Projet Ajacs

Deliverable WP2

Abstract domains and program logics for prototype chains and reflection

August 2017

The attached paper, submitted for publication, presents a framework to systematically derive correct abstract interpretations of a programming language that includes separation logic, required to reason about heaps of JavaScript and prototype chains.

1

From Typed Natural Semantics to Abstract Interpretation with a Frame Rule

ANONYMOUS AUTHOR(S)

This paper provides a framework for deriving program logics from big-step operational semantics using principles from abstract interpretation. The framework is based on a type-based formalisation of rule formats in operational semantics. This formalisation can be given both a standard (inductive) interpretation and an abstract interpretation. We identify local correctness conditions under which we can provide a general soundness result for abstract interpretation. We then prove that our program logics can be extended with the frame rule from separation logic, thus demonstrating how partial order-based abstract interpretation can be combined with the spatial conjunction of separation logic.

ACM Reference format:
Anonymous Author(s). 2017. From Typed Natural Semantics to Abstract Interpretation with a Frame Rule. *Proc. ACM Program. Lang.* 1, 1, Article 1 (January 2017), 27 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

This paper addresses the question of how to combine two major strands of program analysis, abstract interpretation and separation logic, within a common formal framework for defining program analyses. We present a framework that allows the design of program analyses in a systematic fashion, starting from a semantic description of the language to analyse. The framework offers a general principle for proving the correctness of such analyses, with respect to a formal definition of the underlying programming language, a task that is challenging for any non-trivial language.

Abstract interpretation (AI) provides a general framework for the design of semantics-based program analyses, and offers principles for proving the correctness of a static analysis. Separation logic offers a rich formalism for precise and modular reasoning about programs based on its spatial conjunction and the frame rule. However, it has also been recognised [Distefano et al. 2006] that fitting separation logic-based analysis into a framework based on AI is not straightforward. We show how to remedy this, taking an approach based on the abstract interpretation of big-step operational semantics.

Abstract interpretation has been combined with operational, denotational, and axiomatic semantics, in order to prove the correctness of a variety of existing analyses of imperative, functional, logical, object-oriented, and concurrent languages. AI suggests a methodology by which static program analyses can be derived in a systematic fashion from a semantics of the programming language, by composing the semantic function with the abstraction and concretisation functions defining a particular AI. Such derivational abstract interpretation offers a road to correct-by-construction program analyses. The approach has been shown viable on a small imperative language [Cousot 1999] and on idealised functional languages [Midtgaard and Jensen 2008; Van Horn and Might 2010], all starting from small-step operational semantics.

In this paper we show that derivational abstract interpretation can be based on big-step (natural) operational semantics. The framework includes advanced analysis features such as trace partitioning, and permits the analysis to use local reasoning akin to the separating conjunction and

2017. 2475-1421/2017/1-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the frame rule from Separation Logic. To achieve this, we generalise the principles of abstract interpretation of big-step semantics, laid out by Schmidt [Schmidt 1995] and refined by Bodin *et al.* [Bodin et al. 2015]. Schmidt [Schmidt 1995] targeted the classical abstract interpretation framework, in which abstract domains are modelled as complete lattices and the abstract interpretation of a program is defined in terms of iteration towards a least fixed point. Since then, the need for a more general framework with more flexibility on abstract domains and interpretations has become manifest. Industrial-strength analyzers such as Astrée [Cousot et al. 2005] and Facebook's Infer analyzer based on separation logic [Calcagno et al. 2015; Facebook 2015] both operate over partial orders with no apparent lattice structure, and infer information using more sophisticated extrapolation mechanisms.

Outline

We shall revisit big-step operational semantics and the rule format used to specify such semantics in Section 2. The whole approach that we put forward here is based on a formalisation of the rule format relying on types and sorts. We define a typing discipline of inference rules and prove a theorem that well-typed semantics will always be defined. Notice that it is the typing of the operational semantics that is important and that the framework applies equally well to typed and untyped programming languages. A distinguishing feature of the type system is that it allows us to speak about the resources necessary for evaluating a program term. This becomes important for the inclusion of spatial conjunction and a frame rule, as described below.

Given a well-typed big-step semantics, we are then in a position to define the notions of standard (concrete) and abstract interpretations. The standard interpretation is the usual inductive interpretation of inference rules. The abstract interpretation of the rules, defined in Section 3, differs in two essential ways:

- rules are interpreted co-inductively, and
- *all* applicable rules must be applied to a term in order to deduce an abstract property.

This second requirement (which intuitively can be seen as an inference rule equivalent of the "*meet-over-all-paths*" principle from data flow analysis) is necessary for the correctness of the analysis.

The resulting abstract interpretation is obtained in a completely systematic fashion: the resulting analysis consists of a set of syntax-directed rules, that are in one-to-one correspondence with the rules of the standard semantics, *plus* some extra rules that expresses general analysis principles such as weakening of results and trace partitioning. In Section 4 we show how these *non-structural* rules can be included whilst still maintaining the correctness of the analysis.

A major result about this framework, established in Section 5, is that it offers an integration of abstract domains from abstract interpretation with spatial conjunction and the frame rule from separation logic. This is a novel result which has not been established before. Having a frame rule for abstract interpretation means that we can locally reason with the usual abstract interpretation over the relevant part of a program state and then extend this result to the entire state, provided certain well-identified conditions are identified. To prove this result, we make essential use of our novel typing discipline on our big-step semantics, to keep account of the resources needed for an interpretation of a part of a program.

To sum up, the contributions of the paper are:

- A type-based formalisation of the rule format underlying big-step operational semantics, which permits us to be precise about resources and that allows us to state a "well-typed semantics are always defined" result.

$$\textsc{LitInt}\ \frac{}{H, n \Downarrow n} \qquad \textsc{Var}\ \frac{\mathsf{x} \in dom(H)}{H, \mathsf{x} \Downarrow H(\mathsf{x})} \qquad \textsc{Add}\ \frac{H, e_1 \Downarrow v_1 \qquad H, e_2 \Downarrow v_2}{H, e_1 + e_2 \Downarrow v_1 + v_2}$$

$$\textsc{EqTrue}\ \frac{H, e_1 \Downarrow v_1 \qquad H, e_2 \Downarrow v_2 \qquad v_1 = v_2}{H, e_1 = e_2 \Downarrow \mathtt{tt}} \qquad \textsc{EqFalse}\ \frac{H, e_1 \Downarrow v_1 \qquad H, e_2 \Downarrow v_2 \qquad v_1 \neq v_2}{H, e_1 = e_2 \Downarrow \mathtt{ff}}$$

$$\textsc{NegTrue}\ \frac{H, e \Downarrow \mathtt{tt}}{H, \neg e \Downarrow \mathtt{ff}} \qquad \textsc{NegFalse}\ \frac{H, e \Downarrow \mathtt{ff}}{H, \neg e \Downarrow \mathtt{tt}}$$

$$\textsc{Asn}\ \frac{\mathsf{x} \in dom(H) \qquad H, e \Downarrow v}{H, \mathsf{x} := e \Downarrow H[\mathsf{x} \leftarrow v]} \qquad \textsc{Seq}\ \frac{H_0, s_1 \Downarrow H_1 \qquad H_1, s_2 \Downarrow H_2}{H_0, s_1;\ s_2 \Downarrow H_2} \qquad \textsc{IfTrue}\ \frac{H, e \Downarrow \mathtt{tt} \qquad H, s_1 \Downarrow H'}{H, \mathit{if}\, e\, s_1\, s_2 \Downarrow H'}$$

$$\textsc{IfFalse}\ \frac{H, e \Downarrow \mathtt{ff} \qquad H, s_2 \Downarrow H'}{H, \mathit{if}\, e\, s_1\, s_2 \Downarrow H'} \qquad \textsc{WhTrue}\ \frac{H_0, e \Downarrow \mathtt{tt} \qquad H_0, s \Downarrow H_1 \qquad H_1, \mathit{while}\, e\, s \Downarrow H_2}{H_0, \mathit{while}\, e\, s \Downarrow H_2} \qquad \textsc{WhFalse}\ \frac{H, e \Downarrow \mathtt{ff}}{H, \mathit{while}\, e\, s \Downarrow H}$$

Fig. 1. Example of a concrete semantics

- An improvement of Schmidt's framework for abstract interpretation of natural semantics which accommodates more general abstract domains and advanced reasoning principles such as trace partitioning, and which makes the abstraction process completely systematic.
- A frame rule for abstract interpretation and a general theorem stating the conditions under which the frame rule can be freely integrated in a logical specification of a program analysis. This provides a new kind of modularity in abstract interpretation, where abstractions that are deduced locally can be transferred to properties about the global behaviour of a program.

2 TYPED BIG-STEP SEMANTICS

2.1 Rule Format

Thorough this paper, we use examples built from the concrete semantics presented in Figure 1. We however insist that our framework is generic and can be applied on any big-step semantics. For simplicity, we often write n instead of $const(n)$ and x instead of $var(\mathsf{x})$ for the syntactical objects.

Inference rules play a central role in this work so we shall be precise about the format of rules. In the following we write $\overrightarrow{x_n}$ for $x_1, \ldots, x_n$ and $\overrightarrow{f(x_n)}$ for $f(x_1), \ldots, f(x_n)$. To formally describe rules, we first define *terms*. We assume given a finite set of constructors c with a signature $\overrightarrow{s_n} \to s_c$, where each s_i is a *sort*. The arity of c is n. In standard languages, sorts include the syntactic categories of expressions, syntactic variables, literals, and statements. Terms also include *term variables*. A new term is built by applying a constructor of signature $\overrightarrow{s_n} \to s_c$ to n terms $\overrightarrow{t_n}$, each of the expected sort. A term with no variable is *closed*.

Formally, a (big-step) rule has the form:

$$Rule\left(c\overrightarrow{x_{t_n}}\right) : \mathtt{list}\ \mathsf{H}$$

where c is a constructor and H is the type of hypotheses. The variables in the list of hypotheses are all included in $X = \overrightarrow{x_{t_n}} \cup \overrightarrow{x} \cup \{x_\sigma, x_o\}$, where x_σ is the semantic variable associated to the initial semantic context, x_o is the semantic variable associated to the result, and $\overrightarrow{x}$ are additional semantic variables. Hypotheses are either a semantic fact $D(x_1, t', x_2)$, a predicate variables $P(x_1, \ldots, x_n)$, or a computation on variables $f(x_1, \ldots, x_n) = v$. Here, $D(x_1, t', x_2)$ represents a semantic derivation

$$
\begin{aligned}
\textsc{LitInt}(\mathit{const}\,(x_n)) &= [\mathtt{numToInt}(x_n) = x_o] \\
\textsc{Var}(\mathit{var}\,(x_\mathsf{X})) &= [\mathtt{getName}(x_\mathsf{X}) = x_1; \mathtt{isInDom}(x_\sigma, x_1); \mathtt{read}(x_\sigma, x_1) = x_o] \\
\textsc{Add}(x_{e_1} + x_{e_2}) &= [D(x_\sigma, x_{e_1}, x_1); \mathtt{isInt}(x_1); D(x_\sigma, x_{e_2}, x_2); \mathtt{isInt}(x_2); \mathtt{add}(x_1, x_2) = x_o] \\
\textsc{EqTrue}(x_{e_1} = x_{e_2}) &= [D(x_\sigma, x_{e_1}, x_1); \mathtt{isInt}(x_1); \mathtt{toInt}(x_1) = x_1'; \\
&\qquad D(x_\sigma, x_{e_2}, x_2); \mathtt{isInt}(x_2); \mathtt{toInt}(x_2) = x_2'; \mathtt{eq}(x_1', x_2') = x_o] \\
\textsc{EqFalse}(x_{e_1} = x_{e_2}) &= [D(x_\sigma, x_{e_1}, x_1); \mathtt{isInt}(x_1); \mathtt{toInt}(x_1) = x_1'; \\
&\qquad D(x_\sigma, x_{e_2}, x_2); \mathtt{isInt}(x_2); \mathtt{toInt}(x_2) = x_2'; \mathtt{neq}(x_1', x_2') = x_o] \\
\textsc{NegTrue}(\neg x_e) &= [D(x_\sigma, x_e, x_1); \mathtt{isTrue}(x_1); \mathtt{false}() = x_o] \\
\textsc{NegFalse}(\neg x_e) &= [D(x_\sigma, x_e, x_1); \mathtt{isFalse}(x_1); \mathtt{true}() = x_o] \\
\textsc{Asn}(x_\mathsf{X} := x_e) &= [\mathtt{getName}(x_\mathsf{X}) = x_1; \mathtt{isInDom}(x_\sigma, x_1); D(x_\sigma, x_e, x_2); \mathtt{write}(x_\sigma, x_1, x_2) = x_o] \\
\textsc{Seq}(x_{s_1};\ x_{s_2}) &= [D(x_\sigma, x_{s_1}, x_1); D(x_1, x_{s_2}, x_o)] \\
\textsc{IfTrue}(\mathit{if}\, x_e\ x_{s_1}\ x_{s_2}) &= [D(x_\sigma, x_e, x_1); \mathtt{isTrue}(x_1); D(x_\sigma, x_{s_1}, x_o)] \\
\textsc{IfFalse}(\mathit{if}\, x_e\ x_{s_1}\ x_{s_2}) &= [D(x_\sigma, x_e, x_1); \mathtt{isFalse}(x_1); D(x_\sigma, x_{s_2}, x_o)] \\
\textsc{WhTrue}(\mathit{while}\, x_e\ x_s) &= [D(x_\sigma, x_e, x_1); \mathtt{isTrue}(x_1); D(x_\sigma, x_s, x_2); D(x_2, \mathit{while}\, x_e\ x_s, x_o)] \\
\textsc{WhFalse}(\mathit{while}\, x_e\ x_s) &= [D(x_\sigma, x_e, x_1); \mathtt{isFalse}(x_1); \mathtt{id}\,(x_\sigma) = x_o]
\end{aligned}
$$

Fig. 2. Rules of Figure 1 translated into this formalism

for the term t' and every x_i is in X. In the following, we assume there is at least one rule for each constructor c. Figure 2 shows the representation of each semantic rule of Figure 1 in this format.

Definition 2.1. A rule L for term $c\overrightarrow{x_{t_n}}$ is *well-formed* iff $\mathtt{wf}\left(\{x_\sigma\} \cup \overrightarrow{x_{t_n}}\right)(L)$ where

$$
\begin{aligned}
\mathtt{wf}\,(K)\,([]) &\iff x_\sigma \in K \wedge x_o \in K \\
\mathtt{wf}\,(K)\,(D(x_1, t', x_2) :: tl) &\iff x_1 \in K \wedge x_2 \notin K \wedge \mathtt{wf}\,(K \uplus \{x_2\})\,(tl) \\
\mathtt{wf}\,(K)\left(P(\overrightarrow{x_n}) :: tl\right) &\iff \forall i \in [1..n].\ x_i \in K \wedge \left(\forall i, j \in [1..n].\ x_i = x_j \implies i = j\right) \wedge \mathtt{wf}\,(K)\,(tl) \\
\mathtt{wf}\,(K)\left(f(\overrightarrow{x_n}) = x :: tl\right) &\iff x \notin K \wedge \forall i \in [1..n], x_i \in K \wedge \mathtt{wf}\,(K \uplus \{x\})\,(tl)
\end{aligned}
$$

Well-formedness ensures that a variable used in one of the hypotheses of rule L is either the initial semantic context, one of the variables of the initial term, or has been defined by a hypothesis that precedes it in the list of hypotheses. We also require variables used as arguments of a predicate to be all different. It is not a strong restriction as we can duplicate variables using the identity function: $\mathtt{id}\,(x_i) = x_i'$.

2.2 Typing of Semantic Rules

We impose a typing discipline on rules, even if the underlying language is untyped. Types $\tau \in \mathcal{T}$ are left abstract, we only require that term sorts are included in types. For every closed term t we give a set of pair of types, which we write $\mathcal{R}\,(t)$. Each such pair describes the evaluation of a term. For instance, if expressions e_1 and e_2 are closed, then $\mathcal{R}\,(e_1 + e_2) = \{(\mathit{state}, \mathit{int})\}$, indicating that we need a state to evaluate e_1 and e_2 and that the result (if any) is an integer. For every sort s, we specify a set of pairs of types, which we write $\mathcal{R}\,(s)$, so we can specify, *e.g.*, that expressions either map states to integers or they map states to booleans. This relation must be consistent: if t is of sort s_t, then $\mathcal{R}\,(t) \subseteq \mathcal{R}\,(s_t)$. For every predicate P, we are given a set of pairs of tuples, $\mathcal{R}\,(P)$. If n is the

arity of the predicate, these tuples contain n types. The first tuple indicates where the predicate is defined, and the second tuple is an approximation of where the predicate is true. For instance, the predicate `isInt` has type $\{((int), (int)), ((bool), (int))\}$: it may take as argument either a boolean or an integer, and if it is true, its argument is an integer. For every function f of arity n, we are given a set $\mathcal{R}(f)$ of pairs $\left(\overrightarrow{\tau_n}, \tau\right)$, whose first (input) component is a tuple of size n of types and whose second (output) component is a type. As expected, $\overrightarrow{\tau_n}$ is the input type of the function and τ is its return type.

We define an operator to compute sets of pairs of types from a list of hypotheses L, under the assumption that $\mathtt{wf}(K)(L)$, written $\mathtt{ty}\left(\overline{\Gamma}\right)(L)$. Here, $\overline{\Gamma}$ is a non-empty set of functions Γ mapping variables from $K \uplus \left\{x_\sigma^i\right\}$ to types, where x_σ^i is a fresh variable used to remember the type initially associated to x_σ, which may change because of predicates. As for `wf`, `ty` is meant to be used on the hypotheses L of a rule applying to a term $\overrightarrow{ct_n}$, with $\overline{\Gamma}$ mapping the variables $\left\{x_\sigma, x_\sigma^i\right\} \cup \overrightarrow{x_{t_n}}$ to their respective possible types.

$$\mathtt{ty}\left(\overline{\Gamma}\right)([]) = \left\{\left(\Gamma\left(x_\sigma^i\right), \Gamma(x_o)\right) \middle| \Gamma \in \overline{\Gamma}\right\}$$

$$\mathtt{ty}\left(\overline{\Gamma}\right)(D(x_1, t', x_2) :: L) = \begin{cases} \emptyset \text{ if } \exists \Gamma \in \overline{\Gamma}.\ \forall \tau_2.\ (\Gamma(x_1), \tau_2) \notin \mathcal{R}(t') \\ \mathtt{ty}\left(\left\{\Gamma + x_2 \mapsto \tau_2 \middle| \Gamma \in \overline{\Gamma} \wedge (\Gamma(x_1), \tau_2) \in \mathcal{R}(t')\right\}\right)(L) \text{ otherwise} \end{cases}$$

$$\mathtt{ty}\left(\overline{\Gamma}\right)\left(P(\overrightarrow{x_n}) :: L\right) = \begin{cases} \emptyset \text{ if } \exists \Gamma \in \overline{\Gamma}.\ \forall \overrightarrow{\tau_n}.\ \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \notin \mathcal{R}(P) \\ \mathtt{ty}\left(\left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \middle| \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)\right\}\right)(L) \text{ otherwise} \end{cases}$$

$$\mathtt{ty}\left(\overline{\Gamma}\right)\left(f(\overrightarrow{x_n}) = x :: L\right) = \begin{cases} \emptyset \text{ if } \exists \Gamma \in \overline{\Gamma}.\ \forall \tau.\ \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \notin \mathcal{R}(f) \\ \mathtt{ty}\left(\left\{\Gamma + x \mapsto \tau \middle| \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)\right\}\right)(L) \text{ otherwise} \end{cases}$$

Intuitively, this operator computes a set of input types and output types compatible with the definition. In all cases, it first checks if the types in the environment are compatible with the construct under consideration. If not, it returns the empty set. In the predicate case, it then modifies the environment to include information given by the predicate. For instance, the predicate `isInt` has types $\{((int), (int)), ((bool), (int))\}$. If the predicate holds, then the environment is refined to say that the corresponding variable is an integer. Note that the type environment $\Gamma + \overrightarrow{x_n \mapsto \tau_n}$ makes sense as all the variables x_n are supposed different by $\mathtt{wf}(K)(L)$.

Example 2.2 (Typing of semantic rules). We take as example the semantics in Figure 1. Sorts are *expr*, *stat*, *lit*, and *var*. As the last two are never directly evaluated, we have $\mathcal{R}(lit) = \mathcal{R}(var) = \emptyset$. The term constructors have the following signatures:

$$const : lit \to expr \qquad var : var \to expr \qquad + : (expr, expr) \to expr \qquad = : (expr, expr) \to expr$$

$$\neg : expr \to expr \qquad := : (var, expr) \to stat \qquad ; : (stat, stat) \to stat$$

$$if : (expr, stat, stat) \to stat \qquad while : (expr, stat) \to stat$$

Additional types are *int*, *bool*, *ident*(x), *state*, and *state*(x), for any identifier x. Intuitively, *int* and *bool* are the types of values (integers and booleans). The type *ident*(x) is used for a variable whose name is x: we both have *var*(x) : *var* and *var*(x) : *ident*(x). The type *state* is for a heap, and the type *state*(x) is for a heap where x is bound.

s, t, P, or f	$\mathcal{R}(s)$, $\mathcal{R}(t)$, $\mathcal{R}(P)$, or $\mathcal{R}(f)$
expr	$\{(state, int), (state, bool)\}$
stat	$\{(state, state)\}$
$const(n)$	$\{(state, int)\}$
$var(\mathtt{x})$	$\{(state, int), (state, bool)\}$
$e_1 + e_2$	$\{(state, int)\}$
$e_1 = e_2$	$\{(state, bool)\}$
$\mathtt{x} := e$	$\{(state, state)\}$
$s_1; s_2$	$\{(state, state)\}$
$if\, e\, s_1\, s_2$	$\{(state, state)\}$
$while\, e\, s$	$\{(state, state)\}$
`numToInt`	$\{((lit), (int))\}$
`toInt`	$\{((int), int)\}$
`getName`	$\bigcup_{\mathtt{x}} \{((var), ident(\mathtt{x}))\}$
`read`	$\bigcup_{\mathtt{x}, \tau \in \{int, bool\}} \{((state(\mathtt{x}), ident(\mathtt{x})), \tau)\}$
`add`	$\{(int, int), int\}$
`eq`	$\{((int, int), bool)\}$
`neq`	$\{((int, int), bool)\}$
`true`	$\{((), bool)\}$
`false`	$\{((), bool)\}$
`write`	$\bigcup_{\mathtt{x}, \tau \in \{int, bool\}} \{((state(\mathtt{x}), ident(\mathtt{x}), \tau), state)\}$
`id`	$\bigcup_{\tau \in \mathcal{T}} \{(\tau), \tau\}$
`isInDom`	$\bigcup_{\mathtt{x}} \{((state, ident(\mathtt{x})), (state(\mathtt{x}), ident(\mathtt{x})))\}$
`isInt`	$\bigcup_{\tau \in \{bool, int\}} \{((\tau), (int))\}$
`isTrue`	$\bigcup_{\tau \in \{bool, int\}} \{((\tau), (bool))\}$
`isFalse`	$\bigcup_{\tau \in \{bool, int\}} \{((\tau), (bool))\}$

Fig. 3. Typing for the example

The entire typing of the semantics is given in Figure 3. The function `toInt` might seems useless. It is indeed the identity in the concrete interpretation of the semantics, but it is more complex in the abstract interpretation, as shown in Example 3.5.

We then require that rules are compatible with the types of their terms. In the following, we write s_σ for the sort associated to environments, and we write $Rule\left(\overrightarrow{ct_n}\right)$ for the list of hypotheses $Rule\left(\overrightarrow{cx_{t_n}}\right)$ where every $D(x_1, t', x_2)$ is replaced by $D\left(x_1, t'\left\{\overrightarrow{t_n/x_n}\right\}, x_2\right)$.

Definition 2.3. Let c be a constructor with signature $\overrightarrow{s_n} \to s_c$. A rule $Rule\left(\overrightarrow{cx_{t_n}}\right)$ is *well-typed* iff for every closed terms $\overrightarrow{t_n : s_n}$, we have $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \subseteq \mathcal{R}\left(\overrightarrow{ct_n}\right)$, where the set $\overline{\Gamma}$ is $\left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$.

In the following we assume that the rules of the considered semantics are well-typed. This is the case for the rules of Figure 2. We can now focus on the interpretation of rules.

2.3 Interpretations of a Semantics

An *interpretation* of a semantics consists of

- domains of semantic objects, such as states *State* and output values *Out*,
- a relation between semantic objects and types, written $v : \tau$,
- a mapping from predicate and function symbols to actual predicates and functions over the semantic domains, written $\mathtt{I}\,(P)$ and $\mathtt{I}\,(f)$ for an interpretation $\mathtt{I}$,
- and a function computing the interpretation of a rule, as detailed below. This function is written $[\![L]\!]_T\,(\Sigma)$ for the concrete interpretation and $[\![L]\!]_T^\sharp\,(\Sigma)$ for the abstract one. When the interpretation does not matter, we write $[\![L]\!]_T^?\,(\Sigma)$.

Interpretations share the set *Terms* of closed terms.

Example 2.4. The concrete interpretation of the example of Figure 1 is as follows. A value is either an integer, of type *int*, or a boolean, of type *bool*. We assume a countable set of variables. A state H : *state* is a partial function from variables to values. A state H has type *state* (x) if $\mathsf{x} \in dom\,(H)$. The interpretation of every predicate and function is straightforward.

A *semantic triple* $(\sigma, t, o) \in State \times Terms \times Out$ should be read as "term t produces result o when evaluated in state σ". Results of computations can be either a new state or a value, and for the moment we shall not specify *State* and *Out* further. In the following, we let T range over sets of semantic triples.

An interpretation is subject to the following consistency requirements: every semantic object has at least one type: $\forall v. \exists \tau. v : \tau$. In addition, for any closed term t of sort s, we have $t : s$. Let $\left(\overrightarrow{\tau_n}, \overrightarrow{\tau'_n}\right) \in \mathcal{R}\,(P)$. For every $\overrightarrow{v_n}$ such that $\overrightarrow{v_n : \tau_n}$, then $\mathtt{I}\,(P)\left(\overrightarrow{v_n}\right)$ is defined (but may not hold). In addition, if $\mathtt{I}\,(P)\left(\overrightarrow{v_n}\right)$ holds, then $\overrightarrow{v_n : \tau'_n}$. For functions, Let $\left(\overrightarrow{\tau_n}, \tau\right) \in \mathcal{R}\,(f)$. For every $\overrightarrow{v_n}$ such that $\overrightarrow{v_n : \tau_n}$, then $\mathtt{I}\,(f)\left(\overrightarrow{v_n}\right)$ is defined and such that $\mathtt{I}\,(f)\left(\overrightarrow{v_n}\right) : \tau$.

In the standard approach to operational semantics, the interpretation of rules is expressed via an *iterator* that is parameterized by a base set T of semantic triples in which to look for subderivations, and by a mapping Σ from symbolic variables to values and subterms. In the following, we write $\Sigma\,(t)$ for $\Sigma\,(x_t)$ if $t = x_t$, and for $c\overrightarrow{\Sigma\,(x_{t_n})}$ if $t = c\overrightarrow{x_{t_n}}$.

We restrict the set of semantic triples T under consideration to *well-typed* sets: for every $(\sigma, t, o) \in T$, if $\sigma : \tau_\sigma$, then there exists τ_o such that $(\tau_\sigma, \tau_o) \in \mathcal{R}\,(t)$ and $o : \tau_o$.

We first state when an interpretation is *defined*, written $[\![L]\!]_T^?\,(\Sigma)\downarrow$, assuming that $\mathtt{wf}\,(dom\,(\Sigma))\,(L)$. Intuitively, it is defined if every predicate and function it uses are themselves defined, for which we use the same notation. For instance, an addition function is only defined if both its arguments are numbers. An undefined interpretation is written $[\![L]\!]_T^?\,(\Sigma)\uparrow$.

$$
\begin{aligned}
[\![D(x_1, t, x_2) :: L]\!]_T^?\,(\Sigma)\downarrow &\iff \forall r.\ \left(\left(\Sigma(x_1), t, r\right) \in T \implies [\![L]\!]_T^?\,(\Sigma + x_2 \mapsto r)\downarrow\right)\\
[\![P(\overrightarrow{x_n}) :: L]\!]_T^?\,(\Sigma)\downarrow &\iff \mathtt{I}\,(P)\,(\overrightarrow{\Sigma(x_n)})\downarrow \wedge \left(\mathtt{I}\,(P)\left(\overrightarrow{\Sigma\,(x_n)}\right) \implies [\![L]\!]_T^?\,(\Sigma)\downarrow\right)\\
[\![f(\overrightarrow{x_n}) = x :: L]\!]_T^?\,(\Sigma)\downarrow &\iff \mathtt{I}\,(f)\,(\overrightarrow{\Sigma(x_n)})\downarrow \wedge [\![L]\!]_T^?\left(\Sigma + x \mapsto \mathtt{I}\,(f)\,(\overrightarrow{\Sigma(x_n)})\right)\downarrow\\
[\![[]]\!]_T^?\,(\Sigma)\downarrow &\iff \top
\end{aligned}
$$

The following lemma states that the interpretation of well-typed rules is defined when given well-typed semantic contexts.

Lemma 2.5. *Let t a closed term such that $t = c\overrightarrow{t_n}$, where c has signature $\overrightarrow{s_n} \to s_c$ and for every i, t_i has sort s_i. Let $Rule\left(c\overrightarrow{x_{t_n}}\right)$ a well-typed rule, T a well-typed set, σ a state, and τ_σ a type. Let $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$. If $\sigma : \tau_\sigma$ and $(\tau_\sigma, \_) \in \mathcal{R}(t)$, then $\left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_T^? (\Sigma)\downarrow$.*

Proof. See supplementary material. □

2.4 Concrete Interpretation of Semantic Rules

We now define the concrete interpretation of a rule L given a set of semantic triples T and mapping Σ, assuming both $\mathtt{wf}\,(dom\,(\Sigma))\,(L)$ and $[\![L]\!]_T\,(\Sigma)\downarrow$. We assume given an interpretation $\mathtt{C}\,(\cdot)$ that is consistent. The interpretation returns a set of outputs that may be established by the rule.

$$[\![D(x_1, t, x_2) :: L]\!]_T\,(\Sigma) = \left\{ o \;\middle|\; \exists r. \begin{pmatrix} (\Sigma(x_1), t, r) \in T \\ \wedge\, o \in [\![L]\!]_T\,(\Sigma + x_2 \mapsto r) \end{pmatrix} \right\} \tag{1}$$

$$[\![P(\overrightarrow{x_n}) :: L]\!]_T\,(\Sigma) = \begin{cases} [\![L]\!]_T\,(\Sigma) & \text{if } \mathtt{C}\,(P)\,(\overrightarrow{\Sigma(x_n)}) \\ \emptyset & \text{otherwise} \end{cases} \tag{2}$$

$$[\![f(\overrightarrow{x_n}) = x :: L]\!]_T\,(\Sigma) = [\![L]\!]_T\left(\Sigma + x \mapsto \mathtt{C}\,(f)\,(\overrightarrow{\Sigma(x_n)})\right)$$

$$[\![[]]\!]_T\,(\Sigma) = \{\Sigma(x_o)\}$$

Note that we distinguish between $[\![L]\!]_T\,(\Sigma)\uparrow$ and $[\![L]\!]_T\,(\Sigma) = \emptyset$. For instance, in case (1), the interpretation $[\![D(x_1, t, x_2) :: L]\!]_T\,(\Sigma)$ is undefined if for some r, the interpretation of L is undefined. If the interpretation is defined, it is empty if for every r the interpretation is empty. In particular, when T is empty, the interpretation of $[\![D(x_1, t, x_2) :: L]\!]_\emptyset\,(\Sigma)$ is always defined and empty. An interpretation is empty in the predicate case (2) when the predicate is defined but does not hold.

In the following, when we write $o \in [\![L]\!]_T\,(\Sigma)$, we implicitly assume that $[\![L]\!]_T\,(\Sigma)\downarrow$.

Lemma 2.6. *Let t a closed term such that $t = c\overrightarrow{t_n}$, where c has signature $\overrightarrow{s_n} \to s_c$ and for every i, t_i has sort s_i. Let $Rule\left(c\overrightarrow{x_{t_n}}\right)$ a well-typed rule, T a well-typed set, σ a state, and τ_σ a type. Let $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$. If $\sigma : \tau_\sigma$ and $(\tau_\sigma, \_) \in \mathcal{R}(t)$, then for any $o \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_T (\Sigma)$, there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $o : \tau_o$.*

Proof. See supplementary material. □

The semantics of concrete rules, written $\Downarrow$, is defined using the functional F, called *immediate consequence* and detailed below, where T is well-typed. Intuitively, the functional F picks a rule and applies it, using the hypotheses in T whenever the rule requires a semantic fact $D(x_1, t, x_2)$. The semantics of concrete rules $\Downarrow$ is then defined as the smallest fixed point of F. This can be seen as an inductive interpretation of the semantic rules.

$$F(T) = \left\{ (\sigma, t, o) \;\middle|\; \begin{array}{l} t = c\overrightarrow{t_n} \text{ closed} \wedge (\tau_\sigma, \_) \in \mathcal{R}(t) \wedge \sigma : \tau_\sigma \wedge \\ \exists Rule\left(c\overrightarrow{x_{t_n}}\right). \\ \quad o \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_T \left(x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}\right) \end{array} \right\} \tag{3}$$

Lemma 2.7. *If T is well-typed, then $F(T)$ is well-typed.*

Proof. See supplementary material. □

Lemma 2.8. *The functional F is monotonic on well-typed triple sets.*

Proof. See supplementary material. □

An alternative definition of $\Downarrow$ is shown below: as the functional F is monotonic on well-typed triple sets, it preserves such sets, and the empty set is well-typed. Thus the smallest fixed point of F is exactly the union of all its iteration [Tarski 1955].

$$\Downarrow = \bigcup_{k=0}^{\infty} F^k(\emptyset) \tag{4}$$

In the following we write $\sigma, t \Downarrow^k o$ for $(\sigma, t, o) \in F^k(\emptyset)$. Intuitively, $\sigma, t \Downarrow^k o$ means that the semantic triple $\sigma, t \Downarrow o$ can be derived by a derivation tree of depth less than k.

Lemma 2.9. *The triple sets $\Downarrow^k$ are well-typed for any k.*

Proof. This is a direct induction on k, relying on the fact that the empty set is well-typed. □

Corollary 2.10. *The triple set $\Downarrow$ is well-typed.*

3 ABSTRACT INTERPRETATION OF SEMANTIC RULES

This section describes how a set of semantic rules defining a programming language can be re-interpreted over an abstract domain of properties to obtain an abstract interpretation of the language. In Schmidt's original work on abstract interpretation of natural (big-step) semantics, the basic idea is to define *abstract derivations* over the abstract domains of program properties. The correctness of an analysis is then expressed through a relation between concrete and abstract derivations, induced by the relation between concrete and abstract domains. In this framework, abstract rules can be added freely, and correctness amounts to prove that abstract derivations cover all the corresponding concrete derivations. For example, in the case of *if*, it is common to have two concrete rules and one abstract rule that covers both:

$$\textsc{IfTrue}\ \frac{S, e \Downarrow \mathtt{tt} \qquad S, s_1 \Downarrow r}{S, \mathit{if}\, e\, s_1\, s_2 \Downarrow r} \qquad \textsc{IfFalse}\ \frac{S, e \Downarrow \mathtt{ff} \qquad S, s_2 \Downarrow r}{S, \mathit{if}\, e\, s_1\, s_2 \Downarrow r} \qquad \textsc{IfAbstract}\ \frac{S^\sharp, s_1 \Downarrow^\sharp r_1 \qquad S^\sharp, s_2 \Downarrow^\sharp r_2}{S, \mathit{if}\, e\, s_1\, s_2 \Downarrow^\sharp r_1 \sqcup r_2}$$

Bodin *et al.* [Bodin et al. 2015] formalized this approach in Coq in a restricted setting (pretty-big-step) where abstract rules are obtained systematically from concrete rules, by mechanically applying abstraction functions to each rule component.

In this work, we take a different approach in which the abstract semantic derivations are built using the same inference rules as the concrete semantics, but using abstract versions of basic predicates $\mathtt{A}(P)$ and functions $\mathtt{A}(f)$. The key difference between the two interpretations lies in *how* the inference rules are used. Whereas the concrete semantics applies one concrete rule at a time and collects the set of resulting states, the abstract semantics applies all abstract rules that apply to a given (abstract) state, and determines a common abstract result that approximates all the resulting abstract states. We show in Section 6.2 how more common abstract rules such as Rule IfAbstract above can be proven admissible in our formalism.

We start by defining the notion of abstract domain, and correctness of abstract functions and predicates.

Definition 3.1 (Abstract domain). An abstract domain for the concrete domain *Val* is a pre-order $(A, \leq)$ and a concretisation function $\gamma : A \to \mathcal{P}(\mathit{Val})$ such that γ is monotone:

$$\forall v_1^\sharp, v_2^\sharp.\ v_1^\sharp \leq v_2^\sharp \implies \gamma(v_1^\sharp) \subseteq \gamma(v_2^\sharp)$$

and such that they are compatible with typing

$$\forall v^\sharp, v.\ v \in \gamma\left(v^\sharp\right) \wedge v : \tau \implies v^\sharp : \tau;$$
$$\forall v_1^\sharp, v_2^\sharp.\ v_1^\sharp \leq v_2^\sharp \wedge v_1^\sharp : \tau \implies v_2^\sharp : \tau$$

We do not define abstract terms, but we extend γ such that $\gamma(t) = \{t\}$ for every term t.

In order to define an abstract interpretation we must specify an abstract domain for each of the types used to define the semantic rules.

Example 3.2 (Abstraction of the example semantics using intervals). In the running example from Figure 1, we had the following types in the semantic definition (cf. Figure 3)

$$int, bool, state, state(\mathsf{x}), ident(\mathsf{x})$$

where x is a variable.

Writing $[n, m]$ for the interval of integers between n and m (with the convention that $[n, m] = \emptyset$ if $m < n$), we can define the abstract domains for each of the types as follows:

$$\begin{aligned}
int^\sharp &= ([n, m] : n \in \mathbf{Z} \cup \{-\infty\} \wedge m \in \mathbf{Z} \cup \{+\infty\}) \\
bool^\sharp &= \{\bot_{bool}, true, false, \top_{bool}\} \\
Val^\sharp &= int^\sharp \times bool^\sharp \\
state^\sharp, state(\mathsf{x})^\sharp &= \text{partial functions from variables to } Val^\sharp \\
ident(\mathsf{x})^\sharp &= ident(\mathsf{x})
\end{aligned}$$

Note that these are *domains*, and that types are shared between every interpretation. Hence the value $([1, 4], \top_{bool})$ has both types *int* and *bool*. Note also that there is no abstract object that is both a value and a state. A state $H^\sharp$ may have type $state(\mathsf{x})$ only if $\mathsf{x} \in dom\left(H^\sharp\right)$.

We define γ as follows

$$\begin{aligned}
\gamma([n, m]) &= \{i \mid n \leq i \leq m\} & \gamma(\bot_{bool}) &= \emptyset \\
\gamma(true) &= \{\mathtt{tt}\} & \gamma(false) &= \{\mathtt{ff}\} \\
\gamma(\top_{bool}) &= \{\mathtt{tt}, \mathtt{ff}\} & \gamma(i, b) &= \gamma(i) \cup \gamma(b)
\end{aligned}$$

$$\gamma\left(H^\sharp\right) = \left\{ H \;\middle|\; \begin{array}{c} dom(H) = dom\left(H^\sharp\right) \wedge \\ \forall \mathsf{x} \in dom\left(H^\sharp\right).\ H(\mathsf{x}) \in \gamma\left(H^\sharp(\mathsf{x})\right) \end{array} \right\}$$

We have $[n, m] \leq [n', m']$ if $n' \leq n$ and $m \leq m'$. We have $b_1^\sharp \leq b_2^\sharp$ if $\gamma(b_1) \subseteq \gamma(b_2)$. We have $\gamma(i, b) \leq \gamma(i', b')$ if $i \leq i'$ and $b \leq b'$. We have $H_1^\sharp \leq H_2^\sharp$ if $dom\left(H_1^\sharp\right) = dom\left(H_2^\sharp\right)$ and for every $\mathsf{x} \in dom\left(H_1^\sharp\right)$, $H_1^\sharp(\mathsf{x}) \leq H_2^\sharp(\mathsf{x})$.

In the following, we write $\overrightarrow{v_n} \in \gamma\left(\overrightarrow{v_n^\sharp}\right)$ iff $\forall i \in [1..n],\ v_i \in \gamma\left(v_i^\sharp\right)$.

Definition 3.3 (Correct abstraction of predicates). The abstract predicate $\mathsf{A}(P)$ is a correct abstraction of $\mathsf{C}(P)$ iff

$$\forall \left(\overrightarrow{\tau_n}, \overrightarrow{\tau'_n}\right) \in \mathcal{R}(P).\ \forall \overrightarrow{v_n^\sharp : \tau_n}.\ \forall \overrightarrow{v_n} \in \gamma\left(\overrightarrow{v_n^\sharp}\right).\ \overrightarrow{v_n : \tau_n} \wedge \mathsf{C}(P)\left(\overrightarrow{v_n}\right) \implies \mathsf{A}(P)\left(\overrightarrow{v_n^\sharp}\right)$$

Definition 3.4 (Correct abstraction of functions). The abstract function $\mathtt{A}(f)$ is a correct abstraction of $\mathtt{C}(f)$ iff

$$\forall \left(\overrightarrow{\tau_n}, \tau\right) \in \mathcal{R}(f) \,.\, \forall \overrightarrow{v_n^\sharp : \tau_n} .\, \forall \overrightarrow{v_n} \in \gamma\left(\overrightarrow{v_n^\sharp}\right) .\, \overrightarrow{v_n : \tau_n} \implies \mathtt{C}(f)\left(\overrightarrow{v_n}\right) \in \gamma\left(\mathtt{A}(f)\left(\overrightarrow{v_n^\sharp}\right)\right)$$

Example 3.5 (Abstract predicates and functions for the example semantics).

$$\begin{aligned}
\mathtt{A}(\mathtt{isInt}) &= \lambda(i,b).\ i \neq \emptyset \\
\mathtt{A}(\mathtt{toInt}) &= \lambda(i,b).\ i \\
\mathtt{A}(\mathtt{isInDom}) &= \lambda\left(H^\sharp, \mathtt{x}\right) .\, \mathtt{x} \in dom\left(H^\sharp\right) \\
\mathtt{A}(\mathtt{read}) &= \lambda\left(H^\sharp, \mathtt{x}\right) .\, H^\sharp(\mathtt{x}) \\
\mathtt{A}(\mathtt{add}) &= \lambda(i_1, i_2).\ [l_1 + l_2, u_1 + u_2] \text{ if } i_1 = [l_1, u_1] \text{ and } i_2 = [l_2, u_2] \\
\mathtt{A}(\mathtt{eq}) &= \lambda(i_1, i_2).\ \begin{cases} true & \text{if } i_1 = [n,n] = i_2 \\ false & \text{if } i_1 \cap i_2 = \emptyset \\ \top_{bool} & \text{otherwise} \end{cases}
\end{aligned}$$

We now define the abstract interpretation of a rule L given a set of semantic triples T and mapping Σ, assuming both $\mathtt{wf}(dom(\Sigma))(L)$ and $[\![L]\!]^\sharp_T(\Sigma)\downarrow$. The interpretation returns a set of abstract outputs that may be established by the rule. This interpretation is parameterised by an additional parameter O that includes every well-typed abstract output of a term. Intuitively, the interpretation of the rule is a set of semantic triples that do not contradict the rule. In particular, if the rule does not apply, the rule should not rule out any semantic triple: the interpretation of such a rule should be all possible well-typed triples. The set O reflects this, as illustrated by the predicate case below.

Definition 3.6 (Abstract interpretation of a rule).

$$\begin{aligned}
[\![D(x_1, t, x_2) :: L]\!]^\sharp_{T,O}(\Sigma) &= \left\{ o \,\middle|\, \exists r^\sharp . \left(\begin{array}{c} (\Sigma(x_1), t, r^\sharp) \in T \\ \wedge\ o \in [\![L]\!]^\sharp_{T,O}\left(\Sigma + x_2 \mapsto r^\sharp\right) \end{array} \right) \right\} \\
[\![P(\overrightarrow{x_n}) :: L]\!]^\sharp_{T,O}(\Sigma) &= \begin{cases} [\![L]\!]^\sharp_{T,O}(\Sigma) & \text{if } \mathtt{A}(P)\left(\overrightarrow{\Sigma(x_n)}\right) \\ O & \text{otherwise} \end{cases} \\
[\![f(\overrightarrow{x_n}) = x :: L]\!]^\sharp_{T,O}(\Sigma) &= [\![L]\!]^\sharp_{T,O}\left(\Sigma + x \mapsto \mathtt{A}(f)\left(\overrightarrow{\Sigma(x_n)}\right)\right) \\
[\![[]]\!]^\sharp_{T,O}(\Sigma) &= \{\Sigma(x_o)\}
\end{aligned}$$

Definition 3.7. Let $\sigma^\sharp$ a value and t a term, the set of possible outputs is defined as $out\left(\sigma^\sharp, t\right) = \left\{ o^\sharp \,\middle|\, \forall \tau_\sigma .\, \sigma^\sharp : \tau_\sigma \implies \exists \tau_o .\, (\tau_\sigma, \tau_o) \in \mathcal{R}(t) \wedge o^\sharp : \tau_o \right\}$.

The following lemma states that the result of an abstract interpretation of a rule is well-typed.

Lemma 3.8. *Let t a closed term such that $t = c\overrightarrow{t_n}$, where c has signature $\overrightarrow{s_n} \to s_c$ and for every i, t_i has sort s_i. Let $Rule\left(c\overrightarrow{x_{t_n}}\right)$ a well-typed rule, T a well-typed set, $\sigma^\sharp$ an abstract state. Let $\Sigma = x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}$. If for every τ_σ such that $\sigma^\sharp : \tau_\sigma$ we have $(\tau_\sigma, \_) \in \mathcal{R}(t)$, then for any $o^\sharp \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]^\sharp_{T, out\left(\sigma^\sharp, t\right)}(\Sigma)$, there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$.*

Proof. See supplementary material. □

Finally, the abstract interpretation of a set of rules, written $\Downarrow^\sharp$, is defined as the largest fixed point of the functional induced by the abstract interpretation of the semantic rules.

Definition 3.9 (Abstract interpretation of a big-step semantics). Let T be well-typed, and

$$F^\sharp(T) = \left\{ \left(\sigma^\sharp, t, o^\sharp\right) \;\middle|\; \begin{array}{l} t = c\overrightarrow{t_n} \text{ closed} \wedge \\ \left(\forall \tau_\sigma.\ \sigma^\sharp : \tau_\sigma \implies (\tau_\sigma, \_) \in \mathcal{R}(t)\right) \wedge \\ \forall Rule\left(c\overrightarrow{x_{t_n}}\right). \\ \quad o^\sharp \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right) \right]\!\right]^\sharp_{T, out\left(\sigma^\sharp, t\right)} \left(x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}\right) \end{array} \right\} \tag{5}$$

Then

$$\Downarrow^\sharp = \bigcup_{T \subseteq F^\sharp(T)} T \tag{6}$$

The functional $F^\sharp$ applies *all* the rules for a given term, as opposed to F which only applies one. We also consider the largest fixed point instead of the smallest fixed point. This is equivalent to consider the abstract semantics co-inductively. An example of how to use the abstract immediate consequence in practise is shown in Section 6.2.

Lemma 3.10. *If T is well-typed, then $F^\sharp(T)$ is well-typed.*

Proof. See supplementary material. □

Lemma 3.11. *The functional $F^\sharp$ is monotonic on well-typed triple sets.*

Proof. See supplementary material. □

The formalisation highlights the differences between the two interpretations. First, the concrete immediate consequence operator is defined using an existential quantifier (one rule is enough to justify a semantic triple), whereas its abstract counterpart uses a universal quantifier to state that an abstract triple must satisfy all rules for a given term in order to be justified. Second, there is a difference in the way that predicates in the list of hypotheses are interpreted. In the concrete interpretation, a predicate that does not hold means that no result can be produced with the given input state, hence the result is the empty set. In the abstract interpretation, a predicate that does not apply *should not constrain* the set of abstract triples deducible for a given term. In that case, the interpretation returns every term that may be well-typed starting from the input type.

From now on, we only consider sets of abstract triples that are well-typed.

3.1 Correctness of Abstract Semantics

In Sections 2.4 and 3, we have defined two sets $\Downarrow$ and $\Downarrow^\sharp$. The goal of this framework is to make sure that $\Downarrow^\sharp$ is correct with respect to $\Downarrow$. The correctness of an abstract interpretation amounts to prove that every deduction leads to a triple $(\sigma^\sharp, t, o^\sharp)$ that is a valid statement about the concrete semantics. To make this precise, we define the notion of k *correctness* of a set of abstract triples. It states that every triple correctly captures all matching concrete derivation trees of depth less than k. In particular, a set T is correct if and only if it is k correct for all k.

Definition 3.12. A set T of abstract triples is k *correct* iff for every $(\sigma^\sharp, t, o^\sharp) \in T$, if $\sigma, t \Downarrow^k o$ and $\sigma \in \gamma\left(\sigma^\sharp\right)$, then $o \in \gamma\left(o^\sharp\right)$. A set T of abstract triples is *correct* iff it is k correct for all k.

A set T of abstract triples is thus correct iff for every $(\sigma^\sharp, t, o^\sharp) \in T$, if $\sigma, t \Downarrow o$ and $\sigma \in \gamma\left(\sigma^\sharp\right)$, then $o \in \gamma\left(o^\sharp\right)$. Note that the definition of the k correctness of a set is stable over set unions: if T_1 and T_2 are k correct, then so is $T_1 \cup T_2$.

Proving that a full abstract semantics $\Downarrow^\sharp$ is correct can be long and difficult if the underlying programming language is not principled. Our framework aims at providing a local criterium to ensure the global soundness of the abstract semantics. We thus define the following criterium for individual rules.

Definition 3.13. Let t a closed term such that $t = c\overrightarrow{t_n}$, $\sigma^\sharp$ such that for every τ_σ, if $\sigma^\sharp : \tau_\sigma$, then $(\tau_\sigma, \_) \in \mathcal{R}(t)$, τ_σ one such type, i.e., $(\tau_\sigma, \_) \in \mathcal{R}(t)$, σ such that $\sigma : \tau_\sigma$. Let $\Sigma^\sharp = x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}$ and $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$. A rule $Rule\left(c\overrightarrow{x_{t_n}}\right)$ is *step-correct* iff, assuming a set T of abstract triples that is k correct, then $o^\sharp \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T, out\left(\sigma^\sharp, t\right)}\left(\Sigma^\sharp\right)$, $o \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_{\Downarrow^k}(\Sigma)$, and $\sigma \in \gamma\left(\sigma^\sharp\right)$ implies $o \in \gamma\left(o^\sharp\right)$.

Intuitively, if a set of triple $F^\sharp(T)$ is generated from a k correct set T and from the immediate consequence $F^\sharp$ of a semantics whose rules are all step-correct, then this set of triples $F^\sharp(T)$ is $k+1$ correct. The following lemma states exactly this intuition.

Lemma 3.14. *Let T be a k correct set of abstract triples. If every rule is step-correct, then $F^\sharp(T)$ is $k+1$ correct.*

Proof. Unfolding definitions, we assume a k correct set T such that $(\sigma^\sharp, t, o^\sharp) \in F^\sharp(T)$, that $\sigma, t \Downarrow^{k+1} o$, and that $\sigma \in \gamma\left(\sigma^\sharp\right)$. We need to prove that $o \in \gamma\left(o^\sharp\right)$. By definition we have $(\sigma, t, o) \in F\left(\Downarrow^k\right)$, hence there exists a rule $Rule\left(c\overrightarrow{x_{t_n}}\right)$ such that $t = c\overrightarrow{t_n}$, $\sigma : \tau_\sigma$ where $(\tau_\sigma, \_) \in \mathcal{R}(t)$, and $o \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_{\Downarrow^k}\left(x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}\right)$. Since $\sigma \in \gamma\left(\sigma^\sharp\right)$, we have $\sigma^\sharp : \tau_\sigma$. Since we have $(\sigma^\sharp, t, o^\sharp) \in F^\sharp(T)$, we have for all rules and in particular for $Rule\left(c\overrightarrow{x_{t_n}}\right)$ that $\sigma^\sharp$ is such that for every τ'_σ where $\sigma^\sharp : \tau'_\sigma$, then $(\tau'_\sigma, \_) \in \mathcal{R}(t)$, and $o^\sharp \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T, out\left(\sigma^\sharp, t\right)}\left(x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}\right)$. As $Rule\left(c\overrightarrow{x_{t_n}}\right)$ is step-correct, we conclude that $o \in \gamma\left(o^\sharp\right)$, as required. □

We prove and leverage this observation in the following theorem.

Theorem 3.15. *Let T be a set of abstract triples such that $T \subseteq F^\sharp(T)$. If every rule is step-correct, then T is correct.*

Proof. We first show that T is k correct for every k by induction on k. The base case is trivial as there is no empty concrete derivation: any set is 0 correct. For the $k+1$ case, we assume that T is k correct. By Lemma 3.14, we know that $F^\sharp(T)$ is $k+1$ correct. By inclusion $T \subseteq F^\sharp(T)$ is also $k+1$ correct.

Now assume we have $\sigma, t \Downarrow o$, $(\sigma^\sharp, t, o^\sharp) \in T$, and $\sigma \in \gamma\left(\sigma^\sharp\right)$. Let k be the depth of the concrete derivation, that is such that $\sigma, t \Downarrow^k o$—such a k exists by Equality 4 (page 9). As T is k correct, then we have $o \in \gamma\left(o^\sharp\right)$. □

Theorem 3.15 provides a method to prove that an abstract semantics $\Downarrow^\sharp$ is correct. This method is local to the level of rules, but some rules can still be large and complex. The following theorem pushes this locality to the level of the predicates and functions in a rule: to prove an abstract semantics correct, we now only have to prove the local correctness of its rule components, as given by Definitions 3.3 and 3.4.

Theorem 3.16. *If all abstract predicates and functions in a well-typed rule Rule* $\left(c\overrightarrow{x_{t_n}}\right)$ *are correct, then the rule Rule* $\left(c\overrightarrow{x_{t_n}}\right)$ *is step-correct.*

Proof. See supplementary material. □

4 WEAKENING AND TRACE-PARTITIONING

The abstraction of the semantic rules in the previous section was done by replacing the concrete interpretation of basic functions and predicates with an abstract counterpart. This produces a set of abstract rules that is in one-to-one correspondence with the set of concrete rules defining the semantics. This set of abstract rules defines a logic which can be used for reasoning about programs in a syntax-directed manner. It is common to extend such a logic by adding extra rules which are used to construct new derivations from existing derivations. We consider two rules: a rule for *weakening* the result of an analysis, and a rule that corresponds to the principle of *trace partitioning* [Mauborgne and Rival 2005; Rival and Mauborgne 2007] which permits the analysis to split the property about the starting state into several sub-properties that are then analysed separately.

$$\textsc{weaken}\ \frac{\sigma_1^\sharp \leq \sigma_2^\sharp \qquad \sigma_2^\sharp, t \Downarrow o_1^\sharp \qquad o_1^\sharp \leq o_2^\sharp}{\sigma_1^\sharp, t \Downarrow o_2^\sharp}$$

$$\textsc{trace-partitioning}\ \frac{\sigma_1^\sharp, t \Downarrow o^\sharp \quad \ldots \quad \sigma_n^\sharp, t \Downarrow o^\sharp \qquad \sigma^\sharp : \tau \implies \exists i.\ \sigma_i^\sharp : \tau \qquad \gamma\left(\sigma^\sharp\right) \subseteq \gamma\left(\sigma_1^\sharp\right) \cup \ldots \cup \gamma\left(\sigma_n^\sharp\right)}{\sigma^\sharp, t \Downarrow o^\sharp}$$

Weakening and trace partitioning are rules that do not correspond to a syntactic construct of the language and hence are not derived from a corresponding concrete rule. We refer to such rules as *non-structural* rules, as opposed to the structural rules that follow the syntactic structure of the language.

Contrary to structural rules and their formal definitions as in Figure 2, we do not provide a formal definition of non-structural rules. Instead, a non-structural rule is seen as a function computing a new set of semantic triples from an initial set of semantic triples T. For instance, rules weaken and trace-partitioning above are formalised by the following two functions.

$$NS_{\textsc{weaken}}(T) = \left\{\left(\sigma_1^\sharp, t, o_2^\sharp\right) \middle| \sigma_1^\sharp \leq \sigma_2^\sharp \wedge \left(\sigma_2^\sharp, t, o_1^\sharp\right) \in T \wedge o_1^\sharp \leq o_2^\sharp\right\}$$

$$NS_{\textsc{trace-partitioning}}(T) = \left\{\left(\sigma^\sharp, t, o^\sharp\right) \middle| \begin{array}{l} \left(\sigma_1^\sharp, t, o^\sharp\right) \in T \wedge \ldots \wedge \left(\sigma_n^\sharp, t, o^\sharp\right) \in T \\ \wedge\, \forall \tau.\ \sigma^\sharp : \tau \implies \exists i.\ \sigma_i^\sharp : \tau \\ \wedge\, \gamma\left(\sigma^\sharp\right) \subseteq \gamma\left(\sigma_1^\sharp\right) \cup \ldots \cup \gamma\left(\sigma_n^\sharp\right) \end{array}\right\}$$

We suppose non-structural rules to be monotone and to preserve well-typed sets:

$$\forall NS_i, T_1, T_2.\ T_1 \subseteq T_2 \implies NS_i(T_1) \subseteq NS_i(T_2)$$
$$\forall NS_i, T.\ T \text{ well-typed} \implies NS_i(T) \text{ well-typed}$$

This constraint appears when manipulating the non-structural rules, as shown in Section 6.1. Both rules weaken and trace-partitioning are monotone and preserve well-typed sets.

We extend the immediate consequence $F^\sharp$ to take into account the non-structural rules. We enable non-structural rules to be applied in between two rule applications. We limit the application

of the non-structural rules to a finite number of application, as opposed to the infinite depth of derivation trees permitted to computation rule applications. This is crucial for ensuring productivity of the results built from abstract derivation trees.

We suppose given a collection of non-structural rules NS_i and define the iterator $G_j^\sharp(T)$, where T is a set of abstract semantic triples, as below. The set $G_j^\sharp(T)$ is the set of abstract semantic triples computed by one application of an abstract rule and j applications of non-structural rules to the triples from the set T.

$$G_0^\sharp(T) = T$$
$$G_{j+1}^\sharp(T) = \bigcup_{NS_i} NS_i\left(G_j^\sharp(T)\right)$$

Intuitively, $G_j^\sharp(T)$ augments the semantic triples of T by j applications of non-structural rules. The iterator $F^\sharp$ can now be replaced by the following iterator, accounting for non-structural rules before and after $F^\sharp$. The new iterator $G^\sharp$ is monotone by composition of monotone functions.

$$G^\sharp(T) = \bigcup_{j=0}^{\infty} G_j^\sharp\left(F^\sharp\left(\bigcup_{j=0}^{\infty} G_j^\sharp(T)\right)\right) \tag{7}$$

Weakening and trace partitioning enjoy the following *nostep-correctness* property. It is the non-structural equivalent of the step-correctness of Definition 3.13. Contrary to step-correctness, nostep-correctness only requires that the application of the rule to the set T results in a k correct set, and not a $k+1$ correct set.

Definition 4.1. A non-structural rule NS is *nostep-correct* iff for all k, and for all set T of semantic triples, if T is k correct, then $NS(T)$ is also k correct.

Lemma 4.2. *The weakening and trace partitioning rules are nostep-correct.*

Proof. We prove each result separately. Let T be a k correct set of abstract semantic triples. Unfolding definitions, we consider in both cases $(\sigma^\sharp, t, o^\sharp) \in NS(T)$, such that $\sigma, t \Downarrow^k o$ and $\sigma \in \gamma\left(\sigma^\sharp\right)$. We want to prove that $o \in \gamma\left(o^\sharp\right)$.

First consider the case of Rule WEAKEN. As $(\sigma^\sharp, t, o^\sharp) \in NS_{\text{WEAKEN}}(T)$, we know that there exists $\sigma'^\sharp$ and $o'^\sharp$ such that $\sigma^\sharp \leq \sigma'^\sharp$, $o'^\sharp \leq o^\sharp$, and $\left(\sigma'^\sharp, t, o'^\sharp\right) \in T$. As the concretisation γ is monotone by Definition 3.1, we know that $\sigma \in \gamma(\sigma')$. As T is k correct, we have $o \in \gamma\left(o'^\sharp\right)$. We get $o \in \gamma\left(o^\sharp\right)$ by monotony of the concretisation γ.

We now consider the case of Rule TRACE-PARTITIONING. As $(\sigma^\sharp, t, o^\sharp) \in NS_{\text{TRACE-PARTITIONING}}(T)$, we know that there exists $\left(\sigma_1^\sharp, t, o^\sharp\right) \in T, \ldots, \left(\sigma_n^\sharp, t, o^\sharp\right) \in T$ such that $\gamma\left(\sigma^\sharp\right) \subseteq \gamma\left(\sigma_1^\sharp\right) \cup \ldots \cup \gamma\left(\sigma_n^\sharp\right)$. As $\sigma \in \gamma\left(\sigma^\sharp\right)$, there exists an index i such that $\sigma \in \gamma\left(\sigma_i^\sharp\right)$. As $\left(\sigma_i^\sharp, t, o^\sharp\right) \in T$, we get that $o \in \gamma\left(o^\sharp\right)$ by the k correctness of T. □

We extend Theorem 3.15 to account for non-structural rules in the following theorem. This theorem is in particular valid for sets T included in the largest fixed point of $G^\sharp$.

Theorem 4.3. *Let T be a set of abstract semantic triples. If $T \subseteq G^\sharp(T)$, and every abstract rule is step-correct and every non-structural rule is nostep-correct, then T is correct.*

Proof. To prove that T is correct, we prove that it is k correct for all k.

We start by proving the following property for all j, by induction on j.

$$\forall T.\ T \text{ is } k \text{ correct} \implies G^\sharp_j(T) \text{ is } k \text{ correct} \tag{8}$$

The base case is immediate. In the inductive case, let T be a k correct set of abstract semantic triples. The set $G^\sharp_{j+1}(T)$ is a union of sets, which we consider independently. Let thus NS be a non-structural rule. We need to prove the property below, which is exactly the nostep-correctness of NS:

$$G^\sharp_j(T) \text{ is } k \text{ correct} \implies NS\left(G^\sharp_j(T)\right) \text{ is } k \text{ correct}$$

We have now proven Property 8. It follows that the union of all $G^\sharp_j(T)$ is also k correct if T is. We apply Lemma 3.14 to deduce that $F^\sharp\left(\bigcup_{j=0}^{\infty} G^\sharp_j(T)\right)$ is $k+1$ correct. Finally, we apply Property (8) once more to deduce that $G^\sharp(T)$ is $k+1$ correct when T is k correct.

We are left to prove that T is k correct for all k. We prove this by induction over k.

The base case is trivial, any set being 0 correct, as $\Downarrow^0$ is an empty relation.

In the inductive case, we know by induction that T is k correct and want to prove it $k+1$ correct. We know that $T \subseteq G^\sharp(T)$: we conclude by application of the above corollary of Property 8. □

5 THE FRAME RULE

So far, we have shown how the framework of typed big-step operational semantics can be used to formalise abstract interpretations in a way that guarantees the global correctness of a program logic, once the correctness of the basic (local) functions and predicates has been proved. In this section we show that the framework also leads to a solution to the question of how to combine standard abstract interpretation with separation logic [Ishtiaq and O'Hearn 2001; O'Hearn et al. 2001; Reynolds 2002, 2008]. More precisely, we shall show how the *separating conjunction* $\star$ and the accompanying frame rule from separation logic

$$\textsc{Frame}\ \frac{\phi, t \Downarrow \phi'}{\phi \star \phi_c, t \Downarrow \phi' \star \phi_c}$$

can be added to the program logics obtained by abstract interpretations. This rule adds another dimension of locality to the framework. Informally, it extends an analysis of a local part of the memory to a result about the entire state of the computation, *provided* that this local part is well separated from the rest. In terms of big-step semantics, the frame rule is particular because it essentially enables us to produce a new derivation from an existing one.

The frame rule is different from the non-structural rules of Section 4. Indeed, the weakening and trace-partitioning rules basically use properties of concrete derivation trees to infer more abstract semantic triples. In particular, we have $T \subseteq NS(T)$ for these two rules. The frame rule, however, produces a set disjoint from the initial set, as they do not mention the same resources. In our approach, we use types to model the resources of separation logic. The frame rule is thus able to change the types of semantic triples. To capture this, we introduce the notion of type transformers.

We consider a family of partial type transformers, written $\odot_{\tau_v}$. We write $\odot_{\tau_v}(\tau) \downarrow$ if $\odot_{\tau_v}$ is defined for type τ. In addition, types of terms, functions, and predicates must be consistent with these type transformers.

Definition 5.1 (Constraints on semantic rules about type transformers). For every term t, for every $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$, if $\odot_{\tau_v}(\tau_\sigma) \downarrow$, then $\odot_{\tau_v}(\tau_o) \downarrow$ and $(\odot_{\tau_v}(\tau_\sigma), \odot_{\tau_v}(\tau_o)) \in \mathcal{R}(t)$. Conversely, if $(\odot_{\tau_v}(\tau_\sigma), \tau'_o) \in \mathcal{R}(t)$, then there exists τ_o such that $\tau'_o = \odot_{\tau_v}(\tau_o)$.

We also restrict that, for every predicate P, for every $\left(\overrightarrow{\tau_n}, \overrightarrow{\tau'_n}\right) \in \mathcal{R}(P)$, if $\overrightarrow{\odot_{\tau_v}(\tau_n)\downarrow}$, then $\overrightarrow{\odot_{\tau_v}(\tau'_n)\downarrow}$ and $\left(\overrightarrow{\odot_{\tau_v}(\tau_n)}, \overrightarrow{\odot_{\tau_v}(\tau'_n)}\right) \in \mathcal{R}(P)$.

Finally, for every function f, for every $\left(\overrightarrow{\tau_n}, \tau\right) \in \mathcal{R}(f)$, if $\overrightarrow{\odot_{\tau_v}(\tau_n)\downarrow}$, we then have $\odot_{\tau_v}(\tau)\downarrow$ and $\left(\overrightarrow{\odot_{\tau_v}(\tau_n)}, \odot_{\tau_v}(\tau)\right) \in \mathcal{R}(f)$.

In the following, we write $\odot_{\tau_v}(\Gamma)$ for the mapping $\{x \mapsto \odot_{\tau_v}(\Gamma(x)) \mid x \in dom(\Gamma)\}$, and $\odot_{\tau_v}\left(\overline{\Gamma}\right)$ for the set $\left\{\odot_{\tau_v}(\Gamma) \,\middle|\, \Gamma \in \overline{\Gamma}\right\}$. These are undefined as soon as one application of $\odot_{\tau_v}$ is undefined.

On the side of interpretations, we consider a family of partial value transformers, indexed by values, and written $\oplus_v$. Each value transformer is associated to a type transformer, written $\oplus_v : \odot_{\tau_v}$. Application of a value transformer $\oplus_v$ to a value w is written in a postfix notation: $w\oplus_v$. Intuitively, a type transformer $\odot_{\tau_v}(\tau)$ transforms the resources associated to the type τ to add the resources associated to the type of the value v; and a value transformer effectively performs the addition of the resources in the concrete and abstract values. For instance, a typical concrete value transformer is the disjoint union $\uplus$ over states H; and a typical abstract value transformer is the separating conjunction $\star$ between formulae.

Value transformers must be consistent with their type: if $\oplus_v : \odot_{\tau_v}$, then $v : \tau_v$, and for all τ and $v' : \tau$, if $\odot_{\tau_v}(\tau)\downarrow$, then $v'\oplus_v \downarrow$ and $v'\oplus_v : \odot_{\tau_v}(\tau)$. In addition, if $v'' : \odot_{\tau_v}(\tau)$, then there exist v' and $\oplus_v$ such that $\oplus_v : \odot_{\tau_v}$, $v'' = v'\oplus_v$, and $v' : \tau$. Conversely, if $v'\oplus_v : \tau'$ and $\oplus_v : \odot_{\tau_v}$, then there exists τ such that $\tau' = \odot_{\tau_v}(\tau)$, and $v' : \tau$.

Let $\oplus_v : \odot_{\tau_v}$. We assume that for every term t of sort s, we have $\odot_{\tau_v}(s)\downarrow$, $\odot_{\tau_v}(s) = s$, and $t\oplus_v = t$.

Definition 5.2. An interpretation of a predicate is *framing consistent* iff

$$\forall \left(\overrightarrow{\tau_n}, \overrightarrow{\tau'_n}\right) \in \mathcal{R}(P), \overrightarrow{v_n : \tau_n}, \oplus_v : \odot_{\tau_v}. \overrightarrow{\odot_{\tau_v}(\tau_n)\downarrow} \implies \mathtt{I}(P)\left(\overrightarrow{v_n}\right) \iff \mathtt{I}(P)\left(\overrightarrow{v_n\oplus_v}\right) \tag{9}$$

An interpretation of a function is *framing consistent* iff

$$\forall \left(\overrightarrow{\tau_n}, \tau\right), \overrightarrow{v_n : \tau_n}, \oplus_v : \odot_{\tau_v}. \overrightarrow{\odot_{\tau_v}(\tau_n)\downarrow} \implies \mathtt{I}(f)\left(\overrightarrow{v_n}\right)\oplus_v = \mathtt{I}(f)\left(\overrightarrow{v_n\oplus_v}\right) \tag{10}$$

Note that in the previous definition, there is no need to check that the application of $\oplus_v$ is defined, since it is implied by the application of its type being defined.

Example 5.3. To give a concrete interpretation of our ongoing example, we first need to refine types to reflect resources. As in Bornat *et al.* [Bornat et al. 2006], the resources of our example are variables. Types for heaps are now of the form $state(R)$. Intuitively, a heap has type $state(R)$ if it is restricted to variables in R. The refined types are given in Figure 4.

A heap H_R is now parameterized by a set of variables R. A heap $H_R : state(R)$ is a partial function from R to values, we write $dom(H_R)$ for where it is defined. Values are identical to Example 2.4. Most functions and predicates are also identical, with three exceptions. The functions $\mathtt{read}(H_R, \mathtt{x})$ and $\mathtt{write}(H_R, \mathtt{x}, v)$ are undefined if $\mathtt{x} \notin R$. The predicate $\mathtt{isInDom}(H_R, \mathtt{x})$ is also undefined if $\mathtt{x} \notin R$.

Type transformers are defined for types $state(R)$, and we have $\odot_{state(R)}(\tau)\downarrow$ when $\tau = bool$, $\tau = int$, or $\tau = state(R')$ or $\tau = state(\mathtt{x}, R')$ with $R \cap R' = \emptyset$. We then have $\odot_{state(R)}(int) = int$, $\odot_{state(R)}(bool) = bool$, $\odot_{state(R)}(state(R')) = state(R \cup R')$, and $\odot_{state(R)}(state(\mathtt{x}, R')) = state(\mathtt{x}, R \cup R')$. As relations in Figure 4 are all upward closed (one can always replace an R with a bigger R'), the constraints of Definition 5.1 are satisfied.

Value transformers are defined for heaps, and $\oplus_{H_R}$ is written $\uplus H_R$. We have $\uplus H_R : \odot_{state(R)}$. Assuming that $\odot_{state(R)}\left(H'_{R'}\right)\downarrow$, we have $H'_{R'} \uplus H_R = H''_{R\cup R'}$ be the function from $dom(H_R) \cup$

s, t, P, or f	$\mathcal{R}(s)$, $\mathcal{R}(t)$, $\mathcal{R}(P)$, or $\mathcal{R}(f)$
expr	$\bigcup_R \{(state(R), int), (state(R), bool)\}$
stat	$\bigcup_R \{(state(R), state(R))\}$
$const(n)$	$\bigcup_R \{(state(R), int) \mid \}$
$var(\mathsf{x})$	$\bigcup_R \{(state(R), int), (state(R), bool) \mid \mathsf{x} \in R\}$
$e_1 + e_2$	$\bigcup_R \{(state(R), int)\} \cap \mathcal{R}(e_1) \cap \mathcal{R}(e_2)$
$e_1 = e_2$	$\bigcup_R \{(state(R), bool)\} \cap \mathcal{R}(e_1) \cap \mathcal{R}(e_2)$
$\mathsf{x} := e$	$\bigcup_R \{(state(R), state(R)) \mid \mathsf{x} \in R\}$
$s_1; s_2$	$\bigcup_R \{(state(R), state(R))\} \cap \mathcal{R}(s_1) \cap \mathcal{R}(s_2)$
$if\, e\, s_1\, s_2$	$\bigcup_R \{(state(R), state(R)) \mid (state(R), \_) \in \mathcal{R}(e)\} \cap \mathcal{R}(s_1) \cap \mathcal{R}(s_2)$
$while\, e\, s$	$\bigcup_R \{(state(R), state(R)) \mid (state(R), \_) \in \mathcal{R}(e)\} \cap \mathcal{R}(s)$
`numToInt`	$\{((lit), (int))\}$
`toInt`	$\{((int), int)\}$
`getName`	$\bigcup_{\mathsf{x}} \{((var), ident(\mathsf{x}))\}$
`read`	$\bigcup_{R, \mathsf{x}, \tau \in \{int, bool\}} \{((state(\mathsf{x}, R), ident(\mathsf{x})), \tau) \mid \mathsf{x} \in R\}$
`add`	$\{(int, int), int\}$
`eq`	$\{((int, int), bool)\}$
`neq`	$\{((int, int), bool)\}$
`true`	$\{((), bool)\}$
`false`	$\{((), bool)\}$
`write`	$\bigcup_{R, \mathsf{x}, \tau \in \{int, bool\}} \{((state(\mathsf{x}, R), ident(\mathsf{x}), \tau), state(R)) \mid \mathsf{x} \in R\}$
`id`	$\bigcup_{\tau \in \mathcal{T}} \{(\tau), \tau\}$
`isInDom`	$\bigcup_{R, \mathsf{x}} \{((state(R), ident(\mathsf{x})), (state(\mathsf{x}, R), ident(\mathsf{x}))) \mid \mathsf{x} \in R\}$
`isInt`	$\bigcup_{\tau \in \{bool, int\}} \{((\tau), (int))\}$
`isTrue`	$\bigcup_{\tau \in \{bool, int\}} \{((\tau), (bool))\}$
`isFalse`	$\bigcup_{\tau \in \{bool, int\}} \{((\tau), (bool))\}$

Fig. 4. Typing for the example with resources

$dom\left(H'_{R'}\right)$ to values, where $H''_{R \cup R'}(\mathsf{x}) = H_R(\mathsf{x})$ if $\mathsf{x} \in dom(H_R)$, and $H''_{R \cup R'}(\mathsf{x}) = H'_{R'}(\mathsf{x})$ if $\mathsf{x} \in dom\left(H'_{R'}\right)$.

We easily show that value transformers are consistent with their type, and that the concrete interpretation of predicates and functions are framing consistent.

Definition 5.4. A set of semantic triples T is compatible with $\odot_{\tau_v}$ iff for every $(v_1, t, v_2) \in T$, there exists τ_1 and τ_2 such that $v_1 : \tau_1$, $v_2 : \tau_2$, $\odot_{\tau_v}(\tau_1)\downarrow$, and $\odot_{\tau_v}(\tau_2)\downarrow$.

Let $\oplus_v : \odot_{\tau_v}$. If T is compatible with $\odot_{\tau_v}$, we write $T \oplus_v$ for $\{(v_1 \oplus_v, t, v_2 \oplus_v) \mid (v_1, t, v_2) \in T\}$.

Lemma 5.5. *Let T be a well-typed set and $\oplus_v : \odot_{\tau_v}$ such that T is compatible with $\odot_{\tau_v}$. Then $T \oplus_v$ is a well-typed set.*

Proof. Let $\left(v'_1, t, v'_2\right) \in T \oplus_v$ and τ'_1 such that $v'_1 : \tau'_1$. There is $(v_1, t, v_2) \in T$ such that $v'_1 = v_1 \oplus_v$ and $v'_2 = v_2 \oplus_v$. From $v_1 \oplus_v : \tau'_1$, we get a type τ_1 such that $\tau'_1 = \odot_{\tau_v}(\tau_1)$ and $v_1 : \tau_1$. As T is well-typed, there is $(\tau_1, \tau_2) \in \mathcal{R}(t)$ such that $v_2 : \tau_2$. As $\odot_{\tau_v}$ is consistent with $\mathcal{R}(t)$, we have $(\odot_{\tau_v}(\tau_1), \odot_{\tau_v}(\tau_2)) \in \mathcal{R}(t)$, with $v_2 \oplus_v : \odot_{\tau_v}(\tau_2)$, as needed. □

Lemma 5.6. *Let T' be a set compatible with $\odot_{\tau_v}$.*

- *If* $T' \subseteq NS_{\text{WEAKEN}}(T)$, *then there exists* $T'' \subseteq T$ *that is compatible with* $\odot_{\tau_v}$ *and* $T' \subseteq NS_{\text{WEAKEN}}(T'')$.
- *If* $T' \subseteq NS_{\text{TRACE-PARTITIONING}}(T)$, *then there exists* $T'' \subseteq T$ *that is compatible with* $\odot_{\tau_v}$ *and* $T' \subseteq NS_{\text{TRACE-PARTITIONING}}(T'')$.

Proof. See supplementary material. □

Definition 5.7. Let T be a set of semantic triples compatible with $\odot_{\tau_v}$. We write $F^\sharp_{\odot_{\tau_v}}(T)$ the largest subset of $F^\sharp(T)$ that is compatible with $\odot_{\tau_v}$.

In the following, we write $\Sigma \oplus_v$ for $\{x \mapsto \Sigma(x) \oplus_v \mid x \in dom(\Sigma)\}$.

Lemma 5.8. *Let* $t = c\overrightarrow{t_n}$ *a closed term,* $(\tau_\sigma, \_) \in \mathcal{R}(t)$, $\sigma : \tau_\sigma$, *and* $\oplus_v : \odot_{\tau_v}$ *such that* $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$, $\odot_{\tau_v}(\tau_\sigma) \downarrow$ *and* T *is compatible with* $\odot_{\tau_v}$. *For any rule* $Rule\left(c\overrightarrow{x_{t_n}}\right)$, *we have*

- $\left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_T(\Sigma) \oplus_v \downarrow$;
- $\left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_{T \oplus_v}(\Sigma \oplus_v) \downarrow$;
- $\left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_T(\Sigma) \oplus_v = \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_{T \oplus_v}(\Sigma \oplus_v)$.

Proof. See supplementary material. □

Lemma 5.9. *Let* $T \subseteq F^\sharp(T)$ *that is* $\odot_{\tau_v}$ *compatible and* k *correct. Let* $\oplus_v : \odot_{\tau_v}$. *For any* $l \leq k$, *let* $\gamma^l(T) = \left\{(\sigma, t, o) \,\middle|\, \sigma, t \Downarrow^l o \wedge \left(\sigma^\sharp, t, o^\sharp\right) \in T \wedge \sigma \in \gamma\left(\sigma^\sharp\right)\right\}$. *For any* $\left(\sigma^\sharp, t, o^\sharp\right) \in T$ *and any* $\sigma \in \gamma\left(\sigma^\sharp\right)$, *where* $t = c\overrightarrow{t_n}$, *for any rule* $Rule\left(c\overrightarrow{x_{t_n}}\right)$, *if* $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_n \mapsto t_n}$ *and* $o' \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_{\Downarrow^l}(\Sigma \oplus_v)$, *then* $o' \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_{\gamma^l(T) \oplus_v}(\Sigma \oplus_v)$.

Proof. See supplementary material. □

Definition 5.10. Let $\oplus_v$ and $\oplus^\sharp_{v^\sharp}$ a concrete and an abstract value transformers such that $\oplus_v : \odot_{\tau_v}$, and $\oplus^\sharp_{v^\sharp} : \odot_{\tau_v}$. First, we require abstract values to be *stratified* by resources: if $w^\sharp : \tau$ and $\odot_{\tau_v}(\tau) \downarrow$, then for every τ' such that $w^\sharp : \tau'$, we have $\odot_{\tau_v}(\tau') \downarrow$. These transformers are consistent iff for every τ and $w^\sharp : \tau$ such that $\odot_{\tau_v}(\tau) \downarrow$

$$\forall w : \tau.\ w \in \gamma\left(w^\sharp\right) \wedge v \in \gamma\left(v^\sharp\right) \implies w \oplus_v \in \gamma\left(w^\sharp \oplus^\sharp_{v^\sharp}\right);$$

$$\forall z : \tau \oplus_v.\ z \in \gamma\left(w^\sharp \oplus^\sharp_{v^\sharp}\right) \implies \exists v, w.\ z = w \oplus_v \wedge w \in \gamma\left(w^\sharp\right) \wedge v \in \gamma\left(v^\sharp\right);$$

$$\forall w_1^\sharp : \tau_1, w_2^\sharp : \tau_2.\ \odot_{\tau_v}(\tau_1) \downarrow \wedge\ w_1^\sharp \leq w_2^\sharp \implies \odot_{\tau_v}(\tau_2) \downarrow \wedge\ w_1^\sharp \oplus^\sharp_{v^\sharp} \leq w_2^\sharp \oplus^\sharp_{v^\sharp}.$$

In the following, we assume that value transformers are consistent.

Lemma 5.11. *Let* $t = c\overrightarrow{t_n}$ *closed,* $\oplus^\sharp_{v^\sharp} : \odot_{\tau_v}$, $\sigma^\sharp$ *such that for every* τ, *if* $\sigma^\sharp : \tau$ *then* $(\tau_\sigma, \_) \in \mathcal{R}(t)$ *and* $\odot_{\tau_v}(\tau_\sigma) \downarrow$, *and* T *compatible with* $\odot_{\tau_v}$. *If* $o^\sharp \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T, out\left(\sigma^\sharp, t\right)}\left(x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_n \mapsto t_n}\right)$, *then* $o^\sharp : \tau_o \implies \odot_{\tau_v}(\tau_o) \downarrow$ *and* $o^\sharp \oplus^\sharp_{v^\sharp} \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T \oplus^\sharp_{v^\sharp}, out\left(\sigma^\sharp \oplus^\sharp_{v^\sharp}, t\right)}\left(x_\sigma \mapsto \sigma^\sharp \oplus^\sharp_{v^\sharp} + \overrightarrow{x_n \mapsto t_n}\right)$.

Proof. See supplementary material. □

Definition 5.12. Let $\oplus^\sharp_{v^\sharp} : \odot_{\tau_v}$ and T be set of semantic triples compatible with $\odot_{\tau_v}$. The non-structural frame rule is defined as follows.

$$NS_{\text{FRAME}}\left(T, v^\sharp\right) = T \oplus^\sharp_{v^\sharp}$$

THEOREM 5.13. *Let $\oplus^\sharp_{v^\sharp} : \odot_{\tau_v}$, T be a set of semantic triples compatible with $\odot_{\tau_v}$.*

(1) *If $T \subseteq F^\sharp(T)$, then the frame rule is nostep correct on $\left(T, \oplus^\sharp_{v^\sharp}\right)$.*
(2) *If $T' \subseteq F^\sharp(T)$ and T' is compatible with $\odot_{\tau_v}$, then $NS_{FRAME}\left(T', v^\sharp\right) \subseteq F^\sharp\left(NS_{FRAME}\left(T, v^\sharp\right)\right)$.*
(3) *If $T' \subseteq NS_{WEAKEN}(T)$, then T' is compatible with $\odot_{\tau_v}$ and $NS_{WEAKEN}(T') \subseteq F^\sharp\left(NS_{WEAKEN}(T)\right)$.*
(4) *If $T' \subseteq NS_{TRACE\text{-}PARTITIONING}(T)$, then T' is compatible with $\odot_{\tau_v}$ and $NS_{TRACE\text{-}PARTITIONING}(T') \subseteq F^\sharp\left(NS_{TRACE\text{-}PARTITIONING}(T)\right)$.*

PROOF. See supplementary material. □

LEMMA 5.14. *Let T a $\odot_{\tau_v}$ compatible set, and let $G^\sharp_{\odot_{\tau_v}}(T) = \bigcup_{j=0}^{\infty} G^\sharp_j\left(F^\sharp_{\odot_{\tau_v}}\left(\bigcup_{j=0}^{\infty} G^\sharp_j(T)\right)\right)$. If $T' \subseteq G^\sharp_{\odot_{\tau_v}}(T)$, then T' is $\odot_{\tau_v}$ compatible and for any $\oplus_{v^\sharp} : \odot_{\tau_v}$, $T' \oplus^\sharp_{v^\sharp} \subseteq G^\sharp\left(T \oplus^\sharp_{v^\sharp}\right)$.*

PROOF. By induction on the definition of $G^\sharp_{\odot_{\tau_v}}(T)$, applying Theorem 5.13 at each step. □

LEMMA 5.15. *Let T and T' two $\odot_{\tau_v}$ compatible sets such that $T' \subseteq G^\sharp(T)$. Then $T' \subseteq G^\sharp_{\odot_{\tau_v}}(T)$.*

PROOF. Immediate by induction on the definition of $G^\sharp$, relying on Lemma 5.6 and Theorem 5.13. The middle part where $T' \subseteq F^\sharp(T)$ is also immediate, as T' is $\odot_{\tau_v}$ compatible. □

Example 5.16. We now apply the framework of type transformations presented in this section to the example of Figure 2. Our goal is to add the following simple frame rule into our abstract semanics.

$$\text{FRAME-VAR}\ \frac{H_1^\sharp, t \Downarrow H_2^\sharp}{H_1^\sharp \star H_c^\sharp, t \Downarrow H_2^\sharp \star H_c^\sharp}$$

As in the concrete interpretation 5.3, we only modified the abstract domain of heaps to now include a set R of variables. An abstract heap $H^\sharp_R$ of type $state(R)$ is a partial function from S to abstract values. We define $\gamma\left(H^\sharp_R\right)$ as the set of functions from $dom\left(H^\sharp_R\right)$ to values such that $\gamma\left(H^\sharp_R\right)(\mathsf{x}) \in \gamma\left(H^\sharp_R(\mathsf{x})\right)$. We define $H^\sharp_R \leq {H^\sharp}'_R$ for identical R as $dom\left(H^\sharp_R\right) = dom\left({H^\sharp}'_R\right)$ and for every $\mathsf{x} \in dom\left(H^\sharp_R\right)$, $H^\sharp_R(\mathsf{x}) \leq {H^\sharp}'_R(\mathsf{x})$.

Value transformers are parameterised by abstract heaps, and we write $\star H^\sharp_R$ for $\oplus^\sharp_{H^\sharp_R}$. We have $\star H^\sharp_R : \odot_{state(R)}$. If $v^\sharp : int$ or $v^\sharp : bool$, then $v^\sharp$ is necessarily an abstract value and $v^\sharp \star H^\sharp = v^\sharp$. If $v^\sharp : state(R)$ or $v^\sharp : state(\mathsf{x}, R)$, then $v^\sharp$ is necessarily a state ${H^\sharp}'_{R'}$. If $R \cap R' \neq \emptyset$ then the frame is defined, and we have ${H^\sharp}'_{R'} \star H^\sharp_R = {H^\sharp}''_{R \cup R'}$ be the disjoint union of the two heaps.

We easily show that the abstract interpretations of functions and predicates are framing consistent (Definition 5.2). We also show that the concrete and abstract value transformers $\uplus H$ and $\star H^\sharp$ are consistent (Definition 5.10).

6 PROVING SEMANTIC TRIPLES CORRECT

In the previous sections, we have seen how we can build rules and prove them correct. In particular, Theorem 4.3 shows that the immediate consequence $G^\sharp$ generates correct semantic triples when iterated if all the rules are step-correct.

In this section, we are interested in proving sets of semantic triples to be correct. Section 6.1 presents a proof technique to this end and applies it to an example. Section 6.2 compares our proof technique to other well-known rules from abstract interpretation.

6.1 Proof Technique

We show a general technique to prove that a triple $\left(\sigma^\sharp, t, o^\sharp\right)$ is correct. To this end, we show how to build a T such that $\left(\sigma^\sharp, t, o^\sharp\right) \in T$ and $T \subseteq G^\sharp(T)$. The set T is then correct by Lemma 4.3, hence $\left(\sigma^\sharp, t, o^\sharp\right)$ is correct. In the following, we assume that every set that is built is well-typed.

Example 6.1. We illustrate this section by the following program p.

$$p = (s_l; \mathrm{y} := 2)$$
$$s_l = \mathit{while}\, \neg(\mathrm{x} = 0)\, \mathrm{x} := \mathrm{x} + (-1)$$

We want to show that the following abstract triple is correct.

$$\left(\{\mathrm{x} \mapsto [0, +\infty] + \mathrm{y} \mapsto [-\infty, +\infty]\}, p, \{\mathrm{x} \mapsto [0, 0] + \mathrm{y} \mapsto [2, 2]\}\right)$$

We define T_p as follows. Our goal is to find a set T such that $T_p \subseteq T$ and $T \subseteq G^\sharp(T)$, or equivalently a set T' such that $T_p \cup T' \subseteq G^\sharp\left(T_p \cup T'\right)$.

$$T_p = \left\{\left(\{\mathrm{x} \mapsto [0, +\infty] + \mathrm{y} \mapsto [-\infty, +\infty]\}, p, \{\mathrm{x} \mapsto [0, 0] + \mathrm{y} \mapsto [2, 2]\}\right)\right\}$$

Finding such a set T or T' can be complex. We thus provide the following technique.

First, we consider two families of sets $\overrightarrow{T_n}$ and $\overrightarrow{T'_n}$ such that $T = \bigcup_i T_i$, and $\bigcup_i T'_i \subseteq T$. Each pair T_i and T'_i corresponds to a fragment of t we want to show correct. If we can show that $T_i \subseteq G^\sharp\left(T'_i\right)$ for every i, then $T_i \subseteq G^\sharp(T)$ for every i by monotony of $G^\sharp$, and we conclude that $T \subseteq G^\sharp(T)$.

For each pair T_i and T'_i, we may simplify things further by focusing on the parts of the state that are modified. Let $\oplus_{\upsilon^\sharp} : \odot_{\tau_\upsilon}$ a value transformer, and assume that $T_i = T_i^f \oplus^\sharp_{\upsilon^\sharp}$ and $T'_i = T_i'^f \oplus^\sharp_{\upsilon^\sharp}$. If we can show that $T_i^f \subseteq G^\sharp_{\odot_{\tau_\upsilon}}\left(T_i'^f\right)$, for instance by showing $T_i^f \subseteq G^\sharp\left(T_i'^f\right)$ with sets compatible with $\odot_{\tau_\upsilon}$ and Lemma 5.15, then we can conclude by Lemma 5.14.

For the last step, we exhibit two sequences of sets $T_i^f = T_{i,0}^f, \ldots, T_{i,k}^f$ and $T_{i,0}'^f, \ldots, T_{i,l}'^f = T_i'^f$, such that $T_{i,j-1}^f \subseteq NS_j\left(T_{i,j}^f\right)$ for every $j \in [1..k]$, $T_{i,j-1}'^f \subseteq NS'_j\left(T_{i,j}'^f\right)$ for every $j \in [1..l]$, and $T_{i,k}^f \subseteq F^\sharp_{\odot_{\tau_\upsilon}}\left(T_{i,0}'^f\right)$. Since every set is $\odot_{\tau_\upsilon}$ compatible and the operators are monotonous, we apply the following reasoning:

$$\begin{aligned} T_i^f = T_{i,0}^f \subseteq NS_0\left(T_{i,1}^f\right) \subseteq NS_0\left(NS_1\left(T_{i,2}^f\right)\right) \subseteq \ldots \subseteq NS_0\left(NS_1\left(\ldots NS_{k-1}\left(T_{i,k}^f\right)\ldots\right)\right) \subseteq \\ NS_0\left(NS_1\left(\ldots NS_{k-1}\left(F^\sharp_{\odot_{\tau_\upsilon}}\left(T_{i,0}'^f\right)\right)\ldots\right)\right) \subseteq \\ NS_0\left(NS_1\left(\ldots NS_{k-1}\left(F^\sharp_{\odot_{\tau_\upsilon}}\left(NS'_1\left(T_{i,1}'^f\right)\right)\right)\ldots\right)\right) \subseteq \ldots = G^\sharp_{\odot_{\tau_\upsilon}}\left(T_i'^f\right) \end{aligned}$$

Note that if $T \subseteq G^\sharp\left(G^\sharp(T)\right)$, then by monotony, $T \cup G^\sharp(T) \subseteq G^\sharp\left(T \cup G^\sharp(T)\right)$. This enables to chain the proof technique above, making several applications of the immediate consequence $G^\sharp$. Let us now show how this instantiates on the example.

Example 6.2. In the Example 6.1, we consider the following sets for each subterm of p.

t_i	T_i	T'_i
$s_1 = (\mathsf{x} := \mathsf{x} + (-1))$	$T_1 = \{(\{\mathsf{x} \mapsto [1, +\infty]\}, s_1, \{\mathsf{x} \mapsto [0, +\infty]\})\}$	$\emptyset$
$e_2 = \neg(\mathsf{x} = 0)$	$T_2 = \{(\{\mathsf{x} \mapsto [1, +\infty]\}, e_2, \mathtt{tt}), (\{\mathsf{x} \mapsto [0, 0]\}, e_2, \mathtt{ff})\}$	$\emptyset$
s_l	$T_l = \{(\{\mathsf{x} \mapsto [0, +\infty]\}, s_l, \{\mathsf{x} \mapsto [0, 0]\})\}$	$T_1 \cup T_2 \cup T_l$
$s_3 = (\mathsf{y} := 2)$	$T_3 = \{(\{\mathsf{y} \mapsto [-\infty, +\infty]\}, s_3, \{\mathsf{y} \mapsto [2, 2]\})\}$	$\emptyset$
p	T_p	$T_l \cup T_3$

The associated proof for the terms s_1, e_2, and s_4 is straightforward: the sets T_i for $i \in \{1, 2, 3\}$ are such that $T_i \subseteq F^{\sharp^j}(\emptyset)$ for some j. This means that we can directly derive the triples of $T_1 \cup T_2 \cup T_3$ from the axioms of the abstract semantics. Furthermore, there is exactly one rule which applies at each step of these derivations: the universal quantifier of $F^\sharp$ acts as the existential quantifier of F.

Let us now consider the case of the loop statement s_3. If we try to straightforwardly build a derivation from the triple $(\{\mathsf{x} \mapsto [0, +\infty]\}, s_3, \{\mathsf{x} \mapsto [0, 0]\})$ and the set T_1, we fail to get the precise result $\{\mathsf{x} \mapsto [0, 0]\}$. Indeed the expression e_2 of the loop is then naively associated to the triple $(\{\mathsf{x} \mapsto [0, +\infty]\}, e_2, \top_{bool})$. We thus infer that Rule WhFalse may apply. The definition of $F^\sharp$ forces us to consider the rule WhFalse as in Figure 2, but with x_σ associated to the first component of the initial triple, that is to $\{\mathsf{x} \mapsto [0, +\infty]\}$. When we return x_σ in Rule WhFalse, we thus return $\{\mathsf{x} \mapsto [0, +\infty]\}$ instead of the expected $\{\mathsf{x} \mapsto [0, 0]\}$. This phenomenon happens because our framework does not assume anything about the analysed programming language. In particular, the notions of expressions and statements are not formalised: we consider both to be terms. Our framework is thus unable to straightforwardly infer that expressions do not change the global state, as this is not the case is all programming languages.

To precisely analyse the statement s_l, we need to use trace partitioning. We can split the abstract value of x into two abstract values: one in which the loop condition is always false, and one in which it is always true. This enables us to filter the state and infer more precise results. We thus consider the set $T_l^p = T_{l,a}^p \cup T_{l,b}^p$ with $T_{l,a}^p = \{(\{\mathsf{x} \mapsto [1, +\infty]\}, s_l, \{\mathsf{x} \mapsto [0, 0]\})\}$ and $T_{l,b}^p = \{(\{\mathsf{x} \mapsto [0, 0]\}, s_l, \{\mathsf{x} \mapsto [0, 0]\})\}$. We have $T_l \subseteq NS_{\text{TRACE-PARTITIONING}}\left(T_l^p\right)$. We can now prove that $T_{l,a}^p \subseteq F^\sharp(T_1 \cup T_2 \cup T_l)$: only Rule WhTrue applies, the set T_2 refers to the expression and T_1 to the statement, then the recursive hypothesis we need is exactly the one of T_l. The other side is straightforward, as only Rule WhFalse applies, and we have this time a precise x_σ to return: $T_{l,b}^p \subseteq F^\sharp(\emptyset)$. We thus have $T_l \subseteq NS_{\text{TRACE-PARTITIONING}}\left(F^\sharp(T_1 \cup T_2 \cup T_l)\right) \subseteq G^\sharp(T_1 \cup T_2 \cup T_l)$ by monotonicity. Note how we need T_l to prove T_l, and how we force the presence of the immediate consequence to appear in the inclusion $T_l \subseteq G^\sharp(T_1 \cup T_2 \cup T_l)$. This shows that the set $T_1 \cup T_2 \cup T_l$ is correct. This exactly corresponds to the proof technique of loop invariants, although more general: for instance, recursive function calls can also be treated in this way.

We now consider how we can use the frame rule to merge the results of T_l and T_3 into T_p. As we just want to add variable to an existing semantic triple, we use the value transformers defined in Example 5.16. We consider the following set T'_p.

$$
\begin{aligned}
T'_p &= T'_{p,a} \cup T'_{p,b} \\
T'_{p,a} &= \{(\{\mathsf{x} \mapsto [0, +\infty] + \mathsf{y} \mapsto [-\infty, +\infty]\}, s_l, \{\mathsf{x} \mapsto [0, 0] + \mathsf{y} \mapsto [-\infty, +\infty]\})\} \\
T'_{p,b} &= \{(\{\mathsf{x} \mapsto [0, 0] + \mathsf{y} \mapsto [-\infty, +\infty]\}, s_3, \{\mathsf{x} \mapsto [0, 0] + \mathsf{y} \mapsto [2, 2]\})\}
\end{aligned}
$$

We have $T_p \subseteq F^\sharp\left(T'_p\right)$: this is just an application of Rule Seq, hence $T_p \subseteq G^\sharp\left(T'_p\right) \subseteq G^\sharp\left(T'_p \cup T_p\right)$. Let us thus show that both $T'_{p,a}$ and $T'_{p,b}$ are correct. We have $T'_{p,a} = T_l \star \mathsf{y} \mapsto [-\infty, +\infty]$ and

$$\textsc{IfAbstract}\ \frac{H_0^\sharp, e \Downarrow^\sharp \top_{bool} \qquad H_0^\sharp, s_1 \Downarrow^\sharp H_1^\sharp \qquad H_0^\sharp, s_2 \Downarrow^\sharp H_2^\sharp}{H_0^\sharp, \mathit{if}\, e\, s_1\, s_2 \Downarrow^\sharp H_1^\sharp \sqcup H_2^\sharp} \qquad \textsc{IfTrueAbstract}\ \frac{H_0^\sharp, e \Downarrow^\sharp \mathit{true} \qquad H_0^\sharp, s_1 \Downarrow^\sharp H_1^\sharp}{H_0^\sharp, \mathit{if}\, e\, s_1\, s_2 \Downarrow^\sharp H_1^\sharp}$$

$$\textsc{WhileAbstract}\ \frac{H^\sharp, s \Downarrow^\sharp H^\sharp}{H^\sharp, \mathit{while}\, e\, s \Downarrow^\sharp H^\sharp}$$

Fig. 5. Common abstract rules from abstract interpretation

$T'_{p,b} = T_3 \star \mathsf{x} \mapsto [0,0]$. By Lemmas 5.15 and 5.14, $T'_{p,a} \subseteq G^\sharp\left(T'_{p,a}\right)$ and $T'_{p,b} \subseteq G^\sharp\left(T'_{p,b}\right)$ as $\star \mathsf{y} \mapsto [-\infty, +\infty] : \odot_{state(\mathsf{y})}$, $\star \mathsf{x} \mapsto [0,0] : \odot_{state(\mathsf{x})}$, T_l is $\odot_{state(\mathsf{y})}$ compatible, and T_3 is $\odot_{state(\mathsf{x})}$ compatible. Hence $T'_p \subseteq G^\sharp\left(T'_p\right) \subseteq G^\sharp\left(T'_p \cup T_p\right)$. We conclude by $T'_p \cup T_p \subseteq G^\sharp\left(T'_p \cup T_p\right)$ and by Theorem 4.3.

6.2 Proving Rules From Abstract Interpretation

We now show how we can get back usual rules from abstract interpretation into our formalism. The methods presented here are general, and could be used to certify an analyser. We consider the widely used rules of Figure 5. Rule WhileAbstract in particular is used to certify abstract interpretation techniques, such as widening and narrowing [Cousot and Cousot 1977].

There rules are proven admissible in our framework: modeling rules as functions from and to sets of semantic triples, if we have $T \subseteq G^\sharp(T)$, then we also have $T \cup \textsc{Rule}(T) \subseteq G^\sharp(T \cup \textsc{Rule}(T))$ for our considered rule Rule. This property implies that we can use an admissible rule to augment a set of correct semantic triple, leaving it correct, which corresponds to the intuitive meaning of a rule. By monotony, we can alternatively prove that $\textsc{Rule} \subseteq G^\sharp(T)$ for all T well-typed: this implies the previous property.

We first consider Rule IfAbstract. This rule assumes that the expression e of a conditional expression evaluates to $\top_{bool}$. In other words, we do not know whether the expression can return `tt` or `ff`. We thus evaluate both statements s_1 and s_2 in parallel and merge their results using a least upper bound $\sqcup$. We do not define the operation $\sqcup$, but we assume that $H_i^\sharp \leq H_1^\sharp \sqcup H_2^\sharp$ for $i \in \{1,2\}$, which is a usual hypothesis on such an operation in abstract interpretation.

This rule is captured from the immediate consequence $F^\sharp$ combined with the weaken rule from Section 4. Let T a well-typed set of semantic triples. We define T_{IF} as follows. It corresponds to the action of Rule IfAbstract. Our goal is to show that $T_{\text{IF}} \subseteq G^\sharp(T)$.

$$T_{\text{IF}} = \left\{\left(H_0^\sharp, \mathit{if}\, s_1\, s_2, H_1^\sharp \sqcup H_2^\sharp\right) \middle| \left(H_0^\sharp, e, \top_{bool}\right) \in T \wedge \left(H_0^\sharp, s_1, H_1^\sharp\right) \in T \wedge \left(H_0^\sharp, s_2, H_2^\sharp\right) \in T\right\}$$

We define $T_w = T \cup \left\{\left(H_0^\sharp, s_i, H_1^\sharp \sqcup H_2^\sharp\right) \middle| i \in \{1,2\} \wedge \left(H_0^\sharp, s_1, H_1^\sharp\right) \in T \wedge \left(H_0^\sharp, s_2, H_2^\sharp\right) \in T\right\}$. Rule weaken directly applies: we get $T_w \subseteq NS_{\text{WEAKEN}}(T)$ using the hypothesis on the $\sqcup$ relation above. We now show that $T_{\text{IF}} \subseteq F^\sharp(T_w)$ to prove $T_{\text{IF}} \subseteq G^\sharp(T)$. Following Definition 5 of the immediate consequence, we consider all rules applying on the term $\mathit{if}\, s_1\, s_2$. There are two of them: IfTrue and IfFalse. We first consider the abstract interpretation of the rule IfTrue. Let $\left(H_0^\sharp, s_1, H_1^\sharp \sqcup H_2^\sharp\right) \in T_w$, such that $\left(H_0^\sharp, e, \top_{bool}\right) \in T$. From $\left(H_0^\sharp, s_i, H_1^\sharp \sqcup H_2^\sharp\right) \in T_w$, we derive (see Definition 3.6) $H_1^\sharp \sqcup H_2^\sharp \in [\![\textsc{IsTrue}]\!]^\sharp_{T_w,O}\left(x_\sigma \mapsto H_0^\sharp + x_e \mapsto e + x_{s_1} \mapsto s_1 + x_{s_2} \mapsto s_2\right)$, where $O = out\left(H_0^\sharp, \mathit{if}\, s_1\, s_2\right)$. The case of Rule IfFalse is symmetrical: we get $H_1^\sharp \sqcup H_2^\sharp \in F^\sharp(T_w)$, and thus $T_{\text{IF}} \subseteq F^\sharp(T_w)$. We thus

get that if $T \subseteq G^\sharp(T)$, then $T \cup T_{\text{IF}} \subseteq G^\sharp(T \cup T_{\text{IF}})$, and is correct by Theorem 4.3: Rule IfAbstract is admissible in our framework.

Rule IfAbstract is useful when the evaluation of the expression is not precise. We now show that our framework can be precise if needed by showing that Rule IfTrueAbstract is admissible. More precisely, we show that Rule IfTrueAbstract is exactly the result of the abstract immediate consequence:

$$\left\{\left(H_0^\sharp, \mathit{if}\, e\, s_1\, s_2, H_1^\sharp\right) \middle| \left(H_0^\sharp, e, \mathit{true}\right) \in T \wedge \left(H_0^\sharp, s_1, H_1^\sharp\right) \in T\right\} \subseteq F^\sharp(T)$$

We consider a semantic triple $\left(H_0^\sharp, \mathit{if}\, e\, s_1\, s_2, H_1^\sharp\right)$ such that $\left(H_0^\sharp, e, \mathit{true}\right) \in T$ and $\left(H_0^\sharp, s_1, H_1^\sharp\right) \in T$. We want to show that this semantic triple is in $F^\sharp(T)$. We thus consider again the rules IfTrue and IfFalse. The abstract rule IfTrue exactly follows the same steps than the considered Rule IfTrueAbstract: we have $H_1^\sharp \in [\![\text{IsTrue}]\!]^\sharp_{T,O}\left(x_\sigma \mapsto H_0^\sharp + x_e \mapsto e + x_{s_1} \mapsto s_1 + x_{s_2} \mapsto s_2\right)$, where $O = \mathit{out}\left(H_0^\sharp, \mathit{if}\, e\, s_1\, s_2\right)$ is the set of abstract states with the same domain than $H_0^\sharp$. We now consider Rule IfFalse. The abstract interpretation A (IsFalse) of the predicate IsFalse is defined but false when applied to $H_0^\sharp$. The abstract interpretation of the rule IfFalse is thus equal to the entire set $O = \mathit{out}\left(H_0^\sharp, \mathit{if}\, e\, s_1\, s_2\right)$ (see the predicate case of Definition 3.6), effectively accepting all results. Thus, both rules IfTrue and IfFalse accept the triple $\left(H_0^\sharp, \mathit{if}\, e\, s_1\, s_2, H_1^\sharp\right)$, making Rule IfTrueAbstract admissible in our framework.

We now consider Rule WhileAbstract. To prove it admissible, we need to use the general definition of admissibility: given $T \subseteq G^\sharp(T)$, we prove $T_{\text{WH}} = \left\{\left(H^\sharp, \mathit{while}\, e\, s, H^\sharp\right) \middle| \left(H^\sharp, s, H^\sharp\right) \in T\right\} \subseteq G^\sharp(T \cup T_{\text{WH}})$. There are two rules which applies on the term $\mathit{while}\, e\, s$: the rules WhTrue and WhFalse. Both rules require to evaluate the expression e. At first sight, it may seem that it is possible not to have the resources needed to evaluate e in the input state. Types avoid this issue. Indeed, the types for the statement $\mathit{while}\, e\, s$ shown in Figure 4 ensure that the abstract states $H^\sharp : \mathit{state}(R)$ associated with $\mathit{while}\, e\, s$ are also well-typed for e. We thus know by type that there exists a result $v^\sharp$ for the expression e for each well-typed $H^\sharp$ appearing in Rule WhileAbstract. Without loss of generality, we can suppose that T contains the associated semantic triples for e: as for the set T_2 of Example 6.2, we can generate these triples by iterating $F^\sharp$ from the empty set. In the case of Rule WhileAbstract, we do not need to be precise on these results $v^\sharp$. We apply Rule weaken on T to extend all these results $v^\sharp$ to $(\top_{\mathit{int}}, \top_{\mathit{bool}})$: as shown in Example 3.2, the return type of the expression e is the type of values, not of booleans. This follows the choice of our base language to be untyped, but we could have manipulated typed expressions without trouble. The state of the proof is now similar to the proof of admissibility of Rule IfAbstract: we consider both rules WhTrue and WhFalse. Rule WhFalse is straightforward: it returns directly $H^\sharp$, which is what is expected in the set T_{WHF}. The case of Rule WhTrue is a little more complex, as it performs a recursive call to $\mathit{while}\, e\, s$. This is not an issue as we have $T \cup T_{\text{WHF}}$ as an argument of $G^\sharp$: we can directly reuse the corresponding semantic triple if T_{WHF} that we started with, that also accepted $H^\sharp$ as a result. This proves the target inclusion, and thus proves Rule WhileAbstract admissible.

7 RELATED WORK

Schmidt initiated the abstract interpretation of big-step operational semantics [Schmidt 1995] by showing how to abstract derivation trees (using co-induction to harness infinite derivations) and derived classical data flow and control flow analyses as abstract interpretations. The present work proposes a fully systematic way of deriving abstractions, and extends to a larger class of abstract domains. Bodin *et al.* [Bodin et al. 2015] apply Schmidt's framework to the particular format of

pretty big step semantics and show how this leads to an abstract interpretation framework in the setting of a While language that can be formalised in the Coq proof assistant. The present paper generalises this work to big-step semantics and adds a frame rule for the derived logics.

Other systematic derivations of static analyses have taken small-step operational semantics as starting point. Schmidt [Schmidt 1997] discusses the general principles for such an approach and compares small-step and big-step operational semantics as foundations for abstract interpretation. Cousot [Cousot 1999] shows how to derive static analyses for an imperative language using the principles of abstract interpretation. Midtgaard and Jensen [Midtgaard and Jensen 2008] use a similar approach for calculating control-flow analyses for functional languages from operational semantics in the form of abstract machines. Van Horn and Might [Van Horn and Might 2010] show how a series of analyses for functional languages can be derived from operational semantics in the form of abstract machines.

The $\mathbb{K}$ semantic framework [Roşu and Şerbănuţă 2010] is a formalism for defining the semantics of programming languages using rewriting. A number of languages have been specified in $\mathbb{K}$, including C and JavaScript. The $\mathbb{K}$ framework comes with a methodology for defining program analyses, based on matching logic [Ştefănescu et al. 2016]. This methodology consists of a set of language-independent inference rules which are combined with a specific $\mathbb{K}$ semantics for a language to obtain a program verifier for that language. The framework can encode separation logic formulae but does not include a frame rule.

The notion of rule formats has been investigated intensively in the setting of process algebras, where it is used to prove general results (notably, that bisimulation is a congruence), for all transition systems adhering to a particular rule format, see *e.g.* [Mousavi et al. 2007]. In the setting of functional languages, Howe [Howe 1996] defines a rule format for guaranteeing similar properties. Sands [Sands 1997] defines a proof format and develops a meta-theory for SOS specifications of functional that automatically provides an syntactic version of continuity and proof principles for proving program equivalences. These works have in common that that they deal with small-step operational semantics. Leroy used big-step operational semantics in an earlier formalization of Compcert's C semantics, and developed a proof principle based on co-induction for this semantics [Leroy and Grall 2009].

There are several approaches to proving correctness of separation logics. The views framework [Dinsdale-Young et al. 2013] provides a general soundness result for concurrent program logics and type systems written in the framework. It introduced a method for proving soundness where the frame is incorporated in the translation from local views (abstract local states) to the global state of the operational semantics. The Iris framework [Jung et al. 2015] also provides a general soundness result using this method. In contrast, the original work on separation logic proposed a different method of proving soundness. They incorporated locality and frame into the operational semantics and proving soundness by relating local assertions to the concrete local state of the operational semantics. This approach has been revisited, first by Raza and Gardner [Raza and Gardner 2009] and then by Costanzo and Shao [Costanzo and Shao 2012], who provide stronger conditions of locality in the operational semantics to provide better definitions of footprints for operations. Our work linking abstract interpretation with separation logics provides a general soundness result using this second method.

8 CONCLUSION

We have shown that natural (big-step) semantics can serve as a semantic basis for deriving program analyses in a completely systematic manner. By imposing a typing discipline on the rule format for defining such semantics, we obtain a framework in which we can define both standard and abstract interpretations. Furthermore, we prove that the abstract interpretation covers all executions that

are deducible in the standard interpretation, as soon as all atomic semantic functions and predicates respect a correctness relation. The locality of this correctness relation enables to easily add or change rules without having to prove again the (global) coverage of the abstract interpretation.

Our typing discipline on operational semantics provides a framework for combining abstract interpretation with principles from separation logic. In our framework, resources are reflected into types. This leads to a solution to the long-standing open question of how to design a framework for integrating the partial order-based abstract interpretation with the spatial properties and reasoning from separation logic, in particular the frame rule, in a manner that can be proved correct with respect to a semantics.

Although we do not propose specific analysers, we provide a framework and a proof technique that greatly simplify the development and proof of correctness of such analysers. The overall methodology proposed here is amenable to a mechanized formalisation in a proof assistant. The formalisation in *e.g.*, Coq shall provide a systematic way of obtaining certified abstract interpretations [Cachera et al. 2005; Klein and Nipkow 2003] that can scale to semantic definitions of full-fledged languages. This will offer an infra-structure that will both guide and considerably ease the proof effort of the certified program analysis designer, compared to state-of-the-art certified static analyzers such as the Verasco C analyzer [Jourdan et al. 2015].

REFERENCES

Martin Bodin, Thomas Jensen, and Alan Schmitt. 2015. Certified Abstract Interpretation with Pretty-Big-Step Semantics. In *Conference on Certified Programs and Proofs*.

Richard Bornat, Cristiano Calcagno, and Hongseok Yang. 2006. Variables as Resource in Separation Logic. *Conference on Mathematical Foundations of Programming Semantics* (2006).

David Cachera, Thomas Jensen, David Pichardie, and Vlad Rusu. 2005. Extracting a Data Flow Analyser in Constructive Logic. *Theorical Computer Science* (2005), 56–78.

Cristiano Calcagno, Dino Distefano, Jérémy Dubreil, Dominik Gabi, Pieter Hooimeijer, Martino Luca, Peter OHearn, Irene Papakonstantinou, Jim Purbrick, and Dulma Rodriguez. 2015. Moving Fast with Software Verification. In *NASA Formal Methods Symposium*.

David Costanzo and Zhong Shao. 2012. A Case for Behavior-Preserving Actions in Separation Logic. In *Asian Symposium on Programming Languages and Systems*.

Patrick Cousot. 1999. The Calculational Design of a Generic Abstract Interpreter. In *Calculational System Design*.

Patrick Cousot and Radhia Cousot. 1977. Abstract Interpretation: A Unified Lattice Model for Static Analysis of Programs by Construction or Approximation of Fixpoints. In *Symposium on Principles of Programming Languages*.

Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2005. The ASTREÉ Analyzer. In *European Symposium on Programming*.

Andrei Ştefănescu, Daejun Park, Shijiao Yuwen, Yilong Li, and Grigore Roşu. 2016. Semantics-Based Program Verifiers for All Languages. In *International Conference on Object-Oriented Programming, Systems, Languages, and Applications*.

Thomas Dinsdale-Young, Lars Birkedal, Philippa Gardner, Matthew J. Parkinson, and Hongseok Yang. 2013. Views: compositional reasoning for concurrent programs. In *Symposium on Principles of Programming Languages*.

Dino Distefano, Peter W. O'Hearn, and Hongseok Yang. 2006. A Local Shape Analysis Based on Separation Logic. In *Tools and Algorithms for the Construction and Analysis of Systems*.

Facebook 2015. INFER: A tool to detect bugs in Java and C/C++/Objective-C code before it ships. (2015). http://fbinfer.com

Douglas J. Howe. 1996. Proving Congruence of Bisimulation in Functional Programming Languages. *Information and Computation* 124, 2 (1996), 103–112.

Samin Ishtiaq and Peter O'Hearn. 2001. BI as an Assertion Language for Mutable Data Structures. (2001).

Jacques-Henri Jourdan, Vincent Laporte, Sandrine Blazy, Xavier Leroy, and David Pichardie. 2015. A Formally-verified C Static Analyzer. In *Symposium on Principles of Programming Languages*.

Ralf Jung, David Swasey, Filip Sieczkowski, Kasper Svendsen, Aaron Turon, Lars Birkedal, and Derek Dreyer. 2015. Iris: Monoids and Invariants as an Orthogonal Basis for Concurrent Reasoning. In *Symposium on Principles of Programming Languages*.

Gerwin Klein and Tobias Nipkow. 2003. Verified Bytecode Verifiers. *Theorical Computation Sciences* (2003).

Xavier Leroy and Hervé Grall. 2009. Coinductive Big-Step Operational Semantics. *Information and Computation* 207 (2009), 284–304.

Laurent Mauborgne and Xavier Rival. 2005. Trace Partitioning in Abstract Interpretation Based Static Analyzers. In *European Symposium on Programming*.

Jan Midtgaard and Thomas Jensen. 2008. A Calculational Approach to Control-Flow Analysis by Abstract Interpretation. In *International Static Analysis Symposium*.

Mohammad Reza Mousavi, Michel A. Reniers, and Jan Friso Groote. 2007. SOS formats and meta-theory: 20 years after. *Theoretical Computer Science* 373, 3 (2007), 238–272.

Peter O'Hearn, John C. Reynolds, and Hongseok Yang. 2001. Local Reasoning about Programs that Alter Data Structures. In *International Workshop on Computer Science Logic*.

Mohammad Raza and Philippa Gardner. 2009. Footprints in Local Reasoning. *Logical Methods in Computer Science* 5, 2 (2009).

John C. Reynolds. 2002. Separation Logic: A Logic for Shared Mutable Data Structures. In *Symposium on Logic in Computer Science*.

John C. Reynolds. 2008. An Introduction to Separation Logic. *Engineering Methods and Tools for Software Safety and Security* (2008).

Xavier Rival and Laurent Mauborgne. 2007. The Trace Partitioning Abstract Domain. *Transactions on Programming Languages and Systems* (2007).

Grigore Roşu and Traian Florin Şerbănuţă. 2010. An Overview of the $\mathbb{K}$ Semantic Framework. *Journal of Logic and Algebraic Programming* 79, 6 (2010), 397–434.

David Sands. 1997. From SOS Rules to Proof Principles: An Operational Metatheory for Functional Languages. In *Symposium on Principles of Programming Languages*.

David A. Schmidt. 1995. Natural-Semantics-Based Abstract Interpretation (preliminary version). In *International Static Analysis Symposium*.

David A. Schmidt. 1997. Abstract Interpretation of Small-Step Semantics. In *Workshop on Analysis and Verification of Multiple-Agent Languages*, Vol. 1192. Springer, 76–99.

Alfred Tarski. 1955. A Lattice-Theoretical Fixpoint Theorem and its Applications. *Pacific journal of Mathematics* 5, 2 (1955), 285–309.

David Van Horn and Matthew Might. 2010. Abstracting Abstract Machines. In *International Conference on Functional Programming*.

1

From Typed Natural Semantics to Abstract Interpretation with a Frame Rule, Supplementary Material

ANONYMOUS AUTHOR(S)

We present additional lemmas and detailed proofs for the paper "From Typed Natural Semantics to Abstract Interpretation with a Frame Rule"

ACM Reference format:
Anonymous Author(s). 2017. From Typed Natural Semantics to Abstract Interpretation with a Frame Rule, Supplementary Material. *Proc. ACM Program. Lang.* 1, 1, Article 1 (January 2017), 16 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 PROOFS OF SECTION 2

Lemma 1.1. *Let* $\emptyset \neq \overline{\Gamma}_1 \subseteq \overline{\Gamma}_2$ *and* L *such that* $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}_2\right)(L)$, *then* $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}_1\right)(L) \subseteq \mathtt{ty}\left(\overline{\Gamma}_2\right)(L)$.

Proof. See supplementary material. □

Proof. We proceed by induction on L. The base case is immediate. For the other cases, we rely on the fact that since $\mathtt{ty}\left(\overline{\Gamma}_2\right)(L) \neq \emptyset$, then we know that for every $\Gamma \in \overline{\Gamma}_2$, there is a type in $\mathcal{R}(t')$, $\mathcal{R}(P)$, or $\mathcal{R}(f)$. Hence this must be the case for every $\Gamma \in \overline{\Gamma}_1$ and we can conclude by induction. □

1.1 Additional lemma

Lemma 1.2. *Let* Σ, L, T_1, *and* T_2 *such that* $\mathtt{wf}(dom(\Sigma))(L)$ *and* $T_1 \subseteq T_2$. *If* $[\![L]\!]^?_{T_2}(\Sigma)\downarrow$, *then* $[\![L]\!]^?_{T_1}(\Sigma)\downarrow$.

Proof. We proceed by induction on L. The base case is immediate.

Assume $L = D(x_1, t', x_2) :: L'$. Let r such that $(\Sigma(x_1), t', r) \in T_1 \subseteq T_2$, then from $[\![L]\!]^?_{T_2}(\Sigma)\downarrow$ we deduce that $[\![L']\!]^?_{T_2}(\Sigma + x \mapsto r)\downarrow$. By induction, we then have $[\![L']\!]^?_{T_1}(\Sigma + x \mapsto r)\downarrow$. As this is the case for any r, we can conclude that $[\![L]\!]^?_{T_1}(\Sigma)\downarrow$.

The other cases are immediate as they do not rely on T_1 or T_2, except for the inductive hypothesis. □

1.2 Additional lemma on wf

Lemma 1.3. *For any rule* $Rule\left(c\overrightarrow{x_{t_n}}\right)$, *if* $\mathtt{wf}\left(\{x_\sigma\} \cup \overrightarrow{x_{t_n}}\right)\left(Rule\left(c\overrightarrow{x_{t_n}}\right)\right)$, *then* $\mathtt{wf}\left(\{x_\sigma\} \cup \overrightarrow{x_{t_n}}\right)\left(Rule\left(c\overrightarrow{t_n}\right)\right)$.

Proof. By an immediate induction on the list of hypotheses. □

1.3 Proof of Lemma 2.5

Proof. We prove the following property by induction on L, given a $(\tau_\sigma, \_) \in \mathcal{R}(t)$. Let Σ and $\overline{\Gamma}$ such that $\overline{\Gamma} \neq \emptyset$, $\mathtt{wf}(dom(\Sigma))(L)$, for every $\Gamma \in \overline{\Gamma}$, $dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, and for every $x \in dom(\Sigma)$, $\Sigma(x) : \Gamma(x)$. If $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, then $[\![L]\!]^?_T(\Sigma)\downarrow$.

2017. 2475-1421/2017/1-ART1 $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

The base case is immediate, as empty lists are always defined. Let us consider the inductive case, proceeding by case on the hypothesis. As x_σ^i is chosen fresh in relation to the variables occurring in L, we do not check in every case it is different from these variables.

Assume that L is $D(x_1, t', x_2) :: L'$, and let r such that $(\Sigma(x_1), t', r) \in T$. Let $\Sigma' = \Sigma + x_2 \mapsto r$ and $\overline{\Gamma}'_o = \left\{\Gamma + x_2 \mapsto \tau_2 \,\middle|\, \Gamma \in \overline{\Gamma} \wedge (\Gamma(x_1), \tau_2) \in \mathcal{R}(t')\right\}$. For any $\Gamma \in \overline{\Gamma}$, as T is well typed and $\Sigma(x_1) : \Gamma(x_1)$, let τ_Γ the type such that $(\Gamma(x_1), \tau_\Gamma) \in \mathcal{R}(t')$ and $r : \tau_\Gamma$. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_\Gamma \,\middle|\, \Gamma \in \overline{\Gamma}\right\}$. We immediately have $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$.

We now check that all conditions are met to apply the inductive hypothesis with L', Σ', and $\overline{\Gamma}'$. From $\mathtt{wf}(dom(\Sigma))(L)$ we deduce $\mathtt{wf}(dom(\Sigma) \uplus \{x_2\})(L') = \mathtt{wf}(dom(\Sigma'))(L')$. For any $\Gamma' \in \overline{\Gamma}'$, $dom(\Gamma') = dom(\Gamma) \uplus \{x_2\} = dom(\Sigma) \uplus \left\{x_\sigma^i, x_2\right\} = dom(\Sigma') \uplus \left\{x_\sigma^i\right\}$. We now fix $\Gamma' \in \overline{\Gamma}'$, hence a $\Gamma \in \overline{\Gamma}$ and a τ_Γ such that both $\Sigma(x_1) : \Gamma(x_1)$ and $r : \tau_\Gamma$. Let $x \in dom(\Sigma')$. If $x \neq x_2$, we immediately have $\Sigma'(x) = \Sigma(x) : \Gamma(x) = \Gamma'(x)$ since $x_2 \notin dom(\Sigma) = dom(\Gamma)$. Otherwise, we also immediately have $\Sigma'(x_2) = r : \Gamma'(x_2) = \tau_\Gamma$. Finally, since $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we know that $\mathtt{ty}\left(\overline{\Gamma}'_o\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L)$. As $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$, we have by Lemma 1.1 that $\mathtt{ty}\left(\overline{\Gamma}'\right)(L') \neq \emptyset$. By induction hypothesis for this r, $[\![L']\!]^?_T(\Sigma + x_2 \mapsto r)\downarrow$. As this is the case for every r, we conclude that $[\![L]\!]^?_T(\Sigma)\downarrow$.

Assume that L is $P\left(\overrightarrow{x_n}\right) :: L'$. We first need to check that $\mathtt{I}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ is defined. Let $\Gamma \in \overline{\Gamma}$. By hypothesis, we have $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$. From $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that there exists some $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. Since P is well typed, then $\mathtt{I}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ is defined. We now consider two cases. If $\mathtt{I}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ does not hold, then we are done since $[\![L]\!]^?_T(\Sigma)\downarrow$. If it holds, we need to check that $[\![L']\!]^?_T(\Sigma)\downarrow$. To this end, we will apply the induction hypothesis with L', Σ, and $\overline{\Gamma}' = \left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)\right\}$. First, $\overline{\Gamma}'$ is not empty because $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ hence for each Γ there is a $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. We also conclude that $\mathtt{ty}\left(\overline{\Gamma}'\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$. We have $\mathtt{wf}(dom(\Sigma))(L')$ immediately from $\mathtt{wf}(dom(\Sigma))(L)$. Since $\overrightarrow{x_n} \subseteq dom(\Sigma)$, for any $\Gamma' \in \overline{\Gamma}'$ there is a $\Gamma \in \overline{\Gamma}$ and $dom(\Gamma') = dom(\Gamma) = dom(\Sigma) \uplus \left\{x_\sigma^i\right\}$. Finally, let $\Gamma' \in \overline{\Gamma}'$ and $x \in dom(\Sigma)$. If x is not one of the $\overrightarrow{x_n}$ then we immediately have $\Sigma(x) : \Gamma(x) = \Gamma'(x)$. Otherwise, we need to show that $\Sigma(x_i) : \tau_i = \Gamma'(x_i)$ where $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. Since $\mathtt{I}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ holds and P is well typed, then $\overrightarrow{\Sigma(x_n) : \tau_n}$, as needed. By induction, we deduce that $[\![L']\!]^?_T(\Sigma)\downarrow$ hence $[\![L]\!]^?_T(\Sigma)\downarrow$.

Assume that L is $f\left(\overrightarrow{x_n}\right) = x :: L'$. We first need to check that $\mathtt{I}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$ is defined. Let $\Gamma \in \overline{\Gamma}$. By hypothesis, we have $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$. From $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that there exists some τ such that $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$. Since f is well typed, then $\mathtt{I}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$ is defined. Let $\Sigma' = \Sigma + x \mapsto \mathtt{I}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$ and $\overline{\Gamma}' = \left\{\Gamma + x \mapsto \tau \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)\right\}$. We now check we may apply the induction hypothesis with Σ', $\overline{\Gamma}'$, and L'. We first show that $\overline{\Gamma}' \neq \emptyset$. Let $\Gamma \in \overline{\Gamma}$, from $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that there exists some τ such that $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, hence $\overline{\Gamma}'$ is not empty. For every $\Gamma' \in \overline{\Gamma}'$, we have $dom(\Gamma') = dom(\Sigma') \uplus \left\{x_\sigma^i\right\} = dom(\Sigma) \uplus \left\{x, x_\sigma^i\right\}$. From

$\mathtt{wf}\,(dom\,(\Sigma))\,(L)$ we immediately deduce $\mathtt{wf}\,(dom\,(\Sigma'))\,(L')$. Let $\Gamma \in \overline{\Gamma}$ and $\left(\overrightarrow{\Gamma\,(x_n)}, \tau\right) \in \mathcal{R}\,(f)$, we show that $\mathtt{I}\,(f)\left(\overrightarrow{\Sigma\,(x_n)}\right) : \tau$. This is the case since f is well typed. Finally, $\mathtt{ty}\left(\overline{\Gamma}'\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, hence by induction $[\![L']\!]^?_T\,(\Sigma')\downarrow$, hence $[\![L]\!]^?_T\,(\Sigma)\downarrow$.

We now show the thesis. First, let $\overline{\Gamma}_0 = \left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$. Since $Rule\left(\overrightarrow{cx_{t_n}}\right)$ is well typed, we know that $\mathtt{ty}\left(\overline{\Gamma}_0\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. Let $\overline{\Gamma}$ be the singleton $\left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n}\right\}$. As $\mathcal{R}\,(t) \subseteq \mathcal{R}\left(\overrightarrow{cx_{t_n}}\right)$, we have $\overline{\Gamma} \subseteq \overline{\Gamma}_0$. Hence by Lemma 1.1, $\mathtt{ty}\left(\overline{\Gamma}\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. Let $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$. As rules are well formed, we have $\mathtt{wf}\,(dom\,(\Sigma))\left(Rule\left(\overrightarrow{cx_{t_n}}\right)\right)$, hence by Lemma 1.3 $\mathtt{wf}\,(dom\,(\Sigma))\left(Rule\left(\overrightarrow{ct_n}\right)\right)$. For the single $\Gamma \in \overline{\Gamma}$, we have $dom\,(\Gamma) = dom\,(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Sigma\,(x_\sigma) : \tau_\sigma = \Gamma\,(x_\sigma)$, and $\overrightarrow{\Sigma\,(x_{t_n}) : s_n = \Gamma\,(x_{t_n})}$. By the property above, we deduce that $\left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]^?_T\,(\Sigma)\downarrow$. □

1.4 Additional lemma on concrete rules

Lemma 1.4. *If* $[\![L]\!]_{T_1}\,(\Sigma)\downarrow$ *and* $[\![L]\!]_{T_2}\,(\Sigma)\downarrow$, *where* $\mathtt{wf}\,(dom\,(\Sigma))\,(L)$ *and* $T_1 \subseteq T_2$, *then* $[\![L]\!]_{T_1}\,(\Sigma) \subseteq [\![L]\!]_{T_2}\,(\Sigma)$.

Proof. We proceed by induction on L. The base case is immediate as $\{\Sigma\,(x_o)\} \subseteq \{\Sigma\,(x_o)\}$.

Let us consider the inductive case, proceeding by case on the hypothesis.

Assume that L is $D(x_1, t', x_2) :: L'$, and let $o \in [\![L]\!]_{T_1}\,(\Sigma)$. Then there is a r such that $(\Sigma\,(x_1), t', r) \in T_1$ and $o \in [\![L']\!]_{T_1}\,(\Sigma')$, where $\Sigma' = \Sigma + x_2 \mapsto r$. As $T_1 \subseteq T_2$, we immediately have $[\![L']\!]_{T_1}\,(\Sigma')\downarrow$ and $[\![L']\!]_{T_2}\,(\Sigma')\downarrow$. We also have $\mathtt{wf}\,(dom\,(\Sigma'))\,(L')$, hence by induction $o \in [\![L']\!]_{T_2}\,(\Sigma')$, thus $o \in [\![L]\!]_{T_2}\,(\Sigma)$.

The other cases are immediate by induction as T_1 and T_2 are not directly used. □

1.5 Proof of Lemma 2.6

Proof. We prove the following property by induction on L, given a $(\tau_\sigma, \_) \in \mathcal{R}\,(t)$. Let Σ and $\overline{\Gamma}$ such that $\overline{\Gamma} \neq \emptyset$, $\mathtt{wf}\,(dom\,(\Sigma))\,(L)$, for every $\Gamma \in \overline{\Gamma}$, $dom\,(\Gamma) = dom\,(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) = \tau_\sigma$, and for every $x \in dom\,(\Sigma)$, $\Sigma\,(x) : \Gamma\,(x)$. If $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}\right)(L) \subseteq \mathcal{R}\,(t)$, then for every $o \in [\![L]\!]_T\,(\Sigma)$, there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}\,(t)$ such that $o : \tau_o$.

Let $o \in [\![[]]\!]_T\,(\Sigma) = \{\Sigma\,(x_o)\}$. Let $\Gamma \in \overline{\Gamma}$. By hypothesis, we have $o = \Sigma\,(x_o) : \Gamma\,(x_o)$. In addition, $\left(\Gamma\left(x^i_\sigma\right), \Gamma\,(x_o)\right) \in \mathtt{ty}\,([])\left(\overline{\Gamma}\right) \subseteq \mathcal{R}\,(t)$. Let $\tau_o = \Gamma\,(x_o)$, then we have $o : \tau_o$ and $(\tau_\sigma, \tau_o) \in \mathcal{R}\,(t)$, as required.

Let us consider the inductive case, proceeding by case on the hypothesis.

Assume that L is $D(x_1, t', x_2) :: L'$, and let $o \in [\![L]\!]_T\,(\Sigma)$. Then there is a r such that $(\Sigma\,(x_1), t', r) \in T$ and $o \in [\![L']\!]_T\,(\Sigma + x_2 \mapsto r)$. Let $\Sigma' = \Sigma + x_2 \mapsto r$ and $\overline{\Gamma}'_o = \left\{\Gamma + x_2 \mapsto \tau_2 \,\middle|\, \Gamma \in \overline{\Gamma} \wedge (\Gamma\,(x_1), \tau_2) \in \mathcal{R}\,(t')\right\}$. For any $\Gamma \in \overline{\Gamma}$, as T is well typed and $\Sigma\,(x_1) : \Gamma\,(x_1)$, let τ_Γ the type such that $(\Gamma\,(x_1), \tau_\Gamma) \in \mathcal{R}\,(t')$ and $r : \tau_\Gamma$. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_\Gamma \,\middle|\, \Gamma \in \overline{\Gamma}\right\}$. We immediately have $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$.

We now check that all conditions are met to apply the inductive hypothesis with L', Σ', and $\overline{\Gamma}'$. From $\mathtt{wf}\,(dom\,(\Sigma))\,(L)$ we deduce $\mathtt{wf}\,(dom\,(\Sigma) \uplus \{x_2\})\,(L') = \mathtt{wf}\,(dom\,(\Sigma'))\,(L')$. For any $\Gamma' \in \overline{\Gamma}'$, $dom\,(\Gamma') = dom\,(\Gamma) \uplus \{x_2\} = dom\,(\Sigma) \uplus \left\{x_2, x^i_\sigma\right\} = dom\,(\Sigma') \uplus \left\{x^i_\sigma\right\}$. We now fix $\Gamma' \in \overline{\Gamma}'$, hence a $\Gamma \in \overline{\Gamma}$ and a τ_Γ such that both $\Sigma\,(x_1) : \Gamma\,(x_1)$ and $r : \tau_\Gamma$. Let $x \in dom\,(\Sigma')$. If $x \neq x_2$, we immediately have $\Sigma'\,(x) = \Sigma\,(x) : \Gamma\,(x) = \Gamma'\,(x)$ since $x_2 \notin dom\,(\Sigma) = dom\,(\Gamma)$. Otherwise, we also immediately have $\Sigma'\,(x_2) = r : \Gamma'\,(x_2) = \tau_\Gamma$. We also still have $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. Finally, since

$\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we know that $\mathtt{ty}(L')\left(\overline{\Gamma}'_o\right) = \mathtt{ty}(L)\left(\overline{\Gamma}\right)$. As $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$, we have by Lemma 1.1 that $\emptyset \neq \mathtt{ty}(L')\left(\overline{\Gamma}'\right) \subseteq \mathtt{ty}(L')\left(\overline{\Gamma}'_o\right) = \mathtt{ty}(L)\left(\overline{\Gamma}\right) \subseteq \mathcal{R}(t)$. By induction hypothesis, for any $o \in [\![L']\!]_T(\Sigma + x_2 \mapsto r)$, there exists τ_o such that $o : \tau_o$ and $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$.

Assume that L is $P\left(\overrightarrow{x_n}\right) :: L'$, and let $o \in [\![L]\!]_T(\Sigma)$. We then must have $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ and $o \in [\![L']\!]_T(\Sigma)$. We now check we can apply the induction hypothesis with L', Σ, and $\overline{\Gamma}' = \left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)\right\}$. First, $\overline{\Gamma}'$ is not empty because $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ hence for each Γ there is a $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. We also conclude that $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}'\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L) \subseteq \mathcal{R}(t)$. We have $\mathtt{wf}(dom(\Sigma))(L')$ immediately from $\mathtt{wf}(dom(\Sigma))(L)$. Since $\overrightarrow{x_n} \subseteq dom(\Sigma)$, for any $\Gamma' \in \overline{\Gamma}'$ there is a $\Gamma \in \overline{\Gamma}$ and $dom(\Gamma') = dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$. As x^i_σ is distinct from the variables of L, we still have $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. Finally, let $\Gamma' \in \overline{\Gamma}'$ and $x \in dom(\Sigma)$. If x is not one of the $\overrightarrow{x_n}$ then we immediately have $\Sigma(x) : \Gamma(x) = \Gamma'(x)$. Otherwise, we need to show that $\Sigma(x_i) : \tau_i = \Gamma'(x_i)$ where $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. Since $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ holds and P is well typed, then $\overrightarrow{\Sigma(x_n) : \tau_n}$, as needed. By induction, we deduce that for any $o \in [\![L']\!]_T(\Sigma) = [\![L]\!]_T(\Sigma)$ there exists τ_o such that $o : \tau_o$ and $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$.

Assume that L is $f\left(\overrightarrow{x_n}\right) = x :: L'$ and let $o \in [\![L]\!]_T(\Sigma) = [\![L']\!]_T(\Sigma')$ where $\Sigma' = \Sigma + x \mapsto \mathtt{C}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$. Let $\overline{\Gamma}' = \left\{\Gamma + x \mapsto \tau \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)\right\}$. We now check we may apply the induction hypothesis with $\Sigma', \overline{\Gamma}'$, and L'. We first show that $\overline{\Gamma}' \neq \emptyset$. Let $\Gamma \in \overline{\Gamma}$, from $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that there exists some τ such that $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, hence $\overline{\Gamma}'$ is not empty. For every $\Gamma' \in \overline{\Gamma}'$, we have $dom(\Gamma') = dom(\Sigma') \uplus \left\{x^i_\sigma\right\} = dom(\Sigma) \uplus \left\{x, x^i_\sigma\right\}$ and $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. From $\mathtt{wf}(dom(\Sigma))(L)$ we immediately deduce $\mathtt{wf}(dom(\Sigma'))(L')$. Let $\Gamma \in \overline{\Gamma}$ and $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, we show that $\mathtt{C}(f)\left(\overrightarrow{\Sigma(x_n)}\right) : \tau$. This is the case since f is well typed. Finally, $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}'\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L) \subseteq \mathcal{R}(t)$, hence by induction for any $o \in [\![L']\!]_T(\Sigma') = [\![L]\!]_T(\Sigma)$ there exists τ_o such that $o : \tau_o$ and $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$.

We now show the thesis. First, let $\overline{\Gamma}_0 = \left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$. Since $Rule\left(c\overrightarrow{x_{t_n}}\right)$ is well typed, we know that $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}_0\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \subseteq \mathcal{R}(t)$. Let $\overline{\Gamma}$ be the singleton $\left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n}\right\}$ for the τ_σ under consideration. We immediately have $\overline{\Gamma} \subseteq \overline{\Gamma}_0$. Hence by Lemma 1.1, $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \subseteq \mathcal{R}(t)$. Let $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$. As rules are well formed, we have $\mathtt{wf}(dom(\Sigma))\left(Rule\left(c\overrightarrow{x_{t_n}}\right)\right)$, hence $\mathtt{wf}(dom(\Sigma))\left(Rule\left(\overrightarrow{ct_n}\right)\right)$ by Lemma 1.3. For the single $\Gamma \in \overline{\Gamma}$, we have $dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) = \tau_\sigma$, $\Sigma(x_\sigma) : \tau_\sigma = \Gamma(x_\sigma)$, and $\overrightarrow{\Sigma(x_{t_n}) : s_n = \Gamma(x_{t_n})}$. By the property above, we deduce that for every $o \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_T(\Sigma)$ there exists τ_o such that $o : \tau_o$ and $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$. □

1.6 Proof of Lemma 2.7

Proof. Let $(\sigma, \overrightarrow{ct_n}, o) \in F(T)$, then there is a rule $Rule\left(c\overrightarrow{x_{t_n}}\right)$ such that $o \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_T(\Sigma)$ where $(\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)$, $\sigma : \tau_\sigma$, and $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$. Since this rule is well typed and T is well typed, by Lemma 2.6, there exists τ_o such that $o : \tau_o$ and $(\tau_\sigma, \tau_o) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)$, as required. □

1.7 Proof of Lemma 2.8

Proof. Let $T_1 \subseteq T_2$ two well-typed triple set, and $(\sigma, t, o) \in F(T_1)$. Then $t = \overrightarrow{ct_n}$ for some closed $\overrightarrow{t_n}$, and there is a rule $Rule\left(\overrightarrow{cx_{t_n}}\right)$ such that $o \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{T_1}(\Sigma)$ where $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$, $(\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)$, and $\sigma : \tau_\sigma$. By Lemma 2.5, we have $\left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{T_2}(\Sigma)\downarrow$. By Lemma 1.4, we conclude that $o \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{T_2}(\Sigma)$, hence $o \in F(T_2)$. □

2 PROOFS OF SECTION 3

2.1 Additional lemma on abstract rules

Lemma 2.1. *If* $[\![L]\!]^\sharp_{T_1,O}(\Sigma)\downarrow$ *and* $[\![L]\!]^\sharp_{T_2,O}(\Sigma)\downarrow$, *where* $\mathtt{wf}(dom(\Sigma))(L)$ *and* $T_1 \subseteq T_2$, *then* $[\![L]\!]^\sharp_{T_1,O}(\Sigma) \subseteq [\![L]\!]^\sharp_{T_2,O}(\Sigma)$.

Proof. The proof is identical to the one of Lemma 1.4. The only difference in the definition is for the predicate case, where T_1 and T_2 do not directly matter and where the set O is vaccuously included in itself when $\mathtt{A}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ does not hold. □

2.2 Proof of Lemma 3.8

Proof. We prove the following property by induction on L. Let $\mathcal{T}_\sigma = \left\{\tau_\sigma \,\middle|\, \sigma^\sharp : \tau_\sigma\right\}$. We assume that $\mathcal{T}_\sigma \subseteq \{\tau_\sigma \mid (\tau_\sigma, \_) \in \mathcal{R}(t)\}$. Let O such that $\forall o^\sharp \in O$, there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$. Let Σ and $\overline{\Gamma}$ such that $\overline{\Gamma} \neq \emptyset$, $\mathtt{wf}(dom(\Sigma))(L)$, for every $\Gamma \in \overline{\Gamma}$, $dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) \in \mathcal{T}_\sigma$, and for every $x \in dom(\Sigma)$, $\Sigma(x) : \Gamma(x)$. If $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}\right)(L) \subseteq \mathcal{R}(t)$, then for every $o^\sharp \in [\![L]\!]^\sharp_{T,O}(\Sigma)$, there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$.

Let $o^\sharp \in [\![[]]\!]^\sharp_T(\Sigma) = \{\Sigma(x_o)\}$. Let $\Gamma \in \overline{\Gamma}$. By hypothesis, we have $o^\sharp = \Sigma(x_o) : \Gamma(x_o)$. In addition, $\left(\Gamma\left(x^i_\sigma\right), \Gamma(x_o)\right) \in \mathtt{ty}([])\left(\overline{\Gamma}\right) \subseteq \mathcal{R}(t)$. Let $\tau_\sigma = \Gamma\left(x^i_\sigma\right)$ and $\tau_o = \Gamma(x_o)$, then we have $\sigma^\sharp : \tau_\sigma$, $o^\sharp : \tau_o$, and $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$, as required.

Let us consider the inductive case, proceeding by case on the hypothesis. As O never changes, we do not check it for every inductive case.

Assume that L is $D(x_1, t', x_2) :: L'$, and let $o^\sharp \in [\![L]\!]^\sharp_{T,O}(\Sigma)$. Then there is a $r^\sharp$ such that $(\Sigma(x_1), t', r^\sharp) \in T$ and $o^\sharp \in [\![L']\!]^\sharp_{T,O}\left(\Sigma + x_2 \mapsto r^\sharp\right)$. Let $\Sigma' = \Sigma + x_2 \mapsto r^\sharp$ and $\overline{\Gamma}'_o = \left\{\Gamma + x_2 \mapsto \tau_2 \,\middle|\, \Gamma \in \overline{\Gamma} \wedge (\Gamma(x_1), \tau_2) \in \mathcal{R}(t')\right\}$. For any $\Gamma \in \overline{\Gamma}$, as T is well typed and $\Sigma(x_1) : \Gamma(x_1)$, let τ_Γ the type such that $(\Gamma(x_1), \tau_\Gamma) \in \mathcal{R}(t')$ and $r^\sharp : \tau_\Gamma$. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_\Gamma \,\middle|\, \Gamma \in \overline{\Gamma}\right\}$. We immediately have $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$.

We now check that all conditions are met to apply the inductive hypothesis with L', Σ', and $\overline{\Gamma}'$. From $\mathtt{wf}(dom(\Sigma))(L)$ we deduce $\mathtt{wf}(dom(\Sigma) \uplus \{x_2\})(L') = \mathtt{wf}(dom(\Sigma'))(L')$. For any $\Gamma' \in \overline{\Gamma}'$, $dom(\Gamma') = dom(\Gamma) \uplus \{x_2\} = dom(\Sigma) \uplus \left\{x_2, x^i_\sigma\right\} = dom(\Sigma') \uplus \left\{x^i_\sigma\right\}$. We now fix $\Gamma' \in \overline{\Gamma}'$, hence a $\Gamma \in \overline{\Gamma}$ and a τ_Γ such that both $\Sigma(x_1) : \Gamma(x_1)$ and $r^\sharp : \tau_\Gamma$. Let $x \in dom(\Sigma')$. If $x \neq x_2$, we immediately have $\Sigma'(x) = \Sigma(x) : \Gamma(x) = \Gamma'(x)$ since $x_2 \notin dom(\Sigma) = dom(\Gamma)$. Otherwise, we also immediately have $\Sigma'(x_2) = r^\sharp : \Gamma'(x_2) = \tau_\Gamma$. We also still have $\Gamma'\left(x^i_\sigma\right) \in \mathcal{T}_\sigma$. Finally, since $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we know that $\mathtt{ty}(L')\left(\overline{\Gamma}'_o\right) = \mathtt{ty}(L)\left(\overline{\Gamma}\right)$. As $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$, we have by Lemma 1.1 that $\emptyset \neq \mathtt{ty}(L')\left(\overline{\Gamma}'\right) \subseteq \mathtt{ty}(L')\left(\overline{\Gamma}'_o\right) = \mathtt{ty}(L)\left(\overline{\Gamma}\right) \subseteq \mathcal{R}(t)$. By induction hypothesis, for any $o^\sharp \in [\![L']\!]^\sharp_{T,O}\left(\Sigma + x_2 \mapsto r^\sharp\right)$, there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$.

Assume that L is $P\left(\overrightarrow{x_n}\right) :: L'$, and let $o^\sharp \in [\![L]\!]^\sharp_T(\Sigma)$. We first consider the case where $\mathtt{A}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ does not hold. Then $o^\sharp \in O$, hence there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$, as required.

Otherwise, we have $\mathtt{A}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ and $o^\sharp \in [\![L']\!]^\sharp_{T,O}(\Sigma)$. Let $\overline{\Gamma}' = \left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)\right\}$. We now check we can apply the induction hypothesis with L', Σ, and $\overline{\Gamma}'$. First, $\overline{\Gamma}'$ is not empty because $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ hence for each Γ there is a $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. We also conclude that $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}'\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L) \subseteq \mathcal{R}(t)$. We have $\mathtt{wf}(dom(\Sigma))(L')$ immediately from $\mathtt{wf}(dom(\Sigma))(L)$. Since $\overrightarrow{x_n} \subseteq dom(\Sigma)$, for any $\Gamma' \in \overline{\Gamma}'$ there is a $\Gamma \in \overline{\Gamma}$ and $dom(\Gamma') = dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$. As x^i_σ is distinct from the variables of L, we still have $\Gamma'\left(x^i_\sigma\right) \in \mathcal{T}_\sigma$. Finally, let $\Gamma' \in \overline{\Gamma}'$ and $x \in dom(\Sigma)$. If x is not one of the $\overrightarrow{x_n}$ then we immediately have $\Sigma(x) : \Gamma(x) = \Gamma'(x)$. Otherwise, we need to show that $\Sigma(x_i) : \tau_i = \Gamma'(x_i)$ where $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. Since $\mathtt{A}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ holds and P is well typed, then $\overrightarrow{\Sigma(x_n) : \tau_n}$, as needed. By induction, we deduce that for any $o^\sharp \in [\![L']\!]^\sharp_{T,O}(\Sigma) = [\![L]\!]^\sharp_{T,O}(\Sigma)$ there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$.

Assume that L is $f\left(\overrightarrow{x_n}\right) = x :: L'$ and let $o^\sharp \in [\![L]\!]^\sharp_{T,O}(\Sigma) = [\![L']\!]^\sharp_T(\Sigma')$ where $\Sigma' = \Sigma + x \mapsto \mathtt{A}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$. Let $\overline{\Gamma}' = \left\{\Gamma + x \mapsto \tau \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)\right\}$. We now check we may apply the induction hypothesis with Σ', $\overline{\Gamma}'$, and L'. We first show that $\overline{\Gamma}' \neq \emptyset$. Let $\Gamma \in \overline{\Gamma}$, from $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that there exists some τ such that $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, hence $\overline{\Gamma}'$ is not empty. For every $\Gamma' \in \overline{\Gamma}'$, we have $dom(\Gamma') = dom(\Sigma') \uplus \left\{x^i_\sigma\right\} = dom(\Sigma) \uplus \left\{x, x^i_\sigma\right\}$ and $\Gamma'\left(x^i_\sigma\right) \in \mathcal{T}_\sigma$. From $\mathtt{wf}(dom(\Sigma))(L)$ we immediately deduce $\mathtt{wf}(dom(\Sigma'))(L')$. Let $\Gamma \in \overline{\Gamma}$ and $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, we show that $\mathtt{C}(f)\left(\overrightarrow{\Sigma(x_n)}\right) : \tau$. This is the case since f is well typed. Finally, $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}'\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L) \subseteq \mathcal{R}(t)$, hence by induction for any $o^\sharp \in [\![L']\!]^\sharp_{T,O}(\Sigma') = [\![L]\!]^\sharp_{T,O}(\Sigma)$ there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$.

We now show the thesis. First, let $O = out\left(\sigma^\sharp, t\right)$ and $\overline{\Gamma}_0$ be the set $\left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$. The condition on O is immediately verified. Since $Rule\left(\overrightarrow{cx_{t_n}}\right)$ is well typed, we know that $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}_0\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \subseteq \mathcal{R}(t)$. Let $\overline{\Gamma}$ be $\left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, \tau_\sigma \in \mathcal{T}_\sigma\right\}$. We immediately have $\overline{\Gamma} \subseteq \overline{\Gamma}_0$. Hence by Lemma 1.1, $\emptyset \neq \mathtt{ty}\left(\overline{\Gamma}\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \subseteq \mathcal{R}(t)$. Let $\Sigma = x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}$. As rules are well formed, we have $\mathtt{wf}(dom(\Sigma))\left(Rule\left(\overrightarrow{cx_{t_n}}\right)\right)$, hence $\mathtt{wf}(dom(\Sigma))\left(Rule\left(\overrightarrow{ctn}\right)\right)$ by Lemma 1.3. For any $\Gamma \in \overline{\Gamma}$, we have $dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) \in \mathcal{T}_\sigma$, $\Sigma(x_\sigma) = \Gamma(x_\sigma)$, and $\overrightarrow{\Sigma(x_{t_n}) : s_n = \Gamma(x_{t_n})}$. By the property above, we deduce that for every $o^\sharp \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]^\sharp_{T,O}(\Sigma)$ there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$. □

2.3 Proof of Lemma 3.10

Proof. Let $(\sigma^\sharp, t, o^\sharp) \in F^\sharp(T)$, where $t = \overrightarrow{ct_n}$ closed, $\sigma^\sharp$ such that for every τ_σ, if $\sigma^\sharp : \tau_\sigma$ then $(\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)$. For every rule $Rule\left(\overrightarrow{cx_{t_n}}\right)$, we have $o^\sharp \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]^\sharp_{T,out\left(\sigma^\sharp,t\right)}\left(x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}\right)$. Consider any such rule (there is at least one for constructor c). Since this rule is well typed and

T is well typed, by Lemma 3.8, there exists $(\tau_\sigma, \tau_o) \in \mathcal{R}\left(c\overrightarrow{t_n}\right)$ such that $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$, as required. □

2.4 Proof of Lemma 3.11

Proof. Let $T_1 \subseteq T_2$ two well-typed triple set, and $(\sigma^\sharp, t, o^\sharp) \in F(T_1)$. Then $t = c\overrightarrow{t_n}$ for some closed $\overrightarrow{t_n}$, $\sigma^\sharp$ such that for every τ_σ, if $\sigma^\sharp : \tau_\sigma$ then $(\tau_\sigma, \_) \in \mathcal{R}(t)$, and for every rule $Rule\left(c\overrightarrow{x_{t_n}}\right)$, $o^\sharp \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T_1, out\left(\sigma^\sharp, o\right)}(\Sigma)$ where $\Sigma = x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}$. Let τ_σ such that $\sigma^\sharp : \tau_\sigma$ (every value has a least one type), then $(\tau_\sigma, \_) \in \mathcal{R}(t)$, hence by Lemma 2.5, we have $\left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T_2, out\left(\sigma^\sharp, o\right)}(\Sigma)\downarrow$ for every rule. By Lemma 2.1, we conclude that $o^\sharp \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T_2, out\left(\sigma^\sharp, o\right)}(\Sigma)$ for every rule, hence $o^\sharp \in F^\sharp(T_2)$. □

2.5 Proof of Theorem 3.16

Proof. Let t a closed term such that $t = c\overrightarrow{t_n}$, τ_σ such that $(\tau_\sigma, \_) \in \mathcal{R}(t)$, σ such that $\sigma : \tau_\sigma$, $\sigma^\sharp$ such that for every τ'_σ where $\sigma^\sharp : \tau'_\sigma$, then $(\tau'_\sigma, \_) \in \mathcal{R}(t)$, and $\sigma \in \gamma\left(\sigma^\sharp\right)$. Following Definition 3.13, we assume a k correct set T of triples, that $o^\sharp \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]^\sharp_{T, out\left(\sigma^\sharp, o\right)}\left(x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}\right)$, $o \in \left[\!\left[Rule\left(c\overrightarrow{t_n}\right)\right]\!\right]_{\Downarrow^k}\left(x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}\right)$. We need to prove that $o \in \gamma\left(o^\sharp\right)$. We write $\Sigma \in \gamma\left(\Sigma^\sharp\right)$ for $dom(\Sigma) = dom\left(\Sigma^\sharp\right) \wedge \forall x \in dom(\Sigma)\,.\,\Sigma(x) \in \gamma\left(\Sigma^\sharp(x)\right)$.

We prove the following property by induction on L for an arbitrary O:

$$\begin{aligned}\forall \Sigma, \Sigma^\sharp, \overline{\Gamma}.\ \Sigma \in \gamma\left(\Sigma^\sharp\right) \wedge \mathtt{wf}\left(dom(\Sigma)\right)(L) \wedge \mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset \wedge \\ \left(\forall \Gamma \in \overline{\Gamma}.\ dom(\Gamma) = dom(\Sigma) \uplus \{x_\sigma\}^i \wedge \forall x \in dom(\Sigma)\,.\,\Sigma(x) : \Gamma(x)\right) \wedge \\ o^\sharp \in [\![L]\!]^\sharp_{T,O}\left(\Sigma^\sharp\right) \wedge o \in [\![L]\!]_{\Downarrow^k}(\Sigma) \\ \implies o \in \gamma\left(o^\sharp\right) \quad (1)\end{aligned}$$

Note that the set O does not matter in this proof (see the predicate case below), and that we can always infer from the hypotheses that $\Sigma^\sharp(x) : \Gamma(x)$ for any $x \in dom(\Sigma)$ and $\Gamma \in \overline{\Gamma}$.

The base case holds since $x_o \in dom(\Sigma)$ by definition of $\mathtt{wf}\left(dom(\Sigma)\right)([])$, $[\![[]]\!]_{\Downarrow^k}(\Sigma) = \{\Sigma(x_o)\}$, $[\![[]]\!]^\sharp_T\left(\Sigma^\sharp\right) = \left\{\Sigma^\sharp(x_o)\right\}$ hence $o = \Sigma(x_o)$, $o^\sharp = \Sigma^\sharp(x_o)$, and $o \in \gamma\left(o^\sharp\right)$ as $\Sigma \in \gamma\left(\Sigma^\sharp\right)$.

For the inductive case, we proceed by case on the first hypothesis of L.

If $L = D(x_1, t', x_2) :: L'$, then by $\mathtt{wf}\left(dom(\Sigma)\right)(D(x_1, t', x_2) :: L')$, we have $x_1 \in dom(\Sigma)$ and $\mathtt{wf}\left(dom(\Sigma) \uplus \{x_2\}\right)(L')$. From $o^\sharp \in [\![L]\!]^\sharp_{T,O}\left(\Sigma^\sharp\right)$ and $o \in [\![L]\!]_{\Downarrow^k}(\Sigma)$ we deduce that there exists r and $r^\sharp$ such that $\left(\Sigma^\sharp(x_1), t', r^\sharp\right) \in T$, $\left(\Sigma(x_1), t', r\right) \in \Downarrow^k$, $o^\sharp \in [\![L']\!]^\sharp_{T,O}\left(\Sigma'^\sharp\right)$, and $o \in [\![L']\!]_{\Downarrow^k}(\Sigma')$ where $\Sigma'^\sharp = \Sigma^\sharp + x_2 \mapsto r^\sharp$ and $\Sigma' = \Sigma + x_2 \mapsto r$. As T is k correct, we deduce that $o' \in \gamma\left(o'^\sharp\right)$. Let $\overline{\Gamma}'_o = \left\{\Gamma + x_2 \mapsto \tau_2 \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\Gamma(x_1), \tau_2\right) \in \mathcal{R}(t')\right\}$. For any $\Gamma \in \overline{\Gamma}$, as T is well typed and $\Sigma(x_1) : \Gamma(x_1)$, let τ_Γ the type such that $\left(\Gamma(x_1), \tau_\Gamma\right) \in \mathcal{R}(t')$ and $r : \tau_\Gamma$. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_\Gamma \,\middle|\, \Gamma \in \overline{\Gamma}\right\}$. We immediately have $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$.

We now check that all conditions are met to apply the inductive hypothesis with L', Σ', $\Sigma'^\sharp$, and $\overline{\Gamma}'$. From $\mathtt{wf}\left(dom(\Sigma)\right)(L)$ we deduce $\mathtt{wf}\left(dom(\Sigma) \uplus \{x_2\}\right)(L') = \mathtt{wf}\left(dom(\Sigma')\right)(L')$. We also have

$dom(\Sigma') = dom\left(\Sigma'^{\sharp}\right)$, and for any $x \in dom(\Sigma')$, $\Sigma'(x) \in \gamma\left(\Sigma^{\sharp\prime}(x)\right)$ (the only new case to check is for r). For any $\Gamma' \in \overline{\Gamma}'$, $dom(\Gamma') = dom(\Gamma) \uplus \{x_2\} = dom(\Sigma) \uplus \left\{x_2, x_\sigma^i\right\} = dom(\Sigma') \uplus \left\{x_\sigma^i\right\}$. We now fix $\Gamma' \in \overline{\Gamma}'$, hence a $\Gamma \in \overline{\Gamma}$ and a τ_Γ such that both $\Sigma(x_1) : \Gamma(x_1)$ and $r^\sharp : \tau_\Gamma$. Let $x \in dom(\Sigma')$. If $x \neq x_2$, we immediately have $\Sigma'(x) = \Sigma(x) : \Gamma(x) = \Gamma'(x)$ since $x_2 \notin dom(\Sigma) = dom(\Gamma)$. Otherwise, we also immediately have $\Sigma'(x_2) = r^\sharp : \Gamma'(x_2) = \tau_\Gamma$. Finally, since $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we know that $\mathtt{ty}\left(\overline{\Gamma}'_o\right)(L') = \mathtt{ty}\left(\overline{\Gamma}\right)(L)$. As $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$, we have by Lemma 1.1 that $\mathtt{ty}\left(\overline{\Gamma}'\right)(L') \neq \emptyset$. By induction hypothesis, we obtain $o \in \gamma\left(o^\sharp\right)$.

If $L = P(\overrightarrow{x_n}) :: L'$, then by $\mathtt{wf}(dom(\Sigma))\left(P(\overrightarrow{x_n}) :: L'\right)$, we have $x_i \in dom(\Sigma)$ for every i and $\mathtt{wf}(dom(\Sigma))(L')$. From $o \in [\![L]\!]_{\Downarrow^k}(\Sigma)$ we deduce that $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ and $o \in [\![L']\!]_{\Downarrow^k}(\Sigma)$. Let $\overline{\Gamma}'$ be $\left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)\right\}$. From $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that for every $\Gamma \in \overline{\Gamma}$, there is a $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. Hence $\overline{\Gamma}'$ is not empty, and for any such pair of tuples of types, we have $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$ hence $\overrightarrow{\Sigma^\sharp(x_n) : \Gamma(x_n)}$. As $\mathtt{A}(P)$ is a correct abstraction of $\mathtt{C}(P)$ and $\Sigma \in \gamma\left(\Sigma^\sharp\right)$, the predicate $\mathtt{A}(P)\left(\overrightarrow{v_n^\sharp}\right)$ thus holds. Hence $o^\sharp \in [\![L']\!]^\sharp_{T,O}\left(\Sigma^\sharp\right)$. Most hypothesis are immediate to apply the induction hypothesis, with one exception: showing that for all $\Gamma' \in \overline{\Gamma}'$ and $x \in dom(\Sigma)$, $\Sigma(x) : \Gamma'(x)$ for the x in $\overrightarrow{x_n}$. This is the case since $\mathtt{C}(P)\left(\overrightarrow{v_n}\right)$ holds and the concrete interpretation is consistent, hence $\overrightarrow{v_n : \tau_n}$ for any $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. We can then apply the induction hypothesis with L', Σ, $\Sigma^\sharp$, and $\overline{\Gamma}'$, and we obtain $o \in \gamma\left(o^\sharp\right)$.

If $L = f(\overrightarrow{x_n}) = x :: L'$, then by $\mathtt{wf}(dom(\Sigma))\left(f(\overrightarrow{x_n}) = x :: L'\right)$, we have $x_i \in dom(\Sigma)$ for every i and $\mathtt{wf}(dom(\Sigma) \uplus \{x\})(L')$. From $o \in [\![L]\!]_{\Downarrow^k}(\Sigma)$ we deduce that $o \in [\![L']\!]_{\Downarrow^k}(\Sigma')$ where $\Sigma' = \Sigma + x \mapsto \mathtt{C}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$. From $o^\sharp \in [\![L]\!]^\sharp_{T,O}\left(\Sigma^\sharp\right)$ we deduce that $o^\sharp \in [\![L']\!]^\sharp_{T,O}\left(\Sigma'^\sharp\right)$ where $\Sigma'^\sharp = \Sigma^\sharp + x \mapsto \mathtt{A}(f)\left(\overrightarrow{\Sigma^\sharp(x_n)}\right)$. Let $\overline{\Gamma}' = \left\{\Gamma + x \mapsto \tau \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)\right\}$. From $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ we deduce $\overline{\Gamma}' \neq \emptyset$. For any $\Gamma \in \overline{\Gamma}$, $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$ and $\overrightarrow{\Sigma^\sharp(x_n) : \Gamma(x_n)}$. By Definition 3.4, we thus have $\mathtt{C}(f)\left(\overrightarrow{v_n}\right) \in \gamma\left(\mathtt{A}(f)\left(\overrightarrow{v_n^\sharp}\right)\right)$, hence $\Sigma' \in \gamma\left(\Sigma'^\sharp\right)$. In addition, as the concrete interpretation is consistent, we have $\Sigma'(x) = \mathtt{C}(f)\left(\overrightarrow{\Sigma(x_n)}\right) : \tau$ for any τ such that $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$. We may apply the induction hypothesis with L', Σ', $\Sigma'^\sharp$, and $\overline{\Gamma}'$, and deduce that $o \in \gamma\left(o^\sharp\right)$.

We conclude the proof by applying Property (1) to $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$, $\Sigma^\sharp = x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}$, $\overline{\Gamma} = \left\{x_\sigma^i \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$ and $L = Rule\left(\overrightarrow{ct_n}\right)$, as $\mathtt{wf}(dom(\Sigma))(L)$ (the rule is well formed using Lemma 1.3), $\sigma \in \gamma\left(\sigma^\sharp\right)$, $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ (the rule is well typed), and for every $\Gamma \in \overline{\Gamma}$ and $x \in dom(\Sigma)$ we have $\Sigma(x) : \Gamma(x)$. □

3 PROOFS OF SECTION 5

Lemma 3.1. *Let $\overline{\Gamma}$ and L such that $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$. If $\odot_{\tau_v}\left(\overline{\Gamma}\right)\downarrow$, then $\mathtt{ty}\left(\odot_{\tau_v}\left(\overline{\Gamma}\right)\right)(L) = \odot_{\tau_v}\left(\mathtt{ty}\left(\overline{\Gamma}\right)(L)\right)$.*

Proof. We proceed by induction on L. The base case is immediate as $\odot_{\tau_v}\left(\overline{\Gamma}\right)\downarrow$.

Let $L = D(x_1, t', x_2) :: L'$. From $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ we deduce that for every $\Gamma \in \overline{\Gamma}$, there is a τ_2 such that $(\Gamma(x_1), \tau_2) \in \mathcal{R}(t')$. Hence we also have $(\odot_{\tau_v}(\Gamma(x_1)), \odot_{\tau_v}(\tau_2)) \in \mathcal{R}(t')$, and it holds for any such τ_2. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_2 \,\middle|\, \Gamma \in \overline{\Gamma} \wedge (\Gamma x_1, \tau_2) \in \mathcal{R}(t)\right\}$. We thus have $\odot_{\tau_v}\left(\overline{\Gamma}'\right) \downarrow$ and $\mathtt{ty}\left(\odot_{\tau_v}\left(\overline{\Gamma}\right)\right)(L) = \mathtt{ty}\left(\odot_{\tau_v}\left(\overline{\Gamma}'\right)\right)(L')$. By induction, we have $\mathtt{ty}\left(\odot_{\tau_v}\left(\overline{\Gamma}'\right)\right)(L') = \odot_{\tau_v}\left(\mathtt{ty}\left(\overline{\Gamma}'\right)(L')\right) = \odot_{\tau_v}\left(\mathtt{ty}\left(\overline{\Gamma}\right)(L)\right)$, as needed.

The other cases are similar, relying respectively on the consistency of predicates and functions. □

3.1 Proof of Lemma 5.6

Proof. let $T' \subseteq NS_{\text{WEAKEN}}(T)$, and $\left(\sigma'^\sharp, t, o'^\sharp\right) \in T'$. Then there is $\left(\sigma^\sharp, t, o^\sharp\right) \in T$ such that $\sigma'^\sharp \leq \sigma^\sharp$ and $o^\sharp \leq o'^\sharp$. As T' is $\odot_{\tau_v}$ compatible, for any $\sigma'^\sharp : \tau_\sigma$ and $o'^\sharp : \tau_o$, we have $\odot_{\tau_v}(\tau_\sigma) \downarrow$ and $\odot_{\tau_v}(\tau_o) \downarrow$. Let τ_o such that $o^\sharp : \tau_o$. We then have $o'^\sharp : \tau_o$, hence $\odot_{\tau_v}(\tau_o) \downarrow$. Let $\sigma'^\sharp : \tau_{\sigma'}$, then $\odot_{\tau_v}(\tau_{\sigma'}) \downarrow$ and $\sigma^\sharp : \tau_{\sigma'}$. As abstract values are stratified, we have $\odot_{\tau_v}(\tau_\sigma) \downarrow$ for every $\sigma^\sharp : \tau_\sigma$. Let T'' be the biggest subset of T that is $\odot_{\tau_v}$ compatible. Then we have $\left(\sigma^\sharp, t, o^\sharp\right) \in T''$. We immediately conclude that $\left(\sigma'^\sharp, t, o'^\sharp\right) \in NS_{\text{WEAKEN}}(T'')$.

Let $T' \subseteq NS_{\text{FRAME-PARTITIONING}}(T)$, and $\left(\sigma'^\sharp, t, o'^\sharp\right) \in T'$. Then there are $\overrightarrow{\left(\sigma_n^\sharp, t, o'^\sharp\right) \in T}$ such that $\gamma\left(\sigma'^\sharp\right) \subseteq \bigcup_i \gamma\left(\sigma_i^\sharp\right)$. Let T'' the largest subset of T that is $\odot_{\tau_v}$ compatible. We show that $\left(\sigma'^\sharp, t, o'^\sharp\right) \in NS_{\text{TRACE-PARTITIONING}}(T'')$. Let $\sigma'^\sharp : \tau$, then $\odot_{\tau_v}(\tau) \downarrow$ and there is a $\left(\sigma_i^\sharp, t, o'^\sharp\right) \in T$ such that type $\sigma_i^\sharp : \tau$. As compatibility is immediate for any type $o'^\sharp : \tau_o'$, we have $\left(\sigma_i^\sharp, t, o'^\sharp\right) \in T''$. If $\gamma\left(\sigma'^\sharp\right) = \emptyset$, then we immediately have $\left(\sigma'^\sharp, t, o'^\sharp\right) \in NS_{\text{TRACE-PARTITIONING}}(T'')$ as we need no witness from T. Otherwise, for any $v \in \gamma\left(\sigma'^\sharp\right)$, then $v \in \gamma\left(\sigma_j^\sharp\right)$ for some $\left(\sigma_j^\sharp, t, o'^\sharp\right) \in T$. Let τ such that $v : \tau$, then $\odot_{\tau_v}(\tau) \downarrow$. We also have $\sigma_j^\sharp : \tau$, and since abstract values are stratified, for any τ' such that $\sigma_j^\sharp : \tau'$, then $\odot_{\tau_v}(\tau') \downarrow$. Hence $\left(\sigma_j^\sharp, t, o'^\sharp\right) \in T''$. As this is true for every $v \in \gamma\left(\sigma'^\sharp\right)$, we have $\left(\sigma'^\sharp, t, o'^\sharp\right) \in NS_{\text{TRACE-PARTITIONING}}(T'')$. □

3.2 Proof of Lemma 5.8

Proof. We first show that $\left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_T(\Sigma) \downarrow$. This is the case by Lemma 2.5 because T is well typed.

We prove each item separately. First, if $\left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_T(\Sigma) = \emptyset$, the result is immediate. Otherwise, let $o \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_T(\Sigma)$. By Lemma 2.6, there exists some τ_o such that $o : \tau_o$ and $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$. Since $\odot_{\tau_v}(\tau_\sigma) \downarrow$, we must have $\odot_{\tau_v}(\tau_o) \downarrow$, hence $o \oplus_v \downarrow$. As this is the case for every o, we conclude that $\left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_T(\Sigma) \oplus_v \downarrow$

For the second item, we show that the conditions to apply Lemma 2.5 are satisfied. By Lemma 5.5, $T \oplus_v$ is well typed. As $\odot_{\tau_v}(\tau_\sigma) \downarrow$, we have $\sigma \oplus_v : \odot_{\tau_v}(\tau_\sigma)$. From $(\tau_\sigma, \_) \in \mathcal{R}(t)$ we deduce $(\odot_{\tau_v}(\tau_\sigma), \_) \in \mathcal{R}(t)$. Finally, $\Sigma \oplus_v = x_\sigma \mapsto \sigma \oplus_v + \overrightarrow{x_{t_n} \mapsto t_n}$, as required. Hence $\left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{T \oplus_v}(\Sigma \oplus_v) \downarrow$.

We prove the following property by induction on L, given a $(\tau_\sigma, \_) \in \mathcal{R}(t)$. Let Σ and $\overline{\Gamma}$ such that $\overline{\Gamma} \neq \emptyset$, $\odot_{\tau_v}\left(\overline{\Gamma}\right) \downarrow$, $\mathtt{wf}(dom(\Sigma))(L)$, for every $\Gamma \in \overline{\Gamma}$, $dom(\Gamma) = dom(\Sigma) \uplus \left\{x_\sigma^i\right\}$, $\Gamma\left(x_\sigma^i\right) = \tau_\sigma$, and for every $x \in dom(\Sigma)$, $\Sigma(x) : \Gamma(x)$, and $\Sigma \oplus_v \downarrow$. If $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, then $[\![L]\!]_T(\Sigma) \oplus_v = [\![L]\!]_{T \oplus_v}(\Sigma \oplus_v)$.

For the base case, we have $[\![[]]\!]_T(\Sigma) \oplus_v = \{\Sigma(x_o) \oplus_v\} = [\![L]\!]_{T \oplus_v}(\Sigma \oplus_v)$.

Let us consider the inductive case, proceeding by case on the hypothesis. Each time we prove both inclusions.

Assume that L is $D(x_1, t', x_2) :: L'$, and let $o' \in [\![L]\!]_T(\Sigma) \oplus_v$. Then there is some $o \in [\![L]\!]_T(\Sigma)$ such that $o' = o \oplus_v$. Then there is a r such that $(\Sigma(x_1), t', r) \in T$ and $o \in [\![L']\!]_T(\Sigma + x_2 \mapsto r)$. Conversely, let $o' \in [\![L]\!]_{T\oplus_v}(\Sigma \oplus_v)$. Then there is a r' such that $(\Sigma(x_1) \oplus_v, t', r') \in T \oplus_v$ and $o' \in [\![L']\!]_T(\Sigma \oplus_v + x_2 \mapsto r')$. We know that $r' = r \oplus_v$ and $(\Sigma(x_1), t', r) \in T$.

Let $\Sigma' = \Sigma + x_2 \mapsto r$ and $\overline{\Gamma}'_o = \left\{\Gamma + x_2 \mapsto \tau_2 \,\middle|\, \Gamma \in \overline{\Gamma} \wedge (\Gamma(x_1), \tau_2) \in \mathcal{R}(t')\right\}$. For any $\Gamma \in \overline{\Gamma}$, as T is well typed and $\Sigma(x_1) : \Gamma(x_1)$, let τ_Γ a type such that $(\Gamma(x_1), \tau_\Gamma) \in \mathcal{R}(t')$ and $r : \tau_\Gamma$. From $\odot_{\tau_v}(\Gamma) \downarrow$ and consistency of $\mathcal{R}(t')$, we deduce $\odot_{\tau_v}(\tau_\Gamma) \downarrow$, $r \oplus_v \downarrow$, and $r \oplus_v : \odot_{\tau_v}(\tau_\Gamma)$. In addition we have $\Sigma' \oplus_v \downarrow$. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_\Gamma \,\middle|\, \Gamma \in \overline{\Gamma}\right\}$. We immediately have $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$ and $\odot_{\tau_v}\left(\overline{\Gamma}'\right) \downarrow$. We now check that all conditions are met to apply the inductive hypothesis with L', Σ', and $\overline{\Gamma}'$. From $\mathtt{wf}(dom(\Sigma))(L)$ we deduce $\mathtt{wf}(dom(\Sigma) \uplus \{x_2\})(L') = \mathtt{wf}(dom(\Sigma'))(L')$. For any $\Gamma' \in \overline{\Gamma}'$, $dom(\Gamma') = dom(\Gamma) \uplus \{x_2\} = dom(\Sigma) \uplus \left\{x_2, x^i_\sigma\right\} = dom(\Sigma') \uplus \left\{x^i_\sigma\right\}$. We now fix $\Gamma' \in \overline{\Gamma}'$, hence a $\Gamma \in \overline{\Gamma}$ and a τ_Γ such that both $\Sigma(x_1) : \Gamma(x_1)$ and $r : \tau_\Gamma$. Let $x \in dom(\Sigma')$. If $x \neq x_2$, we immediately have $\Sigma'(x) = \Sigma(x) : \Gamma(x) = \Gamma'(x)$ since $x_2 \notin dom(\Sigma) = dom(\Gamma)$. Otherwise, we also immediately have $\Sigma'(x_2) = r : \Gamma'(x_2) = \tau_\Gamma$. We also still have $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. Finally, since $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we know that $\mathtt{ty}(L')\left(\overline{\Gamma}'_o\right) = \mathtt{ty}(L)\left(\overline{\Gamma}\right)$. As $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$, we have by Lemma 1.1 that $\mathtt{ty}(L')\left(\overline{\Gamma}'\right) \neq \emptyset$. By induction hypothesis, we have $[\![L']\!]_T(\Sigma') \oplus_v = [\![L']\!]_{T\oplus_v}(\Sigma' \oplus_v)$.

For the first inclusion, we have $o' = o \oplus_v \in [\![L']\!]_{T\oplus_v}(\Sigma' \oplus_v)$. Since $(\Sigma(x_1) \oplus_v, t', r \oplus_v) \in T \oplus_v$, we conclude that $o \in [\![L]\!]_{T\oplus_v}(\Sigma \oplus_v)$. For the reverse inclusion, we then have $o' \oplus_v \in [\![L']\!]_T(\Sigma') \oplus_v$ hence there is some $o \in [\![L']\!]_T(\Sigma')$ such that $o' = o \oplus_v$. Since $(\Sigma(x_1), t', r) \in T$, we conclude that $o' \in [\![L]\!]_T(\Sigma) \oplus_v$.

Assume that L is $P\left(\overrightarrow{x_n}\right) :: L'$. We first check we can apply the induction hypothesis with L', Σ, and $\overline{\Gamma}' = \left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)\right\}$. First, $\overline{\Gamma}'$ is not empty because $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ hence for each Γ there is a $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. We also conclude that $\mathtt{ty}\left(\overline{\Gamma}'\right)(L') \neq \emptyset$. We have $\mathtt{wf}(dom(\Sigma))(L')$ immediately from $\mathtt{wf}(dom(\Sigma))(L)$. Since $\overrightarrow{x_n} \subseteq dom(\Sigma)$, for any $\Gamma' \in \overline{\Gamma}'$ there is a $\Gamma \in \overline{\Gamma}$ and $dom(\Gamma') = dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$. As x^i_σ is distinct from the variables of L, we still have $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. Finally, let $\Gamma' \in \overline{\Gamma}'$ and $x \in dom(\Sigma)$. If x is not one of the $\overrightarrow{x_n}$ then we immediately have $\Sigma(x) : \Gamma(x) = \Gamma'(x)$. Otherwise, we need to show that $\Sigma(x_i) : \tau_i = \Gamma'(x_i)$ where $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. Since $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ holds and P is well typed, then $\overrightarrow{\Sigma(x_n) : \tau_n}$, as needed. We now show that $\odot_{\tau_v}(\Gamma') \downarrow$. We only need to check this for $\overrightarrow{\Gamma'(x_n)} = \overrightarrow{\tau_n}$. This is the case because predicate P is consistent with type transformers (Definition 5.1). We may thus apply the induction hypothesis and deduce that $[\![L']\!]_T(\Sigma) \oplus_v = [\![L']\!]_{T\oplus_v}(\Sigma \oplus_v)$.

For any $\Gamma \in \overline{\Gamma}$, we have $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$ (by hypothesis) and $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau'_n}\right)$ for some $\overrightarrow{\tau'_n}$ (by $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$). As P is framing consistent, and since $\overrightarrow{\odot_{\tau_v}(\Gamma(x_n)) \downarrow}$, we then have $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n)}\right) \iff \mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n) \oplus_v}\right)$.

If $o' \in [\![L]\!]_T(\Sigma) \oplus_v$, then $o' = o \oplus_v$ with $o \in [\![L]\!]_T(\Sigma)$. We then must have $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n)}\right)$ and $o \in [\![L']\!]_T(\Sigma)$ By induction hypothesis and the property above, we deduce that $o' \in [\![L']\!]_{T\oplus_v}(\Sigma' \oplus_v)$. Conversely, let $o' \in [\![L']\!]_{T\oplus_v}(\Sigma' \oplus_v)$, then $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n) \oplus_v}\right)$ and $o' \in [\![L']\!]_{T\oplus_v}(\Sigma \oplus_v)$. By the induction hypothesis and the property above, we deduce that $o' \in [\![L]\!]_T(\Sigma) \oplus_v$.

Assume that L is $f\left(\overrightarrow{x_n}\right) = x :: L'$. For any $\Gamma \in \overline{\Gamma}$, we have $\overrightarrow{\Sigma\left(x_n\right) : \Gamma\left(x_n\right)}$ (by hypothesis) and $\left(\overrightarrow{\Gamma\left(x_n\right)}, \tau\right)$ for some τ (by $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$). As f is framing consistent, and since $\overrightarrow{\odot_{\tau_v}\left(\Gamma\left(x_n\right)\right)}\downarrow$, we then have $\mathtt{C}(f)\left(\overrightarrow{\Sigma\left(x_n\right)}\right)\oplus_v = \mathtt{C}(f)\left(\overrightarrow{\Sigma\left(x_n\right)\oplus_v}\right)$.

If $o' \in [\![L]\!]_T(\Sigma)\oplus_v$, then $o' = o\oplus_v$ with $o \in [\![L']\!]_T(\Sigma')$ where $\Sigma' = \Sigma + x \mapsto \mathtt{C}(f)\left(\overrightarrow{\Sigma\left(x_n\right)}\right)$. Conversely, if $o' \in [\![L]\!]_{T\oplus_v}(\Sigma\oplus_v)$, then $o' \in [\![L']\!]_{T\oplus_v}(\Sigma'')$ where $\Sigma'' = \Sigma\oplus_v + x \mapsto \mathtt{C}(f)\left(\overrightarrow{\Sigma\left(x_n\right)\oplus_v}\right)$. By the property above, we have $\Sigma'' = \Sigma'\oplus_v$.

Let $\overline{\Gamma}' = \left\{\Gamma + x \mapsto \tau \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma\left(x_n\right)}, \tau\right) \in \mathcal{R}(f)\right\}$. We now check we may apply the induction hypothesis with Σ', $\overline{\Gamma}'$, and L'. We first show that $\overline{\Gamma}' \neq \emptyset$. Let $\Gamma \in \overline{\Gamma}$, from $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that there exists some τ such that $\left(\overrightarrow{\Gamma\left(x_n\right)}, \tau\right) \in \mathcal{R}(f)$, hence $\overline{\Gamma}'$ is not empty. For every $\Gamma' \in \overline{\Gamma}'$, we have $dom(\Gamma') = dom(\Sigma') \uplus \left\{x_\sigma^i\right\} = dom(\Sigma) \uplus \left\{x, x_\sigma^i\right\}$ and $\Gamma'\left(x_\sigma^i\right) = \tau_\sigma$. From $\mathtt{wf}(dom(\Sigma))(L)$ we immediately deduce $\mathtt{wf}(dom(\Sigma'))(L')$. Let $\Gamma \in \overline{\Gamma}$ and $\left(\overrightarrow{\Gamma\left(x_n\right)}, \tau\right) \in \mathcal{R}(f)$, we show that $\mathtt{C}(f)\left(\overrightarrow{\Sigma\left(x_n\right)}\right) : \tau$. This is the case since f is well typed. We also have $\mathtt{ty}\left(\overline{\Gamma}'\right)(L') \neq \emptyset$. As f is consistent with type transformers, we have $\odot_{\tau_v}(\tau)\downarrow$ for any $\Gamma \in \overline{\Gamma}$ and $\left(\overrightarrow{\Gamma\left(x_n\right)}, \tau\right) \in \mathcal{R}(f)$, hence $\odot_{\tau_v}\left(\overline{\Gamma}'\right)\downarrow$. Finally, as $\mathtt{C}(f)\left(\overrightarrow{\Sigma\left(x_n\right)}\right) : \tau$ and $\odot_{\tau_v}(\tau)\downarrow$, then $\mathtt{C}(f)\left(\overrightarrow{\Sigma\left(x_n\right)}\right)\oplus_v \downarrow$ hence $\Sigma'\oplus_v \downarrow$. By induction, we have $[\![L']\!]_T(\Sigma')\oplus_v = [\![L']\!]_{T\oplus_v}(\Sigma'\oplus_v)$, hence $[\![L]\!]_T(\Sigma)\oplus_v = [\![L]\!]_{T\oplus_v}(\Sigma\oplus_v)$.

We now show the thesis. First, let $\overline{\Gamma}_0 = \left\{x_\sigma^i \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$. Since $Rule\left(\overrightarrow{cx_{t_n}}\right)$ is well typed, we know that $\mathtt{ty}\left(\overline{\Gamma}_0\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. Let $\overline{\Gamma}$ be the singleton $\left\{x_\sigma^i \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n}\right\}$ for the τ_σ under consideration. We immediately have $\overline{\Gamma} \subseteq \overline{\Gamma}_0$. Hence by Lemma 1.1, $\mathtt{ty}\left(\overline{\Gamma}\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. Let $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$. As rules are well formed, we have $\mathtt{wf}(dom(\Sigma))\left(Rule\left(\overrightarrow{ct_n}\right)\right)$. For the single $\Gamma \in \overline{\Gamma}$, we have $dom(\Gamma) = dom(\Sigma) \uplus \left\{x_\sigma^i\right\}$, $\Gamma\left(x_\sigma^i\right) = \tau_\sigma$, $\Sigma(x_\sigma) : \tau_\sigma = \Gamma(x_\sigma)$, and $\overrightarrow{\Sigma\left(x_{t_n}\right) : s_n = \Gamma\left(x_{t_n}\right)}$. As $\odot_{\tau_v}(\tau_\sigma)\downarrow$, and $\odot_{\tau_v}(s)\downarrow$ for any sort, we have $\odot_{\tau_v}\left(\overline{\Gamma}\right)\downarrow$. Finally, as $\sigma : \tau_\sigma$ and $\odot_{\tau_v}(\tau_\sigma)\downarrow$, we have $\sigma\oplus_v \downarrow$ hence $\Sigma\oplus_v \downarrow$ since $t_i\oplus_v \downarrow$ for an term t_i. We may thus use the inductive property to deduce that $\left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_T(\Sigma)\oplus_v = \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{T\oplus_v}(\Sigma\oplus_v)$. □

3.3 Proof of Lemma 5.9

Proof. Note that the lemma does not check if the interpretations are defined. This is because of the following. Let τ_σ a type for σ, hence $\sigma^\sharp : \tau_\sigma$. As $T \subseteq F^\sharp(T)$, we have $(\tau_\sigma, \_) \in \mathcal{R}(t)$. As rules are well typed, by Lemma 2.5, we conclude that their interpretations are defined.

We prove the result by a finite induction on $l \leq k$.

For the base case, we have $\Downarrow^0 = \gamma^0(T)\oplus_v = \emptyset$, so the two sets are trivially equal.

For the inductive case with $l + 1 \leq k$, we do an inner induction of the following property on L, given a $(\tau_\sigma, \_) \in \mathcal{R}(t)$ and an arbitrary set O. Let Σ, $\sigma^\sharp$, and $\overline{\Gamma}$ such that $dom(\Sigma) = dom\left(\Sigma^\sharp\right)$, for every $x \in dom(\Sigma)$ we have $\Sigma(x) \in \gamma\left(\Sigma^\sharp(x)\right)$, $\overline{\Gamma} \neq \emptyset$, $\odot_{\tau_v}\left(\overline{\Gamma}\right)\downarrow$, $\mathtt{wf}(dom(\Sigma))(L)$, for every $\Gamma \in \overline{\Gamma}$,

$dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) = \tau_\sigma$, and for every $x \in dom(\Sigma)$, $\Sigma(x) : \Gamma(x)$, and $\Sigma \oplus_v \downarrow$. If $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, $[\![L]\!]^\sharp_{T,O}\left(\Sigma^\sharp\right) \neq \emptyset$, and $o' \in [\![L]\!]_{\Downarrow^{l+1}}(\Sigma \oplus_v)$, then $o' \in [\![L]\!]_{\gamma^{l+1}(T) \oplus_v}(\Sigma \oplus_v)$.

The base case is immediate as $[\![[]]\!]_{\Downarrow^{l+1}}(\Sigma \oplus_v) = [\![[]]\!]_{\gamma^{l+1}(T)}(\Sigma \oplus_v) = \Sigma(x_o)$.

Assume that L is $D(x_1, t', x_2) :: L'$ and let $o' \in [\![L]\!]_{\Downarrow^{l+1}}(\Sigma \oplus_v)$. Hence there is a $(\Sigma(x_1) \oplus_v, t', r') \in \Downarrow^{l+1}$ such that $o' \in [\![L']\!]_{\Downarrow^{l+1}}(\Sigma \oplus_v + x_1 \mapsto r')$. Hence there is a rule $Rule\left(c'\overrightarrow{y_{t_m}}\right)$ such that $t' = c'\overrightarrow{t'_m}$ and $r' \in \left[\!\left[Rule\left(c'\overrightarrow{t_m}\right)\right]\!\right]_{\Downarrow^l}(\Sigma_{t'} \oplus_v)$, where $\Sigma_{t'} = x_\sigma \mapsto \Sigma(x_1) + \overrightarrow{y_m \mapsto t'_m}$. To apply the outer induction hypothesis on l, we need to find a $\left(v_1^\sharp, t', r^\sharp\right) \in T$ such that $\Sigma(x_1) \in \gamma\left(v_1^\sharp\right)$. As $[\![L]\!]^\sharp_{T,O}\left(\Sigma^\sharp\right) \neq \emptyset$, there exists some $\left(\Sigma^\sharp(x_1), t', r^\sharp\right) \in T$ such that $[\![L']\!]^\sharp_{T,O}\left(\Sigma^\sharp + x_2 \mapsto r^\sharp\right) \neq \emptyset$. Let $v_1^\sharp = \Sigma^\sharp(x_1)$. We have $\Sigma(x_1) \in \gamma\left(v_1^\sharp\right)$ and we may apply the outer inductive hypothesis with l. We thus have $r' \in \left[\!\left[Rule\left(c'\overrightarrow{t_m}\right)\right]\!\right]_{\gamma^l(T) \oplus_v}(\Sigma_{t'} \oplus_v)$. Let $\Gamma \in \overline{\Gamma}$, we have $\Sigma(x_1) : \Gamma(x_1)$. As $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we have $(\Gamma(x_1), \_) \in \mathcal{R}(t')$. Since $\odot_{\tau_v}\left(\overline{\Gamma}\right) \downarrow$, we have $\odot_{\tau_v}(\Gamma(x_1)) \downarrow$. Finally, $\gamma^l(T)$ is compatible with $\odot_{\tau_v}$. We may apply Lemma 5.8 and deduce there is some r such that $r' = r \oplus_v$ and $r \in \left[\!\left[Rule\left(c'\overrightarrow{t_m}\right)\right]\!\right]_{\gamma^l(T)}(\Sigma_{t'})$. As $\gamma^l(T) \subseteq \Downarrow^l$, we thus have $\Sigma(x_1), t' \Downarrow^{l+1} r$. As T is k correct, $l + 1 \leq k$, $\left(v_1^\sharp, t', r^\sharp\right) \in T$, and $\Sigma(x_1) \in \gamma\left(v_1^\sharp\right)$, we have $r \in \gamma\left(r^\sharp\right)$. Let $\Sigma' = \Sigma + x_2 \mapsto r$ and $\Sigma'^\sharp = \Sigma^\sharp + x_2 \mapsto r^\sharp$. We have $dom(\Sigma') = dom\left(\Sigma'^\sharp\right)$, $\Sigma' \in \gamma\left(\Sigma'^\sharp\right)$, and $\Sigma' \oplus_v \downarrow$.

Let $\overline{\Gamma}'_o = \left\{\Gamma + x_2 \mapsto \tau_2 \middle| \Gamma \in \overline{\Gamma} \wedge (\Gamma(x_1), \tau_2) \in \mathcal{R}(t')\right\}$. For any $\Gamma \in \overline{\Gamma}$, as $\Downarrow^{l+1}$ is well typed (Lemma 2.9) and $\Sigma(x_1) : \Gamma(x_1)$, let τ_Γ a type such that $(\Gamma(x_1), \tau_\Gamma) \in \mathcal{R}(t')$ and $r : \tau_\Gamma$. From $\odot_{\tau_v}(\Gamma) \downarrow$ and consistency of $\mathcal{R}(t')$, we deduce $\odot_{\tau_v}(\tau_\Gamma) \downarrow$. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_\Gamma \middle| \Gamma \in \overline{\Gamma}\right\}$. We immediately have $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$ and $\odot_{\tau_v}\left(\overline{\Gamma}'\right) \downarrow$. We now check that all conditions are met to apply the inductive hypothesis with L', Σ', and $\overline{\Gamma}'$. From $\mathtt{wf}(dom(\Sigma))(L)$ we deduce $\mathtt{wf}(dom(\Sigma) \uplus \{x_2\})(L') = \mathtt{wf}(dom(\Sigma'))(L')$. For any $\Gamma' \in \overline{\Gamma}'$, $dom(\Gamma') = dom(\Gamma) \uplus \{x_2\} = dom(\Sigma) \uplus \left\{x_2, x^i_\sigma\right\} = dom(\Sigma') \uplus \left\{x^i_\sigma\right\}$. We now fix $\Gamma' \in \overline{\Gamma}'$, hence a $\Gamma \in \overline{\Gamma}$ and a τ_Γ such that both $\Sigma(x_1) : \Gamma(x_1)$ and $r : \tau_\Gamma$. Let $x \in dom(\Sigma')$. If $x \neq x_2$, we immediately have $\Sigma'(x) = \Sigma(x) : \Gamma(x) = \Gamma'(x)$ since $x_2 \notin dom(\Sigma) = dom(\Gamma)$. Otherwise, we also immediately have $\Sigma'(x_2) = r : \Gamma'(x_2) = \tau_\Gamma$. We also still have $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. Finally, since $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we know that $\mathtt{ty}(L')\left(\overline{\Gamma}'_o\right) = \mathtt{ty}(L)\left(\overline{\Gamma}\right)$. As $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$, we have by Lemma 1.1 that $\mathtt{ty}(L')\left(\overline{\Gamma}'\right) \neq \emptyset$. By induction hypothesis we have $o' \in [\![L']\!]_{\gamma^{l+1}(T)}(\Sigma' \oplus_v)$. We finally show that $(\Sigma(x_1), t', r) \in \gamma^{l+1}(T)$. As show above, we have $\Sigma(x_1), t' \Downarrow^{l+1} r$. In addition we have $\left(\Sigma^\sharp(x_1), t', r^\sharp\right) \in T$ and $\Sigma(x_1) \in \gamma\left(\Sigma^\sharp(x_1)\right)$. Hence $r' = r \oplus_v \in \gamma^{l+1}(T) \oplus_v$, and $o' \in [\![L]\!]_{\gamma^{l+1} \oplus_v}(\Sigma \oplus_v)$ as needed.

Assume that L is $P\left(\overrightarrow{x_n}\right) :: L'$ and let $o' \in [\![L]\!]_{\Downarrow^{l+1}}(\Sigma \oplus_v)$. Then the predicate $\mathtt{C}(P)\left(\overrightarrow{\Sigma(x_n) \oplus_v}\right)$ holds and we have $o' \in [\![L']\!]_{\Downarrow^{l+1}}(\Sigma \oplus_v)$. Let $\overline{\Gamma}'$ be $\left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \middle| \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)\right\}$. From $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that for every $\Gamma \in \overline{\Gamma}$, there is a $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. Hence $\overline{\Gamma}'$ is not empty, and for any such pair of tuples of types, we have $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$ hence $\overrightarrow{\Sigma^\sharp(x_n) : \Gamma(x_n)}$. As $\mathtt{A}(P)$ is a correct abstraction of $\mathtt{C}(P)$ and $\Sigma \in \gamma\left(\Sigma^\sharp\right)$, the predicate $\mathtt{A}(P)\left(\overrightarrow{v_n^\sharp}\right)$ thus holds. Hence $[\![L']\!]^\sharp_{T,O}\left(\Sigma^\sharp\right) = [\![L]\!]^\sharp_{T,O}\left(\Sigma^\sharp\right) \neq \emptyset$. Most hypothesis are immediate to apply the induction hypothesis,

with two exceptions: showing that for all $\Gamma' \in \overline{\Gamma}'$ and $x \in dom(\Sigma)$, $\Sigma(x) : \Gamma'(x)$ and $\odot_{\tau_v}(\Gamma'(x)) \downarrow$ for the x in $\overrightarrow{x_n}$. Regarding the typing, as $\mathtt{C}(P)\left(\overrightarrow{v_n}\right)$ holds and the concrete interpretation is consistent, we thus have $\overrightarrow{v_n : \tau_n}$ for any $\left(\overrightarrow{\Gamma(x_n)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}(P)$. For type transformers, we simply rely on Definition 5.1. We can then apply the induction hypothesis with L', Σ, $\Sigma^\sharp$, and $\overline{\Gamma}'$, and we obtain $o' \in [\![L]\!]_{\gamma^{l+1}\oplus_v}(\Sigma\oplus_v)$

Assume that $L = f(\overrightarrow{x_n}) = x :: L'$ and let $o' \in [\![L]\!]_{\Downarrow^{l+1}}(\Sigma\oplus_v)$. We then have $o' \in [\![L']\!]_{\Downarrow^{l+1}}(\Sigma_r)$ where $r' = \mathtt{C}(f)\left(\overrightarrow{\Sigma(x_n)\oplus_v}\right)$ and $\Sigma_r = \oplus_\Sigma + x \mapsto r'$. Let $r = \mathtt{C}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$ As $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, for $\Gamma \in \overline{\Gamma}$ we know that $\left(\overrightarrow{\Gamma(x_n)}, \_\right) \in \mathcal{R}(f)$. Since $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$ and $\odot_{\tau_v}(\Gamma)\downarrow$, then $r' = r\oplus_v$ by framing consistency of functions. Let $\Sigma' = \Sigma + x \mapsto r$, then $\Sigma_r = \Sigma'\oplus_v$. Let $r^\sharp = \mathtt{A}(f)\left(\overrightarrow{\Sigma(x_n)\oplus_v}\right)$. Since $\overrightarrow{\Sigma(x_n) : \Gamma(x_n)}$, we have $\overrightarrow{\Sigma^\sharp(x_n) : \Gamma(x_n)}$. As $\Sigma \in \gamma\left(\Sigma^\sharp\right)$, we deduce from Definition 3.4 that $r \in \gamma\left(r^\sharp\right)$. Let $\Sigma'^\sharp = \Sigma^\sharp + x \mapsto r^\sharp$, we have $\Sigma' \in \gamma\left(\Sigma'^\sharp\right)$. In addition, $[\![L']\!]^\sharp_T\left(\Sigma'^\sharp\right) = [\![L]\!]^\sharp_T\left(\Sigma^\sharp\right) \neq \emptyset$. Let $\overline{\Gamma}' = \left\{\Gamma + x \mapsto \tau \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)\right\}$. From $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ we deduce $\overline{\Gamma}' \neq \emptyset$. As the concrete interpretation is consistent, we have $\Sigma'(x) = r : \tau$ for any τ such that $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$. By Definition 5.1 we have $\odot_{\tau_v}(\tau)$ for any such τ. By induction, we thus have $o' \in [\![L']\!]_{\gamma^{l+1}(T)}(\Sigma'\oplus_v)$, hence $o' \in [\![L]\!]_{\gamma^{l+1}(T)}(\Sigma)$, as needed.

We now show the thesis. Let $\left(\sigma^\sharp, t, o^\sharp\right) \in T$, $\sigma \in \gamma\left(\sigma^\sharp\right)$, and a rule $Rule\left(c\overrightarrow{x_{t_n}}\right)$ such as $t = c\overrightarrow{t_n}$. Let $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_{t_n} \mapsto t_n}$ and $\Sigma^\sharp = x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}$. We immediately have $dom(\Sigma) = dom\left(\Sigma^\sharp\right)$ and $\Sigma \in \gamma\left(\Sigma^\sharp\right)$. Let τ_σ a type for σ, hence $\sigma^\sharp : \tau_\sigma$. As $T \subseteq F^\sharp(T)$, we have $(\tau_\sigma, \_) \in \mathcal{R}(t)$ and $o^\sharp \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]^\sharp_T\left(\Sigma^\sharp\right) \neq \emptyset$. let $\overline{\Gamma}_0 = \left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$. Since $Rule\left(c\overrightarrow{x_{t_n}}\right)$ is well typed, we know that $\mathtt{ty}\left(\overline{\Gamma}_0\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. Let $\overline{\Gamma}$ be the singleton $\left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n}\right\}$. We immediately have $\overline{\Gamma} \subseteq \overline{\Gamma}_0$. Hence by Lemma 1.1, $\mathtt{ty}\left(\overline{\Gamma}\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. As rules are well formed, we have $\mathtt{wf}(dom(\Sigma))\left(Rule\left(\overrightarrow{ct_n}\right)\right)$ by Lemma 1.3. For the single $\Gamma \in \overline{\Gamma}$, we have $dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) = \tau_\sigma$, $\Sigma(x_\sigma) : \tau_\sigma = \Gamma(x_\sigma)$, and $\overrightarrow{\Sigma(x_{t_n}) : s_n = \Gamma(x_{t_n})}$. As $\odot_{\tau_v}(\tau_\sigma)\downarrow$, and $\odot_{\tau_v}(s)\downarrow$ for any sort, we have $\odot_{\tau_v}\left(\overline{\Gamma}\right)\downarrow$. In addition, as $\sigma : \tau_\sigma$ and $\odot_{\tau_v}(\tau_\sigma)\downarrow$, we have $\sigma\oplus_v \downarrow$ hence $\Sigma\oplus_v \downarrow$ since $t_i\oplus_v \downarrow$ for any term t_i. For any $l \leq k$, if $o' \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{\Downarrow^l}(\Sigma\oplus_v)$ then $o' \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{\gamma^l(T)\oplus_v}(\Sigma\oplus_v)$, and we conclude with $l = k$. □

3.4 Proof of Lemma 5.11

Proof. Let $(\tau_\sigma, \_) \in \mathcal{R}(t)$. We first prove that if $o^\sharp : \tau_o$, then $\odot_{\tau_v}(\tau_o)\downarrow$. By Lemma 3.8, we have $(\tau_\sigma, \tau_o) \in \mathcal{R}(t)$. As $\odot_{\tau_v}(\tau_\sigma)\downarrow$ and since type transformers are consistent with types, we then have $\odot_{\tau_v}(\tau_o)\downarrow$.

We next prove the following property, by induction on L, given a set O such that $O\oplus^\sharp_{v^\sharp}\downarrow$. Let Σ and $\overline{\Gamma}$ such that $\overline{\Gamma} \neq \emptyset$, $\odot_{\tau_v}\left(\overline{\Gamma}\right)\downarrow$, $\mathtt{wf}(dom(\Sigma))(L)$, for every $\Gamma \in \overline{\Gamma}$, $dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) = \tau_\sigma$, and for every $x \in dom(\Sigma)$, $\Sigma(x) : \Gamma(x)$, and $\Sigma\oplus_v \downarrow$. If $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$ and $o^\sharp \in [\![L]\!]^\sharp_{T,O}(\Sigma)$, then we have $o^\sharp\oplus^\sharp_{v^\sharp} \in [\![L]\!]^\sharp_{T\oplus^\sharp_{v^\sharp}, O\oplus^\sharp_{v^\sharp}}\left(\Sigma\oplus^\sharp_{v^\sharp}\right)$.

For the base case, we have $o^\sharp = \Sigma\left(x_o\right)$ and $\left[\!\left[L\right]\!\right]^\sharp_{T\oplus^\sharp_{v^\sharp}, O\oplus^\sharp_{v^\sharp}}\left(\Sigma\oplus^\sharp_{v^\sharp}\right) = \left\{\Sigma\left(x_o\right)\oplus^\sharp_{v^\sharp}\right\} \ni o^\sharp\oplus^\sharp_{v^\sharp}$.

Let us consider the inductive case, proceeding by case on the hypothesis.

Assume that L is $D(x_1, t', x_2) :: L'$, and let $o^\sharp \in \left[\!\left[L\right]\!\right]^\sharp_{T,O}\left(\Sigma\right)$. Then there is a $r^\sharp$ such that $\left(\Sigma\left(x_1\right), t', r^\sharp\right) \in T$ and $o \in \left[\!\left[L'\right]\!\right]_{T,O}\left(\Sigma + x_2 \mapsto r^\sharp\right)$. Let $\Sigma' = \Sigma + x_2 \mapsto r$ and $\overline{\Gamma}'_o = \left\{\Gamma + x_2 \mapsto \tau_2 \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\Gamma\left(x_1\right), \tau_2\right) \in \mathcal{R}\left(t'\right)\right\}$. For any $\Gamma \in \overline{\Gamma}$, as T is well typed and $\Sigma\left(x_1\right) : \Gamma\left(x_1\right)$, let τ_Γ a type such that $\left(\Gamma\left(x_1\right), \tau_\Gamma\right) \in \mathcal{R}\left(t'\right)$ and $r^\sharp : \tau_\Gamma$. From $\odot_{\tau_v}\left(\Gamma\right)\downarrow$ and consistency of $\mathcal{R}\left(t'\right)$, we deduce $\odot_{\tau_v}\left(\tau_\Gamma\right)\downarrow$, $r\oplus_v\downarrow$, and $r\oplus_v : \odot_{\tau_v}\left(\tau_\Gamma\right)$. In addition we have $\Sigma'\oplus_v\downarrow$. Let $\overline{\Gamma}' = \left\{\Gamma + x_2 \mapsto \tau_\Gamma \,\middle|\, \Gamma \in \overline{\Gamma}\right\}$. We immediately have $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$ and $\odot_{\tau_v}\left(\overline{\Gamma}'\right)\downarrow$. We now check that all conditions are met to apply the inductive hypothesis with L', Σ', and $\overline{\Gamma}'$. From $\mathtt{wf}\left(dom\left(\Sigma\right)\right)\left(L\right)$ we deduce $\mathtt{wf}\left(dom\left(\Sigma\right)\uplus\left\{x_2\right\}\right)\left(L'\right) = \mathtt{wf}\left(dom\left(\Sigma'\right)\right)\left(L'\right)$. For any $\Gamma' \in \overline{\Gamma}'$, $dom\left(\Gamma'\right) = dom\left(\Gamma\right)\uplus\left\{x_2\right\} = dom\left(\Sigma\right)\uplus\left\{x_2, x^i_\sigma\right\} = dom\left(\Sigma'\right)\uplus\left\{x^i_\sigma\right\}$. We now fix $\Gamma' \in \overline{\Gamma}'$, hence a $\Gamma \in \overline{\Gamma}$ and a τ_Γ such that both $\Sigma\left(x_1\right) : \Gamma\left(x_1\right)$ and $r^\sharp : \tau_\Gamma$. Let $x \in dom\left(\Sigma'\right)$. If $x \neq x_2$, we immediately have $\Sigma'\left(x\right) = \Sigma\left(x\right) : \Gamma\left(x\right) = \Gamma'\left(x\right)$ since $x_2 \notin dom\left(\Sigma\right) = dom\left(\Gamma\right)$. Otherwise, we also immediately have $\Sigma'\left(x_2\right) = r^\sharp : \Gamma'\left(x_2\right) = \tau_\Gamma$. We also still have $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. Finally, since $\mathtt{ty}\left(\overline{\Gamma}\right)\left(L\right) \neq \emptyset$, we know that $\mathtt{ty}\left(L'\right)\left(\overline{\Gamma}'_o\right) = \mathtt{ty}\left(L\right)\left(\overline{\Gamma}\right)$. As $\emptyset \neq \overline{\Gamma}' \subseteq \overline{\Gamma}'_o$, we have by Lemma 1.1 that $\mathtt{ty}\left(L'\right)\left(\overline{\Gamma}'\right) \neq \emptyset$. By induction hypothesis, we have $o^\sharp\oplus^\sharp_{v^\sharp} \in \left[\!\left[L'\right]\!\right]^\sharp_{T\oplus^\sharp_{v^\sharp}, O\oplus^\sharp_{v^\sharp}}\left(\Sigma'\oplus^\sharp_{v^\sharp}\right)$. As $\left(\Sigma\left(x_1\right)\oplus^\sharp_{v^\sharp}, t', r\oplus^\sharp_{v^\sharp}\right) \in T\oplus^\sharp_{v^\sharp}$, we conclude that $o^\sharp\oplus^\sharp_{v^\sharp} \in \left[\!\left[L\right]\!\right]^\sharp_{T\oplus^\sharp_{v^\sharp}, O\oplus^\sharp_{v^\sharp}}\left(\Sigma\oplus^\sharp_{v^\sharp}\right)$.

Assume that L is $P\left(\overrightarrow{x_n}\right) :: L'$. Let $\Gamma \in \overline{\Gamma}$. Since $\overrightarrow{\Sigma\left(x_n\right) : \Gamma\left(x_n\right)}$ and $\mathtt{ty}\left(\overline{\Gamma}\right)\left(L\right) \neq \emptyset$, we know there is $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma\left(x_n\right)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}\left(P\right)$. By Definition 5.1, we have $\overrightarrow{\odot_{\tau_v}\left(\tau_n\right)}\downarrow$ from $\odot_{\tau_v}\left(\Gamma\right)\downarrow$. By Definition 5.2 we have $\mathtt{A}\left(P\right)\left(\overrightarrow{\Sigma\left(x_n\right)}\right) \iff \mathtt{A}\left(P\right)\left(\overrightarrow{\Sigma\left(x_n\right)\oplus_v}\right)$. We distinguish two cases. If the predicate does not hold, then we have $o^\sharp \in O$, hence $o^\sharp\oplus^\sharp_{v^\sharp} \in O\oplus^\sharp_{v^\sharp}$, as needed. Otherwise, We check whether we can apply the induction hypothesis with L', Σ, and $\overline{\Gamma}' = \left\{\Gamma + \overrightarrow{x_n \mapsto \tau_n} \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma\left(x_n\right)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}\left(P\right)\right\}$. First, $\overline{\Gamma}'$ is not empty because $\mathtt{ty}\left(\overline{\Gamma}\right)\left(L\right) \neq \emptyset$ hence for each Γ there is a $\overrightarrow{\tau_n}$ such that $\left(\overrightarrow{\Gamma\left(x_n\right)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}\left(P\right)$. We also conclude that $\mathtt{ty}\left(\overline{\Gamma}'\right)\left(L'\right) \neq \emptyset$. We have $\mathtt{wf}\left(dom\left(\Sigma\right)\right)\left(L'\right)$ immediately from $\mathtt{wf}\left(dom\left(\Sigma\right)\right)\left(L\right)$. Since $\overrightarrow{x_n} \subseteq dom\left(\Sigma\right)$, for any $\Gamma' \in \overline{\Gamma}'$ there is a $\Gamma \in \overline{\Gamma}$ and $dom\left(\Gamma'\right) = dom\left(\Gamma\right) = dom\left(\Sigma\right)\uplus\left\{x^i_\sigma\right\}$. As x^i_σ is distinct from the variables of L, we still have $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. Finally, let $\Gamma' \in \overline{\Gamma}'$ and $x \in dom\left(\Sigma\right)$. If x is not one of the $\overrightarrow{x_n}$ then we immediately have $\Sigma\left(x\right) : \Gamma\left(x\right) = \Gamma'\left(x\right)$. Otherwise, we need to show that $\Sigma\left(x_i\right) : \tau_i = \Gamma'\left(x_i\right)$ where $\left(\overrightarrow{\Gamma\left(x_n\right)}, \overrightarrow{\tau_n}\right) \in \mathcal{R}\left(P\right)$. Since $\mathtt{A}\left(P\right)\left(\overrightarrow{\Sigma\left(x_n\right)}\right)$ holds and P is well typed, then $\overrightarrow{\Sigma\left(x_n\right) : \tau_n}$, as needed. We now show that $\odot_{\tau_v}\left(\Gamma'\right)\downarrow$. We only need to check this for $\overrightarrow{\Gamma'\left(x_n\right)} = \overrightarrow{\tau_n}$, and this has been done above. We may thus apply the induction hypothesis and deduce that $o^\sharp\oplus^\sharp_{v^\sharp} \in \left[\!\left[L\right]\!\right]^\sharp_{T\oplus^\sharp_{v^\sharp}, O\oplus^\sharp_{v^\sharp}}\left(\Sigma\oplus^\sharp_{v^\sharp}\right)$.

Assume that L is $f\left(\overrightarrow{x_n}\right) = x :: L'$. For any $\Gamma \in \overline{\Gamma}$, we have $\overrightarrow{\Sigma\left(x_n\right) : \Gamma\left(x_n\right)}$ (by hypothesis) and $\left(\overrightarrow{\Gamma\left(x_n\right)}, \tau\right)$ for some τ (by $\mathtt{ty}\left(\overline{\Gamma}\right)\left(L\right) \neq \emptyset$). As f is framing consistent, and since $\overrightarrow{\odot_{\tau_v}\left(\Gamma\left(x_n\right)\right)}\downarrow$, we then have $\mathtt{A}\left(f\right)\left(\overrightarrow{\Sigma\left(x_n\right)}\right)\oplus^\sharp_{v^\sharp} = \mathtt{A}\left(f\right)\left(\overrightarrow{\Sigma\left(x_n\right)\oplus^\sharp_{v^\sharp}}\right)$.

If $o \in [\![L]\!]^\sharp_{T,O}(\Sigma)$, then $o \in [\![L']\!]^\sharp_{T,O}(\Sigma')$ where $\Sigma' = \Sigma + x \mapsto \mathtt{A}(f)\left(\overrightarrow{\Sigma(x_n)}\right)$.

Let $\overline{\Gamma}' = \left\{\Gamma + x \mapsto \tau \,\middle|\, \Gamma \in \overline{\Gamma} \wedge \left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)\right\}$. We now check we may apply the induction hypothesis with Σ', $\overline{\Gamma}'$, and L'. We first show that $\overline{\Gamma}' \neq \emptyset$. Let $\Gamma \in \overline{\Gamma}$, from $\mathtt{ty}\left(\overline{\Gamma}\right)(L) \neq \emptyset$, we deduce that there exists some τ such that $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, hence $\overline{\Gamma}'$ is not empty. For every $\Gamma' \in \overline{\Gamma}'$, we have $dom(\Gamma') = dom(\Sigma') \uplus \left\{x^i_\sigma\right\} = dom(\Sigma) \uplus \left\{x, x^i_\sigma\right\}$ and $\Gamma'\left(x^i_\sigma\right) = \tau_\sigma$. From $\mathtt{wf}(dom(\Sigma))(L)$ we immediately deduce $\mathtt{wf}(dom(\Sigma'))(L')$. Let $\Gamma \in \overline{\Gamma}$ and $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, we show that $\mathtt{A}(f)\left(\overrightarrow{\Sigma(x_n)}\right) : \tau$. This is the case since f is well typed. We also have $\mathtt{ty}\left(\overline{\Gamma}'\right)(L') \neq \emptyset$. As f is consistent with type transformers, we have $\odot_{\tau_v}(\tau)\downarrow$ for any $\Gamma \in \overline{\Gamma}$ and $\left(\overrightarrow{\Gamma(x_n)}, \tau\right) \in \mathcal{R}(f)$, hence $\odot_{\tau_v}\left(\overline{\Gamma}'\right)\downarrow$. Finally, as $\mathtt{A}(f)\left(\overrightarrow{\Sigma(x_n)}\right) : \tau$ and $\odot_{\tau_v}(\tau)\downarrow$, then $\mathtt{A}(f)\left(\overrightarrow{\Sigma(x_n)}\right) \oplus^\sharp_{v^\sharp}\downarrow$ hence $\Sigma' \oplus^\sharp_{v^\sharp}\downarrow$. By induction, we have $o^\sharp \oplus^\sharp_{v^\sharp} \in [\![L']\!]^\sharp_{T\oplus^\sharp_{v^\sharp}, O\oplus^\sharp_{v^\sharp}}\left(\Sigma' \oplus^\sharp_{v^\sharp}\right) = [\![L]\!]^\sharp_{T\oplus^\sharp_{v^\sharp}, O\oplus^\sharp_{v^\sharp}}\left(\Sigma' \oplus^\sharp_{v^\sharp}\right)$.

We now show the thesis. First, let $\overline{\Gamma}_0 = \left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n} \,\middle|\, (\tau_\sigma, \_) \in \mathcal{R}\left(\overrightarrow{ct_n}\right)\right\}$. Since $Rule\left(\overrightarrow{cx_{t_n}}\right)$ is well typed, we know that $\mathtt{ty}\left(\overline{\Gamma}_0\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. Let $\overline{\Gamma}$ be the singleton $\left\{x^i_\sigma \mapsto \tau_\sigma + x_\sigma \mapsto \tau_\sigma + \overrightarrow{x_{t_n} \mapsto s_n}\right\}$ for the τ_σ under consideration. We immediately have $\overline{\Gamma} \subseteq \overline{\Gamma}_0$. Hence by Lemma 1.1, $\mathtt{ty}\left(\overline{\Gamma}\right)\left(Rule\left(\overrightarrow{ct_n}\right)\right) \neq \emptyset$. Let $\Sigma = x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_{t_n} \mapsto t_n}$. As rules are well formed, we have $\mathtt{wf}(dom(\Sigma))\left(Rule\left(\overrightarrow{ct_n}\right)\right)$ by Lemma 1.3. For the single $\Gamma \in \overline{\Gamma}$, we have $dom(\Gamma) = dom(\Sigma) \uplus \left\{x^i_\sigma\right\}$, $\Gamma\left(x^i_\sigma\right) = \tau_\sigma$, $\Sigma(x_\sigma) : \tau_\sigma = \Gamma(x_\sigma)$, and $\overrightarrow{\Sigma(x_{t_n}) : s_n = \Gamma(x_{t_n})}$. As $\odot_{\tau_v}(\tau_\sigma)\downarrow$, and $\odot_{\tau_v}(s)\downarrow$ for any sort, we have $\odot_{\tau_v}\left(\overline{\Gamma}\right)\downarrow$. Finally, as $\sigma^\sharp : \tau_\sigma$ and $\odot_{\tau_v}(\tau_\sigma)\downarrow$, we have $\sigma^\sharp \oplus^\sharp_{v^\sharp}\downarrow$ hence $\Sigma \oplus^\sharp_{v^\sharp}\downarrow$ since $t_i \oplus_v \downarrow$ for an term t_i. By the inductive property, we have $o^\sharp \oplus^\sharp_{v^\sharp} \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]^\sharp_{T\oplus^\sharp_{v^\sharp}, out\left(\sigma^\sharp \oplus^\sharp_{v^\sharp}, t\right)}\left(x_\sigma \mapsto \sigma^\sharp \oplus^\sharp_{v^\sharp} + \overrightarrow{x_n \mapsto t_n}\right)$. □

3.5 Proof of Theorem 5.13

Proof. First, we assume that $T \subseteq F^\sharp(T)$ is k correct and show that $NS_{\text{FRAME}}\left(T, v^\sharp\right)$ is k correct.

Let $\left(\sigma'^\sharp, t, o'^\sharp\right) \in NS_{\text{FRAME}}\left(T, v^\sharp\right)$ such that $\sigma', t \Downarrow^k o'$, $t = \overrightarrow{ct_n}$ closed, and $\sigma' \in \gamma\left(\sigma'^\sharp\right)$. We want to show that $o' \in \gamma\left(o'^\sharp\right)$. By definition of the frame rule, we have $\sigma'^\sharp = \sigma^\sharp \oplus^\sharp_{v^\sharp}$ and $o'^\sharp = o^\sharp \oplus^\sharp_{v^\sharp}$, with $\left(\sigma^\sharp, t, o^\sharp\right) \in T$.

Since $\sigma', t \Downarrow^k o'$, we have $t = \overrightarrow{ct_n}$ close, there is a τ' such that $\sigma' : \tau'$ and $(\tau', \_) \in \mathcal{R}(t)$, and there is a rule $Rule\left(\overrightarrow{cx_{t_n}}\right)$ such that $o' \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right)\right]\!\right]_{\Downarrow^{k-1}}(\Sigma)$, where $\Sigma' = x_\sigma \mapsto \sigma' + \overrightarrow{x_n \mapsto t_n}$.

As $\sigma' \in \gamma\left(\sigma'^\sharp\right)$, we have $\sigma'^\sharp : \tau'$. From $\oplus^\sharp_{v^\sharp} : \odot_{\tau_v}$ and $\sigma^\sharp \oplus^\sharp_{v^\sharp} : \tau'$, we deduce that there is some τ_σ such that $\sigma^\sharp : \tau_\sigma$ and $\odot_{\tau_v}(\tau_\sigma) = \tau'$. From $\sigma' : \odot_{\tau_v}(\tau_\sigma)$, we deduce that there are some σ, $\oplus_v$, and v' such that $\oplus_v : \odot_{\tau_v}$, $\sigma' = \sigma \oplus_v$, and $\sigma : \tau_\sigma$. As value transformers are consistent and $\sigma' \in \gamma\left(\sigma'^\sharp\right)$, we have $\sigma \in \gamma\left(\sigma^\sharp\right)$ and $v \in \gamma\left(v^\sharp\right)$. We also have $\sigma^\sharp : \tau_\sigma$.

From $\left(\sigma^\sharp, t, o^\sharp\right) \in T \subseteq F^\sharp(T)$ and $\sigma^\sharp : \tau_\sigma$, we deduce that $(\tau_\sigma, \_) \in \mathcal{R}(t)$.

Let $\Sigma = x_\sigma \mapsto \sigma + \overrightarrow{x_n \mapsto t_n}$, we then have $\Sigma' = \Sigma \oplus_v$. By Lemma 5.9, we have $o' \in \left[\!\left[Rule\,(c)\,\overrightarrow{t_n} \right]\!\right]_{\oplus_{\gamma^{k-1}(T)}} (\Sigma \oplus_v)$. As $\gamma^{k-1}(T)$ is compatible with $\odot_{\tau_v}$, we have $o' \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right) \right]\!\right]_{\gamma^{k-1}(T)} (\Sigma) \oplus_v$. Hence $o' = o \oplus_v$ and $o \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right) \right]\!\right]_{\gamma^{k-1}(T)} (\Sigma)$. Since $\gamma^{k-1}(T) \subseteq \Downarrow^{k-1}$, we have by Lemma 1.4 that $o \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right) \right]\!\right]_{\Downarrow^{k-1}} (\Sigma)$ (both are defined as they are well typed). Hence $(\sigma, t, o) \in F\left(\Downarrow^{k-1}\right) = \Downarrow^k$. As T is k correct, we have $o \in \gamma\left(o^\sharp\right)$. As the $\oplus_v$ and $\oplus^\sharp_{v^\sharp}$ are consistent, we have $o' = o \oplus_v \in \gamma\left(o^\sharp \oplus^\sharp_{v^\sharp}\right)$.

Second, we show that if $T' \subseteq F^\sharp(T)$ where T' is $\odot_{\tau_v}$ compatible, then $NS_{\text{FRAME}}\left(T', v^\sharp\right) \subseteq F^\sharp\left(NS_{\text{FRAME}}\left(T, v^\sharp\right)\right)$. Let $\left(\sigma'^\sharp, t, o'^\sharp\right) \in NS_{\text{FRAME}}\left(T', v^\sharp\right)$. We have $\sigma'^\sharp = \sigma^\sharp \oplus^\sharp_{v^\sharp}$, $o'^\sharp = o^\sharp \oplus^\sharp_{v^\sharp}$, and $\left(\sigma^\sharp, t, o^\sharp\right) \in T'$. Assume $t = \overrightarrow{ct_n}$ closed, and let τ' such that $\sigma^\sharp \oplus^\sharp_{v^\sharp} : \tau'$. We first show that $(\tau', \_) \in \mathcal{R}(t)$. By consistency of value and type transformers, there is a τ such that $\tau' = \odot_{\tau_v}(\tau)$ and $\sigma^\sharp : \tau$. As $T' \subseteq F^\sharp(T)$, we have $(\tau, \_) \in \mathcal{R}(t)$. As $\mathcal{R}(t)$ is consistent with type transformers, we thus have $(\tau', \_) \in \mathcal{R}(t)$. For any rule $Rule\left(c\overrightarrow{x_{t_n}}\right)$, as $\left(\sigma^\sharp, t, o^\sharp\right) \in T' \subseteq F^\sharp(T)$, we have $o^\sharp \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right) \right]\!\right]^\sharp_{T, out(\sigma^\sharp, t)} \left(x_\sigma \mapsto \sigma^\sharp + \overrightarrow{x_n \mapsto t_n}\right)$. By Lemma 5.11, we have $o^\sharp \oplus^\sharp_{v^\sharp} \in \left[\!\left[Rule\left(\overrightarrow{ct_n}\right) \right]\!\right]^\sharp_{T \oplus^\sharp_{v^\sharp}, out\left(\sigma^\sharp \oplus^\sharp_{v^\sharp}, t\right)} \left(x_\sigma \mapsto \sigma^\sharp \oplus^\sharp_{v^\sharp} + \overrightarrow{x_n \mapsto t_n}\right)$ As this is true for any rule, we have $\left(\sigma^\sharp \oplus^\sharp_{v^\sharp}, t, o^\sharp \oplus^\sharp_{v^\sharp}\right) \in F^\sharp\left(NS_{\text{FRAME}}\left(T, v^\sharp\right)\right)$.

Third, let $T' \subseteq NS_{\text{WEAKEN}}(T)$, and $\left(\sigma'^\sharp, t, o'^\sharp\right) \in T'$. Then there is $\left(\sigma^\sharp, t, o^\sharp\right) \in T$ such that $\sigma'^\sharp \leq \sigma^\sharp$ and $o^\sharp \leq o'^\sharp$. As T is $\odot_{\tau_v}$ compatible, for any $\sigma^\sharp : \tau_\sigma$ and $o^\sharp : \tau_o$, we have $\odot_{\tau_v}(\tau_\sigma) \downarrow$ and $\odot_{\tau_v}(\tau_o) \downarrow$. Let $\tau_{\sigma'}$ such that $\sigma'^\sharp : \tau_{\sigma'}$. We then have $\sigma^\sharp : \tau_{\sigma'}$, hence $\odot_{\tau_v}(\tau_{\sigma'}) \downarrow$. As $\odot_{\tau_v}(\tau_o) \downarrow, o'^\sharp : \tau_o$, and abstract values are stratified, we have $\odot_{\tau_v}(\tau_{o'}) \downarrow$ for every $o'^\sharp : \tau_{o'}$. Hence T' is $\odot_{\tau_v}$ compatible. Next, let $\left(\sigma'^\sharp \oplus^\sharp_{v^\sharp}, t, o'^\sharp \oplus^\sharp_{v^\sharp}\right) \in T' \oplus^\sharp_{v^\sharp}$. Then $\left(\sigma'^\sharp, t, o'^\sharp\right) \in T'$ and there is $\left(\sigma^\sharp, t, o^\sharp\right) \in T$ such that $\sigma'^\sharp \leq \sigma^\sharp$ and $o^\sharp \leq o'^\sharp$. Then $\sigma'^\sharp \oplus^\sharp_{v^\sharp} \leq \sigma^\sharp \oplus^\sharp_{v^\sharp}$ and $o^\sharp \oplus^\sharp_{v^\sharp} \leq o'^\sharp \oplus^\sharp_{v^\sharp}$. Hence $\left(\sigma'^\sharp \oplus^\sharp_{v^\sharp}, t, o'^\sharp \oplus^\sharp_{v^\sharp}\right) \in NS_{\text{WEAKEN}}\left(T \oplus^\sharp_{v^\sharp}\right)$.

Fourth, let $T' \subseteq NS_{\text{FRAME-PARTITIONING}}(T)$, and $\left(\sigma'^\sharp, t, o'^\sharp\right) \in T'$. Then there are $\overrightarrow{\left(\sigma^\sharp_n, t, o'^\sharp\right) \in T}$ such that $\gamma\left(\sigma'^\sharp\right) \subseteq \bigcup_i \gamma\left(\sigma^\sharp_i\right)$. The compatibility is immediate for any type $o'^\sharp : \tau'_o$. Let $\sigma'^\sharp : \tau$, then there is a σ_i with type τ, hence $\odot_{\tau_v}(\tau) \downarrow$. Next, let $\left(\sigma'^\sharp \oplus^\sharp_{v^\sharp}, t, o'^\sharp \oplus^\sharp_{v^\sharp}\right) \in T' \oplus^\sharp_{v^\sharp}$. Then $\left(\sigma'^\sharp, t, o'^\sharp\right) \in T'$ and there are $\overrightarrow{\left(\sigma^\sharp_n, t, o'^\sharp\right)} \in T$ such that $\gamma\left(\sigma'^\sharp\right) \subseteq \bigcup_i \gamma\left(\sigma^\sharp_i\right)$. We immediately have $\overrightarrow{\left(\sigma^\sharp_n \oplus^\sharp_{v^\sharp}, t, o'^\sharp \oplus^\sharp_{v^\sharp}\right)} \in T \oplus^\sharp_{v^\sharp}$. Let τ' such that $\sigma'^\sharp \oplus^\sharp_{v^\sharp} : \tau'$, then there is a τ such that $\tau' = \odot_{\tau_v}(\tau)$ and $\sigma'^\sharp : \tau$. Hence on of the $\sigma^\sharp_i$ has type τ, hence $\sigma^\sharp_i \oplus^\sharp_{v^\sharp} : \odot_{\tau_v}(\tau) = \tau'$, as needed. Finally, let $\sigma'' \in \sigma'^\sharp \oplus^\sharp_{v^\sharp}$, then there are some v and σ' such that $v \in \gamma\left(v^\sharp\right)$, $\sigma' \in \gamma\left(\sigma'^\sharp\right)$, and $\sigma'' = \sigma' \oplus_v$. Thus there is some i such that $\sigma' \in \gamma\left(\sigma^\sharp_i\right)$, hence $\sigma' \oplus_v \in \gamma\left(\sigma^\sharp_i \oplus^\sharp_{v^\sharp}\right)$, as needed. □