathbf{f}, \mathbf{f}' \in F_2$ . If  $\mathbf{f} = \mathbf{f}'$ , the proof is straightforward, since  $n_{\mathbf{f}} \in N_2$  and by construction  $\mathbb{G}_2 = \text{add}(\mathbb{G}, \ell(i), \kappa_{\text{res}})$  is closed. Now, assume that  $\mathbf{f} \neq \mathbf{f}'$ . In this case, since by hypothesis  $\{j, j'\} = \{\text{res.f}, \text{res.f}'\} \in \ell_1^{-1}(sh_1)$  we have that  $C(\kappa_{\text{res.f}}) \cap C(\kappa_{\text{res.f}'}) \neq \emptyset$ . Moreover, since  $\mathbf{f}, \mathbf{f}' \in F_2$ , we have that  $\ell_2(j) = n_{\mathbf{f}}$  and  $\ell_2(j') = n_{\mathbf{f}'}$ , where  $n = \ell(i)$ . Then, by Definitions 12 and 22 and since  $\phi'(\text{res}) = \phi'(i) = \phi(i)$ ,  $i$  is not linear in  $\phi \star \mu$ . By definition of  $\alpha$ ,  $\ell'(i) \in nl'$  and since  $\mathbb{G}' \preceq \mathbb{G}$ ,  $\ell(i) \in nl$ . Now, the thesis follows by definition of  $add$ .

(3)  $(\ell_1^{-1}(nl_1) \subseteq \ell_2^{-1}(nl_2))$ . Let  $j \in \ell_1^{-1}(nl_1)$ . We have two possibilities

$j \in W$ . In this case  $j^* \in \ell'^{-1}(N')$ . Since  $\mathbb{G}' \preceq \mathbb{G}$ , we have that  $j^* \in \ell^{-1}(nl)$ . Now, observe that by Lemma 94 and by the first point of Proposition 113, since  $\mathbb{G}' = \alpha(\phi \star \mu) \preceq \mathbb{G}$

and  $j^*$  is not linear in  $\phi \star \mu$ , we have that  $\tau(\phi(j^*)) \in NL$  and  $\downarrow \tau_G(\ell(j^*)) \cap NL \neq \emptyset$ . Now, since by hypothesis  $\tau(\phi(i)) \leq \kappa_{res}$ , and by previous observations we have that  $\tau_{G_2}(\ell(j))$  is  $G_2$ -NL-compatible and then the thesis.

$j \notin W$ . In this case, there exists  $f \in F_2$  such that  $j = res.f$ . By definition of  $\alpha$ ,  $i \in \ell'^{-1}(nl')$  and therefore since  $\mathbb{G}' \preceq \mathbb{G}$ ,  $i \in \ell^{-1}(nl)$ . Moreover, since  $\tau(\phi(i)) \leq \kappa_{res}$  we have that  $\tau(\phi(i).f) \leq \kappa_{res}.f$  and then  $\tau_{G_1}(\ell_1(j)) \leq \kappa_{res}.f$  and since  $\mathbb{G}_1$  is closed,  $\downarrow(\tau_{G_1}(\ell_1(j))) \cap NL \neq \emptyset$ . Then  $\downarrow(\kappa_{res}.f) \cap NL \neq \emptyset$  and since  $f \in F_2$ ,  $\ell_2(j) = n_f$ . Therefore, by definition of  $add$ ,  $j = res.f \in \ell_2^{-1}(nl_2)$  and then the thesis.

Now, the thesis follows by Proposition 106. □

**Theorem 87.** *The abstract semantics formalized in Figures 20–24 is correct wrt the concrete semantics in Section 2.2.2.*

*Proof.* According to the abstract interpretation framework Cousot and Cousot (1977), for each operation  $op : \Sigma_\tau \rightarrow \Sigma_{\tau'}$ ,  $\mathbb{G} \in ALPS$ , interpretation  $I$  and sharing interpretation  $I'$  such that  $I'(\kappa.m)$  is correct w.r.t.  $I(\kappa.m)$  for every method  $\kappa.m$ , we must prove that

$$\underbrace{\alpha(\mathcal{E}_\tau^I[[op]](\gamma(\mathbb{G})))}_{C(op)} \preceq \underbrace{\mathcal{S}\mathcal{E}_\tau^{I'}[[op]](\mathbb{G})}_{A(op)}.$$

where  $\mathcal{E}[\_]$  ( $\mathcal{C}[\_]$  for the commands) is given in Section 2.2.2 and  $\mathcal{S}\mathcal{E}[\_]$  ( $\mathcal{S}\mathcal{C}[\_]$ ) in Figures 20 and 23 (Figures 21 and 22). Remember that for the expressions we have  $\tau' = \tau + type_\tau(op)$  and  $res \notin dom(\tau)$ , while for the commands we have  $\tau' = \tau$  and  $res \notin dom(\tau)$ . The proof is direct for the expressions, while it is by induction for the commands (because commands are defined inductively in Section 2.2). First of all, observe that since in a Galois connection  $\alpha\gamma$  is reductive (Section 2), for each  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that

$$\mathbb{G}' = \alpha(\phi \star \mu) \preceq \mathbb{G} \tag{A6}$$

and therefore by definition of  $\alpha$ ,

$$\mathbb{G}'' = \alpha(\{\phi \star \mu \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \preceq \mathbb{G}. \tag{A7}$$

null  $\kappa$

We have  $\tau' = \tau + type_\tau(\text{null } \kappa) = \tau[res \mapsto \kappa]$ . Then

$$\begin{aligned} C(\text{null } \kappa) &= \alpha(\{\phi[res \mapsto \text{null}] \star \mu \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \\ (\text{Definition 75}) &= \alpha(\{\phi \star \mu \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \\ (A7) &\preceq \mathbb{G} \\ &= A(\text{null } \kappa). \end{aligned}$$

new  $\kappa$

Also in this case we have  $\tau' = \tau + type_\tau(\text{new } \kappa) = \tau[res \mapsto \kappa]$ . The proof is similar to that above, since the new object is allocated in a fresh location  $l$ , and hence is only reachable from  $res$ :

$$\begin{aligned} C(\text{new } \kappa) &= \alpha(\{\phi[res \mapsto l] \star \mu[l \mapsto \text{new}(\kappa)] \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \\ (\text{by definition of } \alpha) &= \bigvee \{\alpha(\phi[res \mapsto l] \star \mu[l \mapsto \text{new}(\kappa)]) \mid \phi \star \mu \in \gamma(\mathbb{G})\} \end{aligned}$$

$$\begin{aligned}
 (\text{Definition 75}) &= \bigvee \{ (N' \cup \{l\} \star E' \star \ell' [\text{res} \mapsto l]) \star sh' \cup \{\{l\} \star nl' \mid \\
 &\quad \mathbb{G}' \in \alpha\gamma(\mathbb{G}) \text{ and } l \notin N'\} \\
 (\text{A6}) &\leq (N \cup \{n_{new}\} \star E \star \ell [\text{res} \mapsto n_{new}]) \star sh \cup \{\{n_{new}\} \star nl \\
 &\quad \text{with } n_{new} \notin N \\
 &= A(\text{new } \kappa).
 \end{aligned}$$

$\boxed{v}$

For  $v$ , we have  $\tau' = \tau + \text{type}_\tau(v) = \tau[\text{res} \mapsto \tau(v)]$ . Then

$$\begin{aligned}
 C(v) &= \alpha(\{\phi[\text{res} \mapsto \phi(v)] \star \mu \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \\
 (\text{by definition of } \alpha) &= \bigvee \{\alpha(\phi[\text{res} \mapsto \phi(v)] \star \mu) \mid \phi \star \mu \in \gamma(\mathbb{G})\} \\
 (\text{Definition 75}) &= \bigvee \{ (N' \star E' \star \ell' [\text{res} \mapsto \ell'(v)]) \star sh' \star nl' \mid \\
 &\quad \mathbb{G}' \in \alpha\gamma(\mathbb{G})\} \\
 (\text{A6}) &\leq (N \star E \star \ell [\text{res} \mapsto \ell(v)]) \star sh \star nl \\
 &= A(v).
 \end{aligned}$$

$\boxed{(\kappa)v}$

For  $(\kappa)v$ , we have  $\tau' = \tau + \text{type}_\kappa = \tau[\text{res} \mapsto \kappa]$ . We have the following possibilities.

- $\ell(v) = \perp$ . By (A6) for each  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that  $\mathbb{G}' = \alpha(\phi \star \mu) \leq \mathbb{G}$  and therefore  $\ell'(v) = \perp$ . By Definition 52,  $\phi(v) = \text{null}$  and then  $\phi[\text{res} \mapsto \phi(v)] \star \mu = \phi \star \mu$ . In this case

$$\begin{aligned}
 C((\kappa)v) &= \alpha(\{\phi[\text{res} \mapsto \phi(v)] \star \mu \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \\
 (\text{by previous observation}) &= \alpha(\{\phi \star \mu \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \\
 (\text{A7}) &\leq \mathbb{G} \\
 &= A((\kappa)v).
 \end{aligned}$$

- $\ell(v) \neq \perp$  and  $\{\tau_G(\ell(v)), \kappa\}$  is a not chain). By Definition 39,  $\tau_G(\ell(v)) = \bigwedge \{\tau(\ell^{-1}(\ell(v)))\}$ . By (A6) for each  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that  $\mathbb{G}' = \alpha(\phi \star \mu) \leq \mathbb{G}$ . Now, we have two cases
  - $\ell'(v) = \perp$ . In this case,  $\phi(v) = \text{null}$  and  $\mathcal{E}_\tau^I[(\kappa)v](\phi \star \mu) = \phi \star \mu$ . Therefore, by Definition 81 and Proposition 115,  $\alpha(\mathcal{E}_\tau^I[(\kappa)v](\phi \star \mu)) = \mathbb{G}' = \mathbb{G}'_{|v=\text{null}} \leq \mathbb{G}_{|v=\text{null}}$ .
  - $\ell'(v) \neq \perp$ . In this case by Lemma 98 and Theorem 99,  $\tau(\ell^{-1}(\ell'(v))) \cup \{\kappa\}$  is not a chain and then  $\{\tau(\phi(v)), \kappa\}$  is not a chain. Therefore,  $\alpha(\mathcal{E}_\tau^I[(\kappa)v](\phi \star \mu)) = \alpha(\{\text{undefined}\}) = \perp$ .

By previous results

$$\begin{aligned}
 C(v) &= \alpha(\{\mathcal{E}_\tau^I[(\kappa)v](\phi \star \mu) \mid \phi \star \mu \in \gamma(\mathbb{G})\}) \\
 (\text{by definition of } \alpha) &= \bigvee \{\alpha(\mathcal{E}_\tau^I[(\kappa)v](\phi \star \mu)) \mid \phi \star \mu \in \gamma(\mathbb{G})\} \\
 (\text{by previous results}) &= \bigvee \{\alpha(\phi \star \mu) \mid \phi \star \mu \in \gamma(\mathbb{G}) \text{ and } \phi(v) = \text{null}\} \\
 (\text{by Definition 75}) &= \bigvee \{\mathbb{G}' \mid \mathbb{G}' \in \alpha\gamma(\mathbb{G}) \text{ and } \ell'(v) = \perp\} \\
 (\text{by definition}) &\leq \bigvee \{\mathbb{G}'_{|v=\text{null}} \mid \mathbb{G}' \in \alpha\gamma(\mathbb{G})\} \\
 (\text{by (A6) and Proposition 115}) &\leq \mathbb{G}_{|v=\text{null}} \\
 &= A(v).
 \end{aligned}$$

- $\ell(v) \neq \perp$  and  $\{\tau_G(\ell(v)), \kappa\}$  is a chain. By previous results, since  $\mathbb{G} \leq \text{add}(\mathbb{G}, \ell(v), \kappa)$  and by definition of  $\alpha$  we have to prove that for each  $\phi \star \mu \in \gamma(\mathbb{G})$  such that  $\phi(v) \neq \text{null}$  and  $\tau(\phi(v)) \leq \kappa$  we have that  $\alpha(\phi' \star \mu) \leq \text{add}(\mathbb{G}, \ell(v), \kappa)$ , where  $\phi' = \phi[\text{res} \mapsto \phi(v)]$ . The proof follows by Proposition 117.

v.f

We have  $\tau' = \tau + \text{type}_\tau(v.f) = \tau[\text{res} \mapsto \tau(v.f)]$ . We have the following possibilities:

- $\ell(v) = \perp$ . In this case, for each  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that  $\phi(v) = \text{null}$  and then by definition of  $\alpha$ ,

$$C(v.f) = \alpha(\{\text{undefined}\}) = \perp = A(v.f).$$

- $\ell(v) \neq \perp$  and  $\ell(v.f) = \perp$ . In this case for each  $\phi \star \mu \in \gamma(\mathbb{G})$ ,  $\phi(v).f = \text{null}$  and therefore,  $\phi[\text{res} \mapsto (\phi(v).f)] \star \mu = \phi \star \mu$ . Now, the proof is the same of the first case of the previous point and hence it is omitted.
- $\ell(v) \neq \perp$  and  $\ell(v.f) \neq \perp$ . Let  $\phi' = \phi[\text{res} \mapsto (\phi(v).f)]$ . Analogously to the previous point, we have to prove that for each  $\phi \star \mu \in \gamma(\mathbb{G})$  such that  $\phi(v) \neq \text{null}$  and  $\phi(v.f) \neq \text{null}$  we have that  $\alpha(\phi' \star \mu) \leq \text{add}(\mathbb{G}, \ell(v.f), \tau(v.f))$ . The proof follows by Proposition 117, since by Proposition 21,  $\tau(\phi(v.f)) \leq \tau(v.f)$ .

v.m(v<sub>1</sub>, . . . , v<sub>n</sub>)

We have  $\tau' = \tau + \text{type}_\tau(v.m(v_1, \dots, v_n)) = \tau[\text{res} \mapsto \text{returnType}(\tau(v).m)]$  and

$$C(v.m(v_1, \dots, v_n)) = \alpha(\underbrace{\{\phi[\text{res} \mapsto \phi^1(\text{out})] \star \mu^1 \mid \phi(v) \neq \text{null} \text{ and } \phi \star \mu \in \gamma(\mathbb{G})\}}_{\phi'})$$

with  $\sigma^\dagger = [\text{this} \mapsto \phi(v), w_1 \mapsto \phi(v_1), \dots, w_n \mapsto \phi(v_n)] \star \mu$  and  $\phi^1 \star \mu^1 = I((\phi(v).k).m)(\sigma^\dagger)$ . As for v.f above,  $C(v.m(v_1, \dots, v_n))$  is best approximated by  $\perp$  when  $\phi(v) = \text{null}$ . Assume hence that  $\phi(v) \neq \text{null}$ .

First of all, observe that the effective method used in the concrete side is  $(\phi(v).k).m$ . By type correctness, we know that  $\mu(\phi(v)).k \leq \tau_G(\ell(v))$ . Therefore, since the abstract semantics is defined as:

$$\Upsilon \{ \text{match}_{v.m}(\mathbb{G}, I(k.m)(\mathbb{G}')) \mid \kappa \leq \tau_G(\ell(v)) \}$$

we only need to prove that  $\alpha(I((\phi(v).k).m)(\gamma(\mathbb{G}))) \leq \text{match}_{v.m}(\mathbb{G}, I((\phi(v).k).m)(\mathbb{G}'))$ , and it immediately follows that  $\alpha(I((\phi(v).k).m)(\gamma(\mathbb{G}))) \leq \Upsilon \{ \text{match}_{v.m}(\mathbb{G}, I(k.m)(\mathbb{G}')) \mid \kappa \leq \tau_G(\ell(v)) \}$ .

Proving that  $\alpha(I((\phi(v).k).m)(\gamma(\mathbb{G}))) \leq \text{match}_{v.m}(\mathbb{G}, I((\phi(v).k).m)(\mathbb{G}'))$  amounts to show that  $\alpha(\phi[\text{res} \mapsto \phi^1(\text{out})] \star \mu^1) \leq \text{match}_{v.m}(\mathbb{G}, I((\phi(v).k).m)(\mathbb{G}'))$ .

First of all, observe that since  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that  $\alpha(\phi \star \mu) \leq \mathbb{G}$ . Then  $\alpha(\sigma^\dagger) \leq \text{prune}((N \star E \star \ell^{\text{input}}) \star sh \star nl)$  and therefore  $\sigma^\dagger \in \gamma(\text{prune}((N \star E \star \ell^{\text{input}}) \star sh \star nl))$ , where  $\ell^{\text{input}} = [\text{this} \mapsto \ell(v), w_1 \mapsto \ell(v_1), \dots, w_n \mapsto \ell(v_n)]$ . Then, by correctness of  $I'$  with respect to  $I$ ,  $\alpha(\phi^1 \star \mu^1) \leq I'((\phi(v).k).m)(\text{prune}((N \star E \star \ell^{\text{input}}) \star sh \star nl))$ .

On the concrete side, only variables in  $\text{dom}(\sigma^\dagger.\phi)$  can be reached in the computation of  $I((\phi(v).k).m)(\sigma^\dagger)$ . Since  $\text{dom}(\sigma^\dagger.\phi) = \{\text{this}, w_1, \dots, w_n\}$ , it follows that for any variable  $x \in V_{\text{other}}$  we have that  $\phi(x) = \phi'(x)$ . The same happens on the abstract side, since  $\mathbb{G}|_{V_{\text{other}}} = \mathbb{G}_{\text{other}}$ . Moreover, for variables in  $V_{\text{comp}}$ , since on the abstract side we compute  $(\top|_{V_{\text{comp}}})|_{\{x=y \mid \ell^1(x) = \ell^1(y), x, y \in V_{\text{comp}}\}}$ , there is nothing to prove.

For any variable  $x$  in  $V_{\text{alias}}$ , since  $x \notin \text{dom}(\sigma^\dagger.\phi)$  and, by definition of  $V_{\text{alias}}$ , there exists  $u \in V_m$  such that  $\ell(x) = \ell(u)$ , it follows that  $\phi'(x) = \phi'(u)$ , which corresponds to the  $\ell_{\text{alias}}$  mapping in  $\mathbb{G}$ .

Therefore, we only need to show that the result is correct for variables in  $V_m$ , which directly follows from the induction hypothesis that  $I(\kappa.m)(\mathbb{G}')$  is correct w.r.t.  $I((\phi(v).\kappa).m)(\sigma^\dagger)$ .

$$\boxed{v := exp}$$

Since the composition of correct operations is correct and since we have proved above that the abstract semantics for the expressions are correct with respect to their concrete counterparts, it is enough to prove that the abstract *prune* operation

$$P_\tau^v = \lambda \mathbb{G} \in \text{ALPS}. \text{prune}((N \star E \star \ell[v \mapsto \ell(\text{res}), \text{res} \mapsto \perp]) \star sh \star nl)$$

is correct with respect to the corresponding concrete operation

$$\text{setVar}_\tau^v = \lambda(\phi \star \mu) \in \Sigma_\tau. \phi|_{-\text{res}}[v \mapsto \phi(\text{res})] \star \mu$$

Let  $\tau' = \tau|_{-\text{res}}$  and let  $\mathbb{G} \in \text{ALPS}$ . We have

$$C(\text{setVar}_\tau^v) = \alpha(\underbrace{\{\phi|_{-\text{res}}[v \mapsto \phi(\text{res})] \star \mu \mid \phi \star \mu \in \gamma(\mathbb{G})\}}_{\phi'})$$

Then to prove the thesis, by additivity of  $\alpha$ , we have to prove that for each  $\phi \star \mu \in \gamma(\mathbb{G})$ ,

$$\mathbb{G}_1 = \alpha(\phi' \star \mu) \leq \text{prune}((N \star E \star \ell[v \mapsto \ell(\text{res}), \text{res} \mapsto \perp]) \star sh \star nl).$$

Let  $\mathbb{G}' = \alpha(\phi \star \mu)$ . By definition

$$\begin{aligned} \mathbb{G}_1 &= (\{n \in N' \mid \exists i \in \ell'^{-1}(n). i \neq v, i \neq v.f \in Q_\tau\} \star \\ &\quad \{n_1 \xrightarrow{f} n_2 \in E' \mid n_1, n_2 \in N_1\} \star \ell'[v \mapsto \ell'(\text{res}), \text{res} \mapsto \perp]) \star \\ &\quad \{\{n_1, n_2\} \in sh' \mid n_1, n_2 \in N_1\} \star nl' \cap N_1. \end{aligned}$$

Now, the proof is straightforward by definition of  $P_\tau^v$  and since by (A6),  $\mathbb{G}' = \alpha(\phi \star \mu) \leq \mathbb{G}$ .

$$\boxed{v.f := exp}$$

Since the composition of correct operations is correct and since we have proved above that the abstract semantics for the expressions is correct w.r.t. their concrete counterparts, it is enough to prove that the abstract operation

$$P_\tau^{v.f} = \lambda \mathbb{G} \in \text{ALPS}. \begin{cases} \perp & \text{if } \ell(v) = \perp \\ \mathbb{G}^* & \text{if } \ell(v) \neq \perp \text{ and } \ell(\text{res}) = \perp \\ \mathbb{G}^{**} & \text{otherwise,} \end{cases}$$

is correct with respect to the corresponding concrete operation

$$\text{setField}_\tau^{v.f} = \lambda(\phi \star \mu) \in \Sigma_\tau. \begin{cases} \underbrace{\phi|_{-\text{res}} \star \mu[l \mapsto \mu(l)[f \mapsto \phi(\text{res})]]}_{\phi'} & \text{if } (l = \phi(v)) \neq \text{null} \\ \perp & \text{otherwise.} \end{cases}$$

where

$$\begin{aligned} \mathbb{G}^* &= \text{prune} (N \cup N_{new} \star E \setminus E_{del} \cup E_{new} \star \ell \star sh \cup sh_{new} \star \\ &\quad nl \cup \{n_{\ell(x)} \mid n_{\ell(x)} \in N_{new}, \ell(x.f) \in nl\}) \\ \mathbb{G}^{**} &= \text{prune} (N \cup N_{new} \star E \setminus E_{del} \cup E'_{new} \star \ell[\text{res} \mapsto \perp] \star sh \cup sh'_{new} \star \\ &\quad nl \cup nl_{new} \cup \{n_{\ell(x)} \mid n_{\ell(x)} \in N_{new}, \ell(x.f) \in nl\}) \end{aligned}$$



where

$$\begin{aligned}
 V_{comp} &= \{x \in \text{dom}(\tau) \mid \ell(x) \neq \ell(v), \{\ell(x), \ell(v)\} \in sh, \{\tau_G(\ell(x)), \tau_G(\ell(v))\} \\
 &\quad \text{is a chain}\} \\
 N_{new} &= \{n_{\ell(x)} \mid x \in V_{comp}, \mathbf{f} \in \text{dom}(\psi_G(\ell(x))), \ell(x.\mathbf{f}) \neq \ell(\text{res})\} \\
 &\quad \text{a set of new distinct nodes} \\
 E_{del} &= \{\ell(v) \xrightarrow{\mathbf{f}} \ell(v.\mathbf{f})\} \cup \{\ell(x) \xrightarrow{\mathbf{f}} \ell(x.\mathbf{f}) \in E \mid x \in V_{comp}, \ell(x.\mathbf{f}) \neq \ell(\text{res})\} \\
 E_{new} &= \{\ell(x) \xrightarrow{\mathbf{f}} n_{\ell(x)} \mid x \in V_{comp}, n_{\ell(x)} \in N_{new}\} \\
 sh_{new} &= \{\{n_{\ell(x)}, n'\} \mid n_{\ell(x)} \in N_{new}, \{\ell(x.\mathbf{f}), n'\} \in sh\} \cup \\
 &\quad \{\{n_{\ell(x)}, n_{\ell(y)}\} \mid n_{\ell(x)}, n_{\ell(y)} \in N_{new}, \{\ell(x.\mathbf{f}), \ell(y.\mathbf{f})\} \in sh\} \\
 E'_{new} &= E_{new} \cup \{\ell(v) \xrightarrow{\mathbf{f}} \ell(\text{res})\} \\
 sh'_{new} &= \{\{n, n'\} \mid \{\ell(v), n\} \in sh \text{ and } \{\ell(\text{res}), n'\} \in sh\} \cup \\
 &\quad \{\{n_{\ell(x)}, n'\} \mid n_{\ell(x)} \in N_{new}, \{\ell(\text{res}), n'\} \in sh\} \cup \\
 &\quad \{\{n_{\ell(x)}, n'\} \mid n_{\ell(x)} \in N_{new}, \{\ell(x.\mathbf{f}), n'\} \in sh\} \cup \\
 &\quad \{\{n_{\ell(x)}, n_{\ell(y)}\} \mid n_{\ell(x)}, n_{\ell(y)} \in N_{new}\} \\
 nl_{new} &= \begin{cases} \{n \in N \mid \{n, \ell(v)\} \in sh\} \cup \{n_{\ell(x)} \mid n_{\ell(x)} \in N_{new}\} & \text{if } \{\ell(\text{res}), \ell(v)\} \in sh \text{ or } \ell(\text{res}) \in nl \\ \{n \in N \mid \{n, \ell(v)\} \in sh, \{\ell(\text{res}), n\} \in sh\} & \\ \text{otherwise} & \end{cases}
 \end{aligned}$$

Let  $\tau' = \tau|_{-\text{res}}$  and  $\mathbb{G} \in \text{ALPS}$ . We have to prove that

$$\alpha(\text{setField}_{\tau'}^{v.\mathbf{f}}(\gamma(\mathbb{G}))) \leq P_{\tau'}^{v.\mathbf{f}}(\mathbb{G}).$$

We have the following possibilities:

- $\ell(v) = \perp$ . In this case, for each  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that  $\phi(v) = \text{null}$  and then by definition of  $\alpha$ ,

$$\alpha(\text{setField}_{\tau'}^{v.\mathbf{f}}(\gamma(\mathbb{G}))) = \alpha(\{\text{undefined}\}) = \perp = P_{\tau'}^{v.\mathbf{f}}(\mathbb{G}).$$

- $\ell(v) \neq \perp$  and  $\ell(\text{res}) = \perp$ . By (A6) for each  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that  $\mathbb{G}' = \alpha(\phi \star \mu) \leq \mathbb{G}$  and therefore  $\ell'(\text{res}) = \perp$ . By Definition 75,  $\phi(\text{res}) = \text{null}$  and then  $\phi' = \phi$  and  $\phi'(v).\mathbf{f} = \text{null}$ . Let  $\mathbb{G}_1 = \alpha(\phi' \star \mu')$ . By definition of  $\alpha$ ,

$$\begin{aligned}
 \mathbb{G}_1 &= (\{l \in \text{Loc} \mid \exists i \in I_{\tau}.\phi(i) = l \text{ and either } i \neq w.\mathbf{f} \text{ or } \phi(v) \neq \phi(w)\} \star \\
 &\quad \{l \xrightarrow{\mathbf{f}'} l' \mid l.\mathbf{f}' = l' \in N' \text{ and either } v \notin \phi^{-1}(l) \text{ or } \mathbf{f}' \neq \mathbf{f}\} \star \phi),
 \end{aligned}$$

$sh_1 \subseteq \{\{n, n'\} \in sh' \mid n, n' \in N_1\}$  and  $nl_1 \subseteq nl' \cap N_1$ . Now the proof is straightforward by definition of  $\mathbb{G}^*$  and since  $\mathbb{G}' \leq \mathbb{G}$ .

- $\ell(v) \neq \perp$  and  $\ell(\text{res}) \neq \perp$ . By definition of  $\alpha$  and by previous results, we have to prove that for each  $\phi \star \mu \in \gamma(\mathbb{G})$  such that  $\phi(v) \neq \text{null}$  we have that

$$\mathbb{G}_1 = \alpha(\phi' \star \mu') \leq \mathbb{G}^{**},$$

where  $\mathbb{G}^{**}$  is defined as in definition of  $P_{\tau'}^{v.\mathbf{f}}$ .

Let  $\phi \star \mu \in \gamma(\mathbb{G})$  such that  $\phi(v) \neq \text{null}$  and let  $\mathbb{G}' = \alpha(\phi \star \mu)$ . By (A6),  $\mathbb{G}' \preceq \mathbb{G}$ . Then, by definition of  $\alpha$ ,

$$G_1 = \{l \in \text{Loc} \mid \exists i \in I_\tau. \phi(i) = l \text{ and either } i \neq w.f \text{ or } \phi(v) \neq \phi(w)\} \star$$

$$(\{l \xrightarrow{f'} l' \mid l.f' = l' \in N' \text{ and either } v \notin \phi^{-1}(n) \text{ or } f' \neq f\} \cup$$

$$\{\phi(v) \xrightarrow{f} \phi(\text{res}) \mid \phi(\text{res}) \neq \text{null}\}) \star \phi[\text{res} \mapsto \perp],$$

$$sh'_1 \subseteq \{\{l, l'\} \in sh' \mid l, l' \in N_1\} \cup$$

$$\{\{l, l'\} \mid l, l' \in N_1, \{\phi(v), l\} \in sh' \text{ and } \{\phi(\text{res}), l'\} \in sh'\},$$

$$nl'_1 \subseteq (nl' \cap N_1) \cup$$

$$\{\phi'(v) \mid \phi'(v) \text{ is not linear in } \phi' \star \mu'\} \cup$$

$$\{l \mid l \text{ and } \phi'(v) \text{ share in } \phi' \star \mu', \phi'(v) \text{ is not linear in } \phi' \star \mu'$$

$$\text{and } \phi(v) \text{ is linear in } \phi \star \mu\}$$

and  $\mathbb{G}_1 = \text{cl}^\uparrow(G_1, sh'_1, nl'_1)$ . Now, the proof is straightforward, by definition of  $\mathbb{G}^{**}$  and since  $\mathbb{G}' \preceq \mathbb{G}$ .

$\text{if } v = w \text{ then } com_1 \text{ else } com_2$

Analogously to the previous case, we have to prove that

$$A_{\text{if}_{\text{eq}}} = \lambda \mathbb{G} \in \text{ALPS}. \begin{cases} \mathcal{S}\mathcal{C}_\tau^I[\text{com}_1](\mathbb{G}) & \text{if } \ell(v) = \ell(w) \\ \mathcal{S}\mathcal{C}_\tau^I[\text{com}_1](\mathbb{G}_{|v=w}) \Upsilon \mathcal{S}\mathcal{C}_\tau^I[\text{com}_2](\mathbb{G}) & \text{otherwise} \end{cases}$$

is correct with respect to the corresponding concrete operation

$$C_{\text{if}_{\text{eq}}} = \lambda(\phi \star \mu) \in \Sigma_\tau. \begin{cases} \mathcal{C}_\tau^I[\text{com}_1](\phi \star \mu) & \text{if } \phi(v) = \phi(w) \\ \mathcal{C}_\tau^I[\text{com}_2](\phi \star \mu) & \text{if } \phi(v) \neq \phi(w) \end{cases}$$

Let  $\mathbb{G} \in \text{ALPS}$ . We have the following possibilities:

- $\ell(v) = \ell(w)$ . In this case for each  $\phi \star \mu \in \gamma(\mathbb{G})$ , we have that  $\phi(v) = \phi(w)$ . Therefore,

$$\alpha(C_{\text{if}_{\text{eq}}}(\gamma(\mathbb{G})))$$

(by previous observation and

$$\text{by definition of } \alpha) = \alpha(\mathcal{C}_\tau^I[\text{com}_1](\gamma(\mathbb{G})))$$

(by definition) =  $C(\text{com}_1)$

(by induction)  $\preceq A(\text{com}_1)$

(by definition) =  $A_{\text{if}_{\text{eq}}}(\mathbb{G})$ .

- $\ell(v) \neq \ell(w)$ . In this case

$$\begin{aligned} & \alpha(C_{\text{if}_{\text{eq}}}(\gamma(\mathbb{G}))) \\ \text{(by definition)} &= \alpha \left( \begin{array}{c} \mathcal{E}_\tau^I[\text{com}_1](\{\phi \star \mu \in \gamma(\mathbb{G}) \mid \phi(v) = \phi(w)\}) \\ \cup \\ \mathcal{E}_\tau^I[\text{com}_2](\{\phi \star \mu \in \gamma(\mathbb{G}) \mid \phi(v) \neq \phi(w)\}) \end{array} \right) \\ \\ \text{(by definition of } \alpha) &= \alpha(\mathcal{E}_\tau^I[\text{com}_1](\{\phi \star \mu \in \gamma(\mathbb{G}) \mid \phi(v) = \phi(w)\}) \curlywedge \\ & \quad \alpha(\mathcal{E}_\tau^I[\text{com}_2](\{\phi \star \mu \in \gamma(\mathbb{G}) \mid \phi(v) \neq \phi(w)\})) \\ \\ \text{(by Proposition 84 and monotonicity)} &\leq \alpha(\mathcal{E}_\tau^I[\text{com}_1](\{\phi \star \mu \in \gamma(\mathbb{G}_{|v=w})\})) \curlywedge \\ & \quad \alpha(\mathcal{E}_\tau^I[\text{com}_2](\{\phi \star \mu \in \gamma(\mathbb{G})\})) \\ \\ \text{(by induction)} &\leq \mathcal{S}\mathcal{E}_\tau^I[\text{com}_1](\mathbb{G}_{|v=w}) \curlywedge \mathcal{S}\mathcal{E}_\tau^I[\text{com}_2](\mathbb{G}) \\ \text{(by definition)} &= A_{\text{if}_{\text{eq}}}(\mathbb{G}). \end{aligned}$$

if  $v = \text{null}$  then  $\text{com}_1$  else  $\text{com}_2$

The proof is analogous to that one of the previous case and hence it is omitted.

$\{\text{com}_1; \dots; \text{com}_p\}$

The proof of this case follows directly by induction hypothesis. □

**Theorem 88.** *The abstract denotational semantics is correct wrt the concrete one.*

*Proof.* To prove the correctness of the denotational semantics is enough to prove the correctness of the abstract transformer. Since composition preserves correctness it is sufficient to prove that

- (1)  $\lambda \mathbb{G} \in \text{ALPS}_{\text{scope}(\kappa.m)} \cdot \mathbb{G} \parallel \text{dom}(\text{output}(\kappa.m))$  is correct wrt  $\lambda \sigma \in \Sigma_{\text{scope}(\kappa.m)} \cdot \sigma \parallel \text{dom}(\text{output}(\kappa.m)) = \lambda \phi \star \mu \in \Sigma_{\text{scope}(\kappa.m)} \cdot \phi \parallel \text{dom}(\text{output}(\kappa.m)) \star \mu$ ;
- (2)  $\mathcal{S}\mathcal{E}_{\text{scope}(\kappa.m)}^I[\text{body}(\kappa.m)]$  is correct wrt  $\mathcal{E}_{\text{scope}(\kappa.m)}^I[\text{body}(\kappa.m)]$ ;
- (3)  $\lambda \mathbb{G} \in \text{ALPS}_{\text{input}(\kappa.m)} \cdot N \star E \star \ell[w'_1 \mapsto \ell(w_1), \dots, w'_n \mapsto \ell(w_n)] \star sh \star nl$  is correct wrt  $\lambda \phi \star \mu \in \Sigma_{\text{input}(\kappa.m)} \cdot \phi[\text{out} \mapsto \text{null}, w'_1 \mapsto \phi(w_1), \dots, w'_n \mapsto \phi(w_n), w_{n+1} \mapsto \text{null}, \dots, w_{n+m} \mapsto \text{null}] \star \mu$ .

The first and the second point are the content of Proposition 80 and Theorem 87, respectively. The third point is trivial since adding null variables does not add any node or edge to the graph. □