Transforming certificates of program correctness along justified program transformations

Certificate translation in abstract interpretation (extended version)

Gilles Barthe and César Kunz

INRIA Sophia Antipolis-Méditerranée
{Gilles.Barthe,Cesar.Kunz}@inria.fr

Abstract. A certificate is a mathematical object that can be used to establish that a piece of mobile code satisfies some security policy. Since in general certificates cannot be generated automatically, there is an interest in developing methods to reuse certificates. This article studies methods that transform certificates of a program into certificates of another program derived from the initial one by a semantically justified program transformation. We also study the related problem to transform certificates of a program instrumented with the results of a semantically justified program analysis, into a certificate of the original program. The transformations are conveniently described in the setting of abstract interpretation. We illustrate applications of our results to certificate translation and to the justification of hybrid certificates.

1 Introduction

A certificate c is a mathematical object that can be checked automatically against some property ϕ it intends to prove; certificates arise naturally in logic, in the context of proof checking (via the Curry-Howard isomorphism) and of result checking. Certificates are also used to carry evidence of innocuousness of components in mobile code: in a typical proof carrying code [18] scenario, a piece of mobile code is downloaded together with a certificate that shows its adherence to the consumer policy. While certificate checking is usually well-understood, certificate generation remains a challenging problem: while it is possible to generate certificates automatically for properties that are enforceable by automated program analyses, and in particular type systems, certificate generation remains necessarily interactive in the general case. It is therefore of interest to develop methods that simplify the construction of certificates.

In this paper, we use the setting of abstract interpretation [13, 14] to describe a method for transforming certificates along program transformations. We provide sufficient conditions for transforming a certificate of a program G into a certificate of a program G', where G' is derived from G by a semantically

We indicate in red the modifications with respect to the submitted version.

justified program transformation. These results provide substantial leverage on earlier work on certificate translation [8, 5]; see Section 2. A secondary contribution is to provide an abstract justification of hybrid verification methods, in which auxiliary verification methods, typically type systems, provide information to a primary verification method, e.g. a type system or a program logic, which exploits the information to increase its precision or efficiency. We formulate an abstract notion of hybrid certificate, in which the certificate for the auxiliary and primary methods cohabit, and apply our results on certificate translation to show that hybrid certificates can be translated into primary certificates, and thus bring the same guarantees (often in more compact form).

2 A primer on certificate translation

The primary goal of certificate translation is to extend the scope of Proof Carrying Code [18] to arbitrarily complex policies, by supporting the generation of certificates from interactive source code verification. The scenario is of interest in situations where the functional correctness of the downloaded code is essential, and where certificate issues such as size or checking time are not relevant, e.g. in wholesale Proof Carrying Code, where one code verifier checks the certificate prior to distributing a cryptographically signed version to millions of code consumers [12].

Certificate translation is tightly bound to the compilation infrastructure: for compilers that do not perform any optimization, proof obligations are preserved (up to syntactic equality), and hence it is possible to reuse directly certificates of source code programs for their compilation; see e.g. [8, 6].

In contrast, program optimizations make certificate translation more challenging. In [5], we show in a simplified setting that one can define certificate transformers for common program optimizations, provided one can infer automatically certificates of correctness for the underlying program analyses, by means of certifying analyzers. The existence of certifying analyzers and certificate translators is shown individually for each optimization.

Comparison with our previous work. The initial motivation for the present work is to provide a framework in which to formulate the basic concepts of certificate translation. The lack of such a framework was a clear limitation of our earlier work: as a consequence, it was quite difficult to extend our results or even to assess the scalability of certificate translation.[1]

The present article addresses these limitations: we capture the essence of certificate translation in an algebraic setting that abstracts away from the specifics of programming languages, program transformations, and of verification methods. By providing sufficient conditions for the existence of certificate translators, the applicability of the method becomes easy to assess. In fact, our results

[1] Since certificate translation relies on program verification, one may consider that the scalability of certificate translation strictly depends on the scalability of program verification. However, we are only interested in the scalability of the translation process.

provide a means to generate, for given verification settings and program transformations, a set of proof obligations that guarantee the existence of certificate translators. All results of [6, 5, 8] can then be recovered by discharging these proof obligations. In order to further demonstrate the benefits of our framework as regards applicability, we show that certificate translation scales to concurrent languages, and can be used to justify hybrid certificates.

3 Certified solutions

This section extends the basic framework of abstract interpretation with certificate infrastructures, in order to introduce formally the notion of certified solution. Definition 5 provides a general definition of certified solution that is of independent interest from certificate transformation, and provides a unifying framework for existing *ad hoc* definitions, see Section 8. For the purpose of this article, one can think about certified solutions as:

- programs annotated with logical assertions, and bundled with a certificate of the correctness of the verification conditions, or;
- programs annotated with abstract values (or types), and bundled with a certificate that the program is correct with respect to the logical interpretation of the abstract values.

We view programs as flow graphs. Thus, programs are directed pointed graphs with a distinguished set of output nodes, from which execution may not flow.

Definition 1 (Programs). *A program is a pointed directed graph $G = \langle \mathcal{N}, \mathcal{E}, l_{\mathsf{sp}} \rangle$, where $\mathcal{N}$ is a set of nodes, $l_{\mathsf{sp}} \in \mathcal{N}$ is a distinguished initial node, and $\mathcal{E} \subseteq \mathcal{N} \times \mathcal{N}$ a finitely branching relation; elements of $\mathcal{E}$ are called edges. We let $\mathcal{O}$ be the set of nodes without successors.*

Throughout this section, we let $G = \langle \mathcal{N}, \mathcal{E}, l_{\mathsf{sp}} \rangle$ be a program. Both the analysis and verification frameworks are coined as abstract interpretations. Note that, in contrast to abstract interpretation, our domains are pre-orders, rather than partial orders[2].

Definition 2 (Abstract interpretation). *Let $G = \langle \mathcal{N}, \mathcal{E}, l_{\mathsf{sp}} \rangle$ be a program. An abstract interpretation of G is a triple $I = \langle A, \{T_e\}_{e \in \mathcal{E}}, f \rangle$, where*

- *A is a pre-lattice[3] $\langle D_A, \sqsubseteq_A, \sqsupseteq_A, \sqcup_A, \sqcap_A, \top_A, \bot_A \rangle$ of abstract states. By abuse of notation, we write A instead of D_A;*
- *f is the flow sense, either forward ($f = \downarrow$), or backward ($f = \uparrow$);*

[2] One natural domain for the verification infrastructure is that of propositions; we do not want to view it as a partial order, since it would later imply (in Definition 4) that logically equivalent formulas have the same certificates, which is not desirable.

[3] Although it is sufficient to consider meet or join semi-lattices, depending on the flow of the interpretation, we find it more convenient to require our domains to be pre-lattices, since we deal both with forward and backwards analyses.

– $\{T_e\}_{e\in\mathcal{E}} : A \to A$ *is a family of monotone transfer functions.*

Thus, an abstraction of the program consists of an abstract domain, e.g. assertions or types, and of transfer functions, e.g. weakest precondition transformers, or transfer functions. Program semantics can be defined in the same way; however, there is no need to consider semantics as soundness issues are orthogonal to the paper. A common means to verify program properties is to consider (pre- or post-) fixpoints of the transfer functions.

Definition 3 (Solution). *A labeling $S : \mathcal{N} \to A$ is a solution of I if*

– $f = \uparrow$ *and for every l in $\mathcal{N}$, $S(l) \sqsubseteq \bigsqcap_{\langle l,l'\rangle\in\mathcal{E}} T_{\langle l,l'\rangle}(S(l'))$;*
– $f = \downarrow$ *and for every node l in $\mathcal{N}$, $S(l) \sqsupseteq \bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T_{\langle l',l\rangle}(S(l'))$.*

In order to capture the notion of certified solution at an appropriate level of abstraction, we rely on a general notion of certificate infrastructure.

Definition 4 (Certificate infrastructure). *A certificate infrastructure for G consists of an abstract interpretation $I = \langle A, \{T_e\}_{e\in\mathcal{E}}, f\rangle$ for G, and a proof algebra $\mathcal{P}$ that assigns to every $a, a' \in A$ a set of certificates $\mathcal{P}(\vdash a \sqsubseteq a')$ s.t.:*

– *$\mathcal{P}$ is closed under the operations of Figure 1, where $a, b, c \in A$;*
– *$\mathcal{P}$ is sound, i.e. for every $a, a' \in A$, if $a \not\sqsubseteq a'$, then $\mathcal{P}(\vdash a \sqsubseteq a') = \emptyset$.*

In the sequel, we write $c :\vdash a \sqsubseteq a'$ or $c :\vdash a' \sqsupseteq a$ instead of $c \in \mathcal{P}(\vdash a \sqsubseteq a')$.

In the context of standard proof carrying code, the underlying pre-lattice is that of logical assertions, with logical implication $\Rightarrow$ as pre-order, and the transfer functions are the predicate transformers (based on weakest precondition or strongest postcondition) induced by instructions at any given program point. The particular form of certificates is irrelevant for this paper. It may nevertheless be helpful for the reader to think about certificates in terms of the Curry-Howard isomorphism and consider that $\mathcal{P}$ is given by the typing judgment in a dependently typed λ-calculus, i.e. $\mathcal{P}(\phi) = \{e \in \mathcal{E} \mid\vdash e : \phi\}$, where $\mathcal{E}$ is the set of expressions of the type theory. Under such assumptions, one can provide an obvious type-theoretical interpretation to the functions of Figure 1; for example, $\mathsf{intro}_\sqcap$ is given by the λ-term $\lambda f.\ \lambda g.\ \lambda a.\ \langle fa, ga\rangle$.

In the sequel, we let $I = \langle A, \{T_e\}, f\rangle$ be a certificate infrastructure for G.

Definition 5 (Certified solution). *A certified solution for I is a pair $\langle S, \boldsymbol{c}\rangle$, where $S : \mathcal{N} \to A$ is a labeling and $\boldsymbol{c} = (c_l)_{l\in\mathcal{N}}$ is a family of certificates s.t. for every $l \in \mathcal{N}$,*

– *if $f = \uparrow$ then $c_l :\vdash S(l) \sqsubseteq \bigsqcap_{\langle l,l'\rangle\in\mathcal{E}} T_{\langle l,l'\rangle}(S(l'))$;*
– *if $f = \downarrow$ then $c_l :\vdash \bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T_{\langle l',l\rangle}(S(l')) \sqsubseteq S(l)$.*

It follows that S is a solution for I.

Many techniques, including lightweight bytecode verification and abstraction carrying code, do not bundle code with (certified) solutions, but with a partial labeling (and some certificates) from which a (certified) solution can be reconstructed. The remaining of this section relates the construction of a (certified) solution from a partial labeling.

$$
\begin{array}{lcl}
\mathsf{axiom} & : & \mathcal{P}(\vdash a \sqsubseteq a) \\
\mathsf{weak}_{\sqcap} & : & \mathcal{P}(\vdash a \sqsubseteq b) \to \mathcal{P}(\vdash a \sqcap c \sqsubseteq b) \\
\mathsf{weak}_{\sqcup} & : & \mathcal{P}(\vdash a \sqsubseteq b) \to \mathcal{P}(\vdash a \sqsubseteq b \sqcup c) \\
\mathsf{elim}_{\sqcap} & : & \mathcal{P}(\vdash c \sqcap a \sqsubseteq b) \to \mathcal{P}(\vdash c \sqsubseteq a) \to \mathcal{P}(\vdash c \sqsubseteq b) \\
\mathsf{intro}_{\sqcup} & : & \mathcal{P}(\vdash a \sqsubseteq c) \to \mathcal{P}(\vdash b \sqsubseteq c) \to \mathcal{P}(\vdash a \sqcup b \sqsubseteq c) \\
\mathsf{intro}_{\sqcap} & : & \mathcal{P}(\vdash a \sqsubseteq b) \to \mathcal{P}(\vdash a \sqsubseteq c) \to \mathcal{P}(\vdash a \sqsubseteq b \sqcap c)
\end{array}
$$

Fig. 1. Proof Algebra

Definition 6 (Labeling). *A partial labeling is a partial function $S : \mathcal{N} \rightharpoonup A$ s.t. entry and output nodes are annotated, i.e. $\mathcal{O} \cup \{l_{\mathsf{sp}}\} \subseteq \mathsf{dom}(S)$, and such that the program is sufficiently annotated, i.e. the restriction $G_{\mathcal{N} \setminus \mathsf{dom}(S)}$ of G to nodes that are not annotated is acyclic. A labeling S is total if $\mathsf{dom}(S) = \mathcal{N}$.*

In a partial labeling, annotations on entry and output nodes serve as specification, whereas we need sufficient annotations to reconstruct a total labeling from a partial one.

Definition 7 (Annotation propagation, verification condition). *Let* annot *be a partial labeling. The labeling* $\overline{\mathsf{annot}}$ *is defined by the clause:*

- *if* $f = \uparrow$, $\overline{\mathsf{annot}}(l) = \begin{cases} \mathsf{annot}(l) & \text{if } l \in \mathsf{dom}(\mathsf{annot}) \\ \bigsqcap_{\langle l,l' \rangle \in \mathcal{E}} T_{\langle l,l' \rangle}(\overline{\mathsf{annot}}(l')) & \text{otherwise} \end{cases}$
- *if* $f = \downarrow$, $\overline{\mathsf{annot}}(l) = \begin{cases} \mathsf{annot}(l) & \text{if } l \in \mathsf{dom}(\mathsf{annot}) \\ \bigsqcup_{\langle l',l \rangle \in \mathcal{E}} T_{\langle l',l \rangle}(\overline{\mathsf{annot}}(l')) & \text{otherwise} \end{cases}$

For every $l \in \mathsf{dom}(\mathsf{annot})$*, the verification condition* $\mathsf{vc}(l)$ *is defined by the clause*

- $\mathsf{vc}(l) := \mathsf{annot}(l) \sqsubseteq \bigsqcap_{\langle l,l' \rangle \in \mathcal{E}} T_{\langle l,l' \rangle}(\overline{\mathsf{annot}}(l'))$ *if* $f = \uparrow$*;*
- $\mathsf{vc}(l) := \bigsqcup_{\langle l',l \rangle \in \mathcal{E}} T_{\langle l',l \rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq \mathsf{annot}(l)$ *if* $f = \downarrow$*.*

Given a partial labeling annot, one can build a certificate for $\overline{\mathsf{annot}}$ from certificates for the verification conditions on $\mathsf{dom}(\mathsf{annot})$.

Lemma 1. *Let* annot *be a partial labeling for I and assume given $c_l :\vdash \mathsf{vc}(l)$ for every $l \in \mathsf{dom}(\mathsf{annot})$. Then there exists $\boldsymbol{c}'$ s.t. $\langle \overline{\mathsf{annot}}, \boldsymbol{c}' \rangle$ is a certified solution.*

In the sequel, we shall abuse language and speak about certified solutions of the form $\langle \mathsf{annot}, \boldsymbol{c} \rangle$ where annot is a partial labeling and $\boldsymbol{c}$ is an indexed family of certificates that establish all verification conditions of annot.

4 Certifying analyzers

The certificate transformations studied in the next sections require that the analyzers upon which the program transformation is based are certifying, i.e. produce certificates which justify their results. In this section, we thus provide

sufficient conditions under which every solution may be certified. Proposition 1 below generalizes a previous result of Chaieb [11], who only considered the case where $f = \uparrow$ and $f^\sharp = \downarrow$.

Let G be a program, $I^\sharp = \langle A^\sharp, \{T_e^\sharp\}, f^\sharp \rangle$ be an abstract interpretation, $I = \langle A, \{T_e\}, f \rangle$ a certificate infrastructure of program G, and $\gamma : A^\sharp \to A$ a concretization function.

Proposition 1 (Existence of certifying analyzers). *For every solution $S^\sharp$ of $I^\sharp$, one can compute $\boldsymbol{c}$ s.t. $\langle \gamma \circ S^\sharp, \boldsymbol{c} \rangle$ is a certified solution for I, provided there exist:*

- *for every $a, a' \in A^\sharp$ s.t. $a \sqsubseteq^\sharp a'$, a certificate $\mathsf{monot}_\gamma(a, a') :\vdash \gamma(a) \sqsubseteq \gamma(a')$;*
- *for every $x \in A$, a certificate $\mathsf{cons}(x) :\vdash \phi(x)$, where $\phi(x)$ is defined in Figure 2 according to the flows of the interpretations.*

Proof.

$$f = f^\sharp = \downarrow \begin{cases} \mathsf{hyp} := T^\sharp_{\langle l', l\rangle}(S(l')) \sqsubseteq S(l) \\ p_1 := \mathsf{monot}_\gamma(\mathsf{hyp}) :\vdash \gamma(T^\sharp_{\langle l', l\rangle}(S(l'))) \sqsubseteq \gamma(S(l)) \\ p_2 := \mathsf{cons}(S(l')) :\vdash T_{\langle l', l\rangle}(\gamma(S(l'))) \sqsubseteq \gamma(T^\sharp_{\langle l', l\rangle}(S(l'))) \\ p_3 := \mathsf{weak}_\sqcap(-, p_1) :\vdash \gamma(T^\sharp_{\langle l', l\rangle}(S(l'))) \sqcap T_{\langle l', l\rangle}(\gamma(S(l'))) \sqsubseteq \gamma(S(l)) \\ p_4 := \mathsf{elim}_\sqcap(p_3, p_2) :\vdash T_{\langle l', l\rangle}(\gamma(S(l'))) \sqsubseteq \gamma(S(l)) \\ \boldsymbol{c}_l := \mathsf{intro}_\sqcup(\{p_4\}_{\langle l', l\rangle \in \mathcal{E}}) :\vdash \bigsqcup_{\langle l', l\rangle \in \mathcal{E}} T_{\langle l', l\rangle}(\gamma(S(l'))) \sqsubseteq \gamma(S(l)) \end{cases}$$

$$f = f^\sharp = \uparrow \begin{cases} \mathsf{hyp} := S(l) \sqsubseteq^\sharp T^\sharp_{\langle l, l'\rangle}(S(l')) \\ p_1 := \mathsf{monot}_\gamma(\mathsf{hyp}) :\vdash \gamma(S(l)) \sqsubseteq \gamma(T^\sharp_{\langle l, l'\rangle}(S(l'))) \\ p_2 := \mathsf{cons}(S(l')) :\vdash \gamma(T^\sharp_{\langle l, l'\rangle}(S(l'))) \sqsubseteq T_{\langle l, l'\rangle}(\gamma(S(l'))) \\ p_3 := \mathsf{trans}(p_1, p_2) :\vdash \gamma(S(l)) \sqsubseteq T_{\langle l, l'\rangle}(\gamma(S(l'))) \\ \boldsymbol{c}_l := \mathsf{intro}_\sqcap(\{p_4\}_{\langle l, l'\rangle \in \mathcal{E}}) :\vdash \gamma(S(l)) \sqsubseteq \bigsqcap_{\langle l, l'\rangle \in \mathcal{E}} T_{\langle l, l'\rangle}(\gamma(S(l'))) \end{cases}$$

$$f = \uparrow \text{ and } f^\sharp = \downarrow \begin{cases} \mathsf{hyp} := T^\sharp_{\langle l', l\rangle}(S(l')) \sqsubseteq^\sharp S(l) \\ p_1 := \mathsf{monot}_\gamma(\mathsf{hyp}) :\vdash \gamma(T^\sharp_{\langle l', l\rangle}(S(l'))) \sqsubseteq \gamma(S(l)) \\ p_2 := \mathsf{monot}_T :\vdash T_{\langle l', l\rangle}(\gamma(T^\sharp_{\langle l', l\rangle}(S(l')))) \sqsubseteq T_{\langle l', l\rangle}((\gamma(S(l))) \\ p_3 := \mathsf{cons}(S(l')) :\vdash \gamma(S(l')) \sqsubseteq T_{\langle l', l\rangle}(\gamma(T^\sharp_{\langle l', l\rangle}(S(l')))) \\ p_4 := \mathsf{trans}(p_3, p_2) :\vdash \gamma(S(l')) \sqsubseteq T_{\langle l', l\rangle}(\gamma(S(l))) \\ \boldsymbol{c}_{l'} := \mathsf{intro}_\sqcap(\{p_4\}_{\langle l', l\rangle \in \mathcal{E}}) :\vdash \gamma(S(l')) \sqsubseteq \bigsqcap_{\langle l', l\rangle \in \mathcal{E}} T_{\langle l', l\rangle}(\gamma(S(l))) \end{cases}$$

$$f = \downarrow \text{ and } f^\sharp = \uparrow \begin{cases} \mathsf{hyp} := S(l) \sqsubseteq^\sharp T^\sharp_{\langle l, l'\rangle}(S(l')) \\ p_1 := \mathsf{monot}_\gamma(\mathsf{hyp}) :\vdash \gamma(S(l)) \sqsubseteq \gamma(T^\sharp_{\langle l, l'\rangle}(S(l'))) \\ p_2 := \mathsf{monot}_T :\vdash T_{\langle l, l'\rangle}(\gamma(S(l))) \sqsubseteq T_{\langle l, l'\rangle}(\gamma(T^\sharp_{\langle l, l'\rangle}(S(l')))) \\ p_3 := \mathsf{cons}(S(l')) :\vdash T_{\langle l, l'\rangle}(\gamma(T^\sharp_{\langle l, l'\rangle}(S(l')))) \sqsubseteq \gamma(S(l')) \\ p_4 := \mathsf{trans}(p_3, p_2) :\vdash T_{\langle l', l\rangle}(\gamma(S(l))) \sqsubseteq \gamma(S(l')) \\ \boldsymbol{c}_{l'} := \mathsf{intro}_\sqcup(\{p_4\}_{\langle l, l'\rangle \in \mathcal{E}}) :\vdash \bigsqcup_{\langle l, l'\rangle \in \mathcal{E}} T_{\langle l, l'\rangle}(\gamma(S(l))) \sqsubseteq \gamma(S(l')) \end{cases}$$

While Proposition 1 provides a means to construct certifying analyzers, it is sometimes of interest to rely on more direct methods to generate certificates: in [5], we show how to construct compact certificates for constant propagation and common sub-expression elimination in an intermediate language.

$f = f^\sharp = \downarrow$	$T_e(\gamma(x)) \sqsubseteq \gamma(T_e^\sharp(x))$
$f = f^\sharp = \uparrow$	$T_e(\gamma(x)) \sqsupseteq \gamma(T_e^\sharp(x))$
$f = \uparrow,\ f^\sharp = \downarrow$	$T_e(\gamma(T_e^\sharp(x))) \sqsupseteq \gamma(x)$
$f = \downarrow,\ f^\sharp = \uparrow$	$T_e(\gamma(T_e^\sharp(x))) \sqsubseteq \gamma(x)$

Fig. 2. Definition of $\phi(x)$

5 Certificate transformation

In this section, we provide sufficient conditions of existence for certificate transformers, that maps certificates of a program G into certificates of another program G', derived from G by a program transformation. Rather than attempting to prove a general result where G and G' are related in some complex manner, we establish three existence results that can be used in combination to cover many cases of interest.

In a first instance, certificate transformation as defined in Section 5.1 requires that the transformed program G' is a subgraph of the original program G. This is the case, for example, when G' is derived from G by applying optimizations such as constant propagation or common sub-expression elimination. In a second instance, Section 5.2 generalizes program transformations by allowing G' to contain additional nodes that arise from duplicating fragments of G, as is the case for transformations such as loop unrolling. Finally, in Section 5.3, we provide a notion of program skeleton, which abstracts away some of the structure of the program, to deal with transformations that do not preserve so tightly the structure of programs, such as code motion.

Throughout this section, we assume given two programs: an initial program $G = \langle \mathcal{N}, \mathcal{E}, l_{\mathsf{sp}} \rangle$ and a transformed program $G' = \langle \mathcal{N}', \mathcal{E}', l_{\mathsf{sp}} \rangle$. Furthermore, we assume given the required infrastructure to certify these programs; more concretely, consider the two abstract interpretations $I = \langle A, \{T_e\}_{e \in \mathcal{E}}, f \rangle$ and $I' = \langle A, \{T'_e\}_{e \in \mathcal{E}'}, f \rangle$ over G and G', and a proof algebra $\mathcal{P}$ over A. Note that the abstract interpretations share the same underlying domain and flow sense.

5.1 Basic case

In this section, we assume that G' is a subgraph of G, i.e. $\mathcal{N}' \subseteq \mathcal{N}$ and $\mathcal{E}' \subseteq \mathcal{E}$. Furthermore, we assume given an abstract interpretation $I^\sharp = \langle A^\sharp, \{T_e^\sharp\}_{e \in \mathcal{E}}, f^\sharp \rangle$ of G that justifies the transformation from G to G'.

Proposition 2 (Existence of certificate transformers). *Let $\langle S, \boldsymbol{c}^S \rangle$ be a certified solution for I such that for every $\langle l_1, l_2 \rangle \in \mathcal{E}'$ and $a \in A$:*

- *if $f = \uparrow$ then* $\mathsf{justif}(l_1, l_2) :\vdash S(l_1) \sqcap T_{\langle l_1, l_2 \rangle}(a) \sqsubseteq T'_{\langle l_1, l_2 \rangle}(a)$*;*
- *if $f = \downarrow$ then* $\mathsf{justif}(l_1, l_2) :\vdash T'_{\langle l_1, l_2 \rangle}(a) \sqsubseteq S(l_2) \sqcap T_{\langle l_1, l_2 \rangle}(a)$

Let $a = S(l)$, $a' = S(l')$, $T = T_{\langle l,l'\rangle}$ and $T' = T'_{\langle l,l'\rangle}$ in:

$$
\begin{array}{rl}
\mathsf{hyp}_1 := & \mathsf{monot}_T : \mathcal{P}(\vdash b_1 \sqsubseteq b_2) \to \mathcal{P}(\vdash T(b_1) \sqsubseteq T(b_2)) \\
\mathsf{hyp}_2 := & \mathsf{distrib}_T : \mathcal{P}(\vdash T(b_1) \sqcap T(b_2) \sqsubseteq T(b_1 \sqcap b_2)) \\
p_1 := & \mathsf{goal}(l') : \vdash a' \sqcap \overline{\mathsf{annot}}(l') \sqsubseteq \overline{\mathsf{annot}}'(l') \\
p_2 := & \mathsf{hyp}_1(p_1) : \vdash T'(a' \sqcap \overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
p_3 := & \mathsf{justif}(l,l') : \vdash a \sqcap T(a' \sqcap \overline{\mathsf{annot}}(l')) \sqsubseteq T'(a' \sqcap \overline{\mathsf{annot}}(l')) \\
p_5 := & \mathsf{elim}_\sqcap(\mathsf{weak}_\sqcap(-, p_2), p_3) : \vdash a \sqcap T(a' \sqcap \overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
p_6 := & \mathsf{hyp}_2 : \vdash T(a') \sqcap T(\overline{\mathsf{annot}}(l')) \sqsubseteq T(a' \sqcap \overline{\mathsf{annot}}(l')) \\
p_7 := & \mathsf{axiom} : \vdash a \sqsubseteq a \\
p_8 := & \mathsf{intro}_\sqcap(\mathsf{weak}_\sqcap(p_7), \mathsf{weak}_\sqcap(p_6)) : \vdash a \sqcap T(a') \sqcap T(\overline{\mathsf{annot}}(l')) \sqsubseteq a \sqcap T(a' \sqcap \overline{\mathsf{annot}}(l')) \\
p_9 := & \mathsf{elim}_\sqcap(\mathsf{weak}_\sqcap(p_5), p_8) : \vdash a \sqcap T(a') \sqcap T(\overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
p_{10} := & \boldsymbol{c}_l^S : \vdash a \sqsubseteq T(a') \\
p_{11} := & \mathsf{elim}_\sqcap(p_9, p_{10}) : \vdash a \sqcap T(\overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
p_{12} := & \mathsf{weak}_\sqcap(p_{11}) : \vdash a \sqcap \textstyle\prod_{\langle l,l'\rangle \in \mathcal{E}} T(\overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
\mathsf{goal}(l) := & \mathsf{intro}_\sqcap(\{p_{12}\}_{\langle l,l'\rangle \in \mathcal{E}}) : \vdash a \sqcap \textstyle\prod_{\langle l,l'\rangle \in \mathcal{E}} T(\overline{\mathsf{annot}}(l')) \sqsubseteq \prod_{\langle l,l'\rangle \in \mathcal{E}} T'(\overline{\mathsf{annot}}'(l'))
\end{array}
$$

Fig. 3. Definition of $\mathsf{goal}(l)$ for certificate translation (case $f = \uparrow$)

Then, provided the certificates in Fig. 5 are given for every $a_1, a_2, b_1, b_2 \in A$, one can transform every certified labeling $\langle \mathsf{annot}, \boldsymbol{c}\rangle$ for G into a certified labeling $\langle \mathsf{annot}', \boldsymbol{c}'\rangle$ for G', where $\mathsf{annot}'(l) = \mathsf{annot}(l) \sqcap S(l)$ for every node l in $\mathsf{dom}(\mathsf{annot}') = \mathsf{dom}(\mathsf{annot}) \cap \mathcal{N}'$.

Proof. The idea is to build for every l in $\mathcal{N}'$ the certificate

- $\mathsf{goal}(l) : \vdash S(l) \sqcap \overline{\mathsf{annot}}(l) \sqsubseteq \overline{\mathsf{annot}}'(l)$ *if $f = \uparrow$, or*
- $\mathsf{goal}(l) : \vdash \overline{\mathsf{annot}}'(l) \sqsubseteq S(l) \sqcap \overline{\mathsf{annot}}(l)$ *if $f = \downarrow$,*

from which the existence of a certificate for annot' follows. We proceed by induction, using the principle derived from the fact that annot is a sufficient annotation. More concretely, one can attach to every node a weight that corresponds to the length of the longest path to an annotated node, i.e. a node $l \in \mathsf{dom}(\mathsf{annot})$. In the base case, where $l \in \mathsf{dom}(\mathsf{annot}')$, the certificate $\mathsf{goal}(l)$ is defined trivially, since $\overline{\mathsf{annot}}'(l) = S(l) \sqcap \overline{\mathsf{annot}}(l)$. For the inductive step, where $l \notin \mathsf{dom}(\mathsf{annot}')$, the proof is given in Figures 3 and 4 respectively for the backward and forward case, where the application of certificates $\mathsf{assoc}_\sqcap^\leftarrow$, $\mathsf{assoc}_\sqcap^\rightarrow$ and $\mathsf{commut}_\sqcap$ is omitted for readability.

Using the results of Proposition 1, Proposition 2 can be instantiated to prove the existence of certificate transformers for many common optimizations, including constant propagation and common subexpression elimination. In a nutshell, one first runs the certifying analyzer, which provides the solution S, then performs the optimization, and finally one provides a justification $\mathsf{justif}(l_1, l_2)$ for each edge (instruction) that has been modified by the optimization. This process is further illustrated in Section 5.4. However, Proposition 2 does not cover optimizations that rely on second-order analyses $I^\sharp$ to justify their result, such

Let $a = S(l)$, $a' = S(l')$, $T = T_{\langle l,l'\rangle}$ and $T' = T'_{\langle l,l'\rangle}$ in:

$$
\begin{aligned}
p_1 &:= \mathsf{goal}'(l') :\vdash \overline{\mathsf{annot}}'(l') \sqsubseteq a' \sqcap \overline{\mathsf{annot}}(l') \\
p_2 &:= \mathsf{monot}_{T'} :\vdash T'_{\langle l',l\rangle}(\overline{\mathsf{annot}}'(l')) \sqsubseteq T'_{\langle l',l\rangle}(a' \sqcap \overline{\mathsf{annot}}(l')) \\
p_3 &:= \mathsf{justif} :\vdash T'_{\langle l',l\rangle}(a' \sqcap \overline{\mathsf{annot}}(l')) \sqsubseteq a \sqcap T_{\langle l',l\rangle}(a' \sqcap \overline{\mathsf{annot}}(l')) \\
p_4 &:= \mathsf{distrib}_T :\vdash T_{\langle l',l\rangle}(a' \sqcap \overline{\mathsf{annot}}(l')) \sqsubseteq T_{\langle l',l\rangle}(a') \sqcap T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \\
p_5 &:= \mathsf{weak}_{\sqcap}(p_4) :\vdash a \sqcap T_{\langle l',l\rangle}(a' \sqcap \overline{\mathsf{annot}}(l')) \sqsubseteq T_{\langle l',l\rangle}(a') \sqcap T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \\
p_6 &:= \boldsymbol{c}^S_{\langle l',l\rangle} :\vdash T_{\langle l',l\rangle}(a') \sqsubseteq a \\
p_7 &:= \mathsf{weak}_{\sqcap}(p_6) :\vdash T_{\langle l',l\rangle}(a') \sqcap T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq a \sqcap T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \\
p_8 &:= \mathsf{trans}(p_2, \mathsf{trans}(p_3, \mathsf{trans}(p_5,))) :\vdash T'_{\langle l',l\rangle}(\overline{\mathsf{annot}}'(l')) \sqsubseteq a \sqcap T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \\
p_9 &:= \mathsf{weak}_{\sqcup}(\mathsf{axiom}) :\vdash T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq \bigsqcup_{\langle l,l'\rangle\in\mathcal{E}} T(\overline{\mathsf{annot}}(l')) \\
p_{10} &:= \mathsf{intro}_{\sqcap}(\mathsf{weak}_{\sqcap}(\mathsf{axiom}), \mathsf{weak}_{\sqcap}(p_9)) : \\
&\qquad\qquad \vdash a \sqcap T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq a \sqcap \bigsqcup_{\langle l,l'\rangle\in\mathcal{E}} T(\overline{\mathsf{annot}}(l')) \\
p_{11} &:= \mathit{trans}(p_8, p_{10}) :\vdash T'_{\langle l',l\rangle}(\overline{\mathsf{annot}}'(l')) \sqsubseteq a \sqcap \bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T(\overline{\mathsf{annot}}(l')) \\
P_{12} &:= \mathsf{intro}_{\sqcup}(\{p_{13}\}_{\langle l',l\rangle\in\mathcal{E}'}) :\vdash \bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T'_{\langle l',l\rangle}(\overline{\mathsf{annot}}'(l')) \sqsubseteq a \bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T(\overline{\mathsf{annot}}(l'))
\end{aligned}
$$

Fig. 4. Definition of $\mathsf{goal}(l)$ for certificate translation (case $f = \downarrow$)

$$
\begin{aligned}
&\mathsf{monot}_T : \mathcal{P}(\vdash a_1 \sqsubseteq a_2) \to \mathcal{P}(\vdash T(a_1) \sqsubseteq T(a_2)) \\
&\mathsf{distr}^{\leftarrow}_{(T,\sqcap)} :\vdash T(a_1) \sqcap T(a_2) \sqsubseteq T(a_1 \sqcap a_2) \\
&\mathsf{distr}^{\rightarrow}_{(T,\sqcap)} :\vdash T(a_1 \sqcap a_2) \sqsubseteq T(a_1) \sqcap T(a_2) \\
&\mathsf{assoc}^{\leftarrow}_{\sqcap} : \mathcal{P}(\vdash a_1 \sqcap (b_1 \sqcap b_2) \sqsubseteq (a_1 \sqcap b_1) \sqcap b_2) \\
&\mathsf{assoc}^{\rightarrow}_{\sqcap} : \mathcal{P}(\vdash (a_1 \sqcap b_1) \sqcap b_2 \sqsubseteq a_1 \sqcap (b_1 \sqcap b_2)) \\
&\mathsf{commut}_{\sqcap} : \mathcal{P}(\vdash a_1 \sqcap a_2 \sqsubseteq a_2 \sqcap a_1)
\end{aligned}
$$

Fig. 5. Requirements for certificate translation.

as dead variable elimination. This motivates the following mild generalization, in which the transformation is justified w.r.t. a composition operator.

Proposition 3. *Let* $\centerdot : A \times A \to A$ *be a composition operator s.t. for every* $a_1, a_2, b_1, b_2 \in A$ *there exists a certificate*

$$\mathsf{monot}_{\centerdot} : \mathcal{P}(\vdash a_1 \sqsubseteq a_2) \to \mathcal{P}(\vdash b_1 \sqsubseteq b_2) \to \mathcal{P}(\vdash a_1 \centerdot b_1 \sqsubseteq a_2 \centerdot b_2)$$

Let $\langle S, \boldsymbol{c}^S\rangle$ *be a certified solution for* I *s.t. for every* $\langle l_1, l_2\rangle \in \mathcal{E}'$ *and* $a \in A$*:*

- *if* $f = \uparrow$ *then* $\mathsf{justif}(l_1, l_2) :\vdash S(l_1) \centerdot T_{\langle l_1,l_2\rangle}(a) \sqsubseteq T'_{\langle l_1,l_2\rangle}(a \centerdot S(l_2))$*;*
- *if* $f = \downarrow$ *then* $\mathsf{justif}(l_1, l_2) :\vdash T'_{\langle l_1,l_2\rangle}(a \centerdot S(l_1)) \sqsubseteq S(l_2) \centerdot T_{\langle l_1,l_2\rangle}(a)$

Then, provided the certificate monot_T *defined in Fig. 5 exist for all* $a_1, a_2 \in A$*, every certified labeling* $\langle \mathsf{annot}, \boldsymbol{c}\rangle$ *for* G *can be transformed into a certified labeling* $\langle \mathsf{annot}', \boldsymbol{c}'\rangle$ *for* G'*, where* $\mathsf{annot}'(l) = \mathsf{annot}(l) \centerdot S(l)$ *for every node* l *in* $\mathsf{dom}(\mathsf{annot}') = \mathsf{dom}(\mathsf{annot}) \cap \mathcal{N}'$*.*

Proof. The proof similar that of Proposition 2. The definition of goal*, from which the existence of* $\boldsymbol{c}'$ *follows, is sketched in Figures 6 and 7.*

Let $a = \gamma(S(l))$, $a' = \gamma(S(l'))$, $T = T_{\langle l,l'\rangle}$ and $T' = T'_{\langle l,l'\rangle}$ in:

$$
\begin{aligned}
p_1 &:= \mathsf{goal}(l') :\vdash a' \centerdot \overline{\mathsf{annot}}(l') \sqsubseteq \overline{\mathsf{annot}}'(l') \\
p_2 &:= \mathsf{monot}_{T'}(p_1) :\vdash T'(a' \centerdot \overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
p_3 &:= \mathsf{justif} :\vdash a \centerdot T(\overline{\mathsf{annot}}(l')) \sqsubseteq T'(a' \centerdot \overline{\mathsf{annot}}(l')) \\
p_4 &:= \mathsf{trans}(p_3, p_2) :\vdash a \centerdot T(\overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
p_5 &:= \mathsf{axiom} :\vdash T(\overline{\mathsf{annot}}(l')) \sqsubseteq T(\overline{\mathsf{annot}}(l')) \\
p_6 &:= \mathsf{axiom} :\vdash a \sqsubseteq a \\
p_7 &:= \mathsf{weak}_{\sqcap}(p_5) :\vdash \textstyle\bigsqcap_{\langle l,l'\rangle\in\mathcal{E}} T_{\langle l,l'\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq T(\overline{\mathsf{annot}}(l')) \\
p_8 &:= \mathsf{monot}_{\centerdot}(p_6, p_7) :\vdash a \centerdot \textstyle\bigsqcap_{\langle l,l'\rangle\in\mathcal{E}} T_{\langle l,l'\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq a \centerdot T(\overline{\mathsf{annot}}(l')) \\
p_9 &:= \mathsf{trans}(p_8, p_4) :\vdash a \centerdot \textstyle\bigsqcap_{\langle l,l'\rangle\in\mathcal{E}} T_{\langle l,l'\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq T'(\overline{\mathsf{annot}}'(l')) \\
p_{10} &:= \mathsf{intro}_{\sqcap}(\{p_9\}_{\langle l,l'\rangle\in\mathcal{E}}) :\vdash a \centerdot \textstyle\bigsqcap_{\langle l,l'\rangle\in\mathcal{E}} T_{\langle l,l'\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq \bigsqcap_{\langle l,l'\rangle\in\mathcal{E}} T'_{\langle l,l'\rangle}(\overline{\mathsf{annot}}'(l'))
\end{aligned}
$$

Fig. 6. Definition of goal for relational certificate translation (case $f = \uparrow$)

Let $a = \gamma(S(l))$, $a' = \gamma(S(l'))$, $T = T_{\langle l',l\rangle}$ and $T' = T'_{\langle l',l\rangle}$ in:

$$
\begin{aligned}
p_1 &:= \mathsf{goal}(l') :\vdash \overline{\mathsf{annot}}'(l') \sqsubseteq a' \centerdot \overline{\mathsf{annot}}(l') \\
p_2 &:= \mathsf{monot}_{T'}(p_1) :\vdash T'(\overline{\mathsf{annot}}'(l')) \sqsubseteq T'(a' \centerdot \overline{\mathsf{annot}}(l')) \\
p_3 &:= \mathsf{justif} :\vdash T'(a' \centerdot \overline{\mathsf{annot}}(l')) \sqsubseteq a \centerdot T(\overline{\mathsf{annot}}(l')) \\
p_4 &:= \mathsf{trans}(p_2, p_3) :\vdash T'(\overline{\mathsf{annot}}'(l')) \sqsubseteq a \centerdot T(\overline{\mathsf{annot}}(l')) \\
p_5 &:= \mathsf{axiom} :\vdash T(\overline{\mathsf{annot}}(l')) \sqsubseteq T(\overline{\mathsf{annot}}(l')) \\
p_6 &:= \mathsf{axiom} :\vdash a \sqsubseteq a \\
p_7 &:= \mathsf{weak}_{\sqcup}(p_5) :\vdash T(\overline{\mathsf{annot}}(l')) \sqsubseteq \textstyle\bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \\
p_8 &:= \mathsf{monot}_{\centerdot}(p_6, p_7) :\vdash a \centerdot T(\overline{\mathsf{annot}}(l')) \sqsubseteq a \centerdot \textstyle\bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \\
p_9 &:= \mathsf{trans}(p_4, p_8) :\vdash T'(\overline{\mathsf{annot}}'(l')) \sqsubseteq a \centerdot \textstyle\bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l')) \\
p_{10} &:= \mathsf{intro}_{\sqcup}(\{p_9\}_{\langle l',l\rangle\in\mathcal{E}}) :\vdash \textstyle\bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T'_{\langle l',l\rangle}(\overline{\mathsf{annot}}'(l')) \sqsubseteq a \centerdot \bigsqcup_{\langle l',l\rangle\in\mathcal{E}} T_{\langle l',l\rangle}(\overline{\mathsf{annot}}(l'))
\end{aligned}
$$

Fig. 7. Definition of goal for relational certificate translation (case $f = \downarrow$)

5.2 Code duplication

In this section, we consider the case where some subgraphs of the initial program are duplicated in the transformed program, with the aim to trigger further program optimizations. Typical cases of code duplication are loop unrolling and function inlining.

Definition 8 (Node replication). *A program $G^+ = \langle \mathcal{N} \cup \mathcal{N}^+, E^+, l_{\mathsf{sp}}\rangle$ is a result of replicating nodes of program $G = \langle \mathcal{N}, E, l_{\mathsf{sp}}\rangle$ if $\mathcal{N}^+ \subseteq \{l^+ \mid l \in \mathcal{N}\}$ and $\mathcal{E} = \{\langle l_1, l_2\rangle \mid \langle l, l'\rangle \in \mathcal{E}^+ \wedge \langle l, l'\rangle \in \{l_1, l_1^+\} \times \{l_2, l_2^+\}\}$.*

Let $\langle I, \mathcal{P}\rangle$ be a certificate infrastructure with $I = \langle A, \{T_e\}_{e\in\mathcal{E}}, f\rangle$. Then, we define an extended certificate infrastructure $I^+ = \langle A, \{T_e\}_{e\in\mathcal{E}^+}, f\rangle$ for program G^+, the transfer functions T_e for $e \in \mathcal{E}^+ \setminus \mathcal{E}$ being such that for all $\langle \bar{l}_1, \bar{l}_2\rangle \in \mathcal{E}^+$, with $\bar{l}_i \in \{l_i, l_i^+\}$, $T_{\langle l_1,l_2\rangle} = T_{\langle \bar{l}_1,\bar{l}_2\rangle}$.

Proposition 4. *Assume the certificates of Fig. 5 exist. Then every certified solution $\langle S, \boldsymbol{c}\rangle$ for G can be transformed into a certified solution $\langle S^+, \boldsymbol{c}'\rangle$ for G^+, s.t. $S^+(l^+) = S(l)$ for all $l \in \mathsf{dom}(S)$.*

Proof. The proof proceeds by the induction principle explained in Proposition 2. It consist mainly on a proof, for all $l \in \mathcal{N}$ and $\bar{l} \in \mathcal{N} \cup \mathcal{N}^+$ s.t. $\bar{l} \in \{l, l^+\}$. of the certificate

- $\mathsf{goal}(l, \bar{l}) :\vdash \overline{\mathsf{annot}}(l) \sqsubseteq \overline{\mathsf{annot}}^+(\bar{l})$, *if* $f = \uparrow$, *or*
- $\mathsf{goal}(l, \bar{l}) :\vdash \overline{\mathsf{annot}}^+(\bar{l}) \sqsubseteq \overline{\mathsf{annot}}(l)$, *if* $f = \downarrow$.

For $l, \bar{l}$ s.t. $l \in \mathsf{dom}(\mathsf{annot})$, goal is trivial by definition. A sketch of the inductive step for the backward case follows, where we implicitly used the hypothesis $T_{\langle l,l'\rangle} = T_{\langle \bar{l},\bar{l'}\rangle}$:

$$
\begin{aligned}
p_1 &:= \mathsf{goal}(l', \bar{l'}) :\vdash \overline{\mathsf{annot}}(l') \sqsubseteq \overline{\mathsf{annot}}^+(\bar{l'}) \\
p_2 &:= \mathsf{monot}_T(p_1) :\vdash T_{\langle l,l'\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq T_{\langle \bar{l},\bar{l'}\rangle}(\overline{\mathsf{annot}}^+(\bar{l'})) \\
p_3 &:= \mathsf{weak}(p_2) :\vdash \textstyle\prod_{\langle l,l'\rangle \in \mathcal{E}} T_{\langle l,l'\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq T_{\langle \bar{l},\bar{l'}\rangle}(\overline{\mathsf{annot}}^+(\bar{l'})) \\
\mathsf{goal}(l) &:= \mathsf{intro}_{\sqcap}(p_3) :\vdash \textstyle\prod_{\langle l,l'\rangle \in \mathcal{E}} T_{\langle l,l'\rangle}(\overline{\mathsf{annot}}(l')) \sqsubseteq \prod_{\langle \bar{l},\bar{l'}\rangle \in \mathcal{E}^+} T_{\langle \bar{l},\bar{l'}\rangle}(\overline{\mathsf{annot}}^+(\bar{l'}))
\end{aligned}
$$

5.3 Program skeletons

Proposition 2 requires that the transformation is justified for each edge of the program; this rules out several well known optimizations such as instruction swapping or code motion, whose justification involve more than one instruction. To overcome this limitation, one can abandon the intuitive representation of programs, where each edge represents one instruction, and cluster several instructions into a single edge. The purpose of this section is to capture formally this idea of clustering, and use it to strengthen our basic result.

Throughout this section, we assume that $\mathcal{N}_0 \subseteq \mathcal{N}$ is a set of nodes such that $G_{|\mathcal{N}\setminus\mathcal{N}_0}$ and $G'_{|\mathcal{N}'\setminus\mathcal{N}_0}$ are acyclic. We define $\mathcal{E}_0 = \mathcal{E}^\star \cap \mathcal{N}_0 \times \mathcal{N}_0$ where $\mathcal{E}^\star$ denote the transitive closure of $\mathcal{E}$. Let $\langle I, \mathcal{P}\rangle$ be a certificate infrastructure with $I = \langle A, \{T_e\}, f\rangle$. The transfer functions $\hat{T}$ are defined for every $\langle l, l'\rangle \in \mathcal{E}^0$ and $a \in A$ as $\check{T}_{\langle l,l'\rangle}(a)$, where $\check{T}_e$ is defined for every $e \in \mathcal{E}$ as:

- if $f = \uparrow$, $\begin{cases} \check{T}_{\langle l,l'\rangle} = T_{\langle l,l'\rangle} & \langle l, l'\rangle \in \mathcal{E} \\ \check{T}_{\langle l,l'\rangle}(a) = \prod_{\{\langle l,l''\rangle \in \mathcal{E} \mid \mathsf{reaches}(l'',l')\}} T_{\langle l,l''\rangle}(\check{T}_{\langle l',l''\rangle}(a)) & \langle l, l'\rangle \notin \mathcal{E} \end{cases}$
- if $f = \downarrow$, $\begin{cases} \check{T}_{\langle l',l\rangle} = T_{\langle l',l\rangle} & \langle l', l\rangle \in \mathcal{E} \\ \check{T}_{\langle l',l\rangle}(a) = \bigsqcup_{\{\langle l'',l\rangle \in \mathcal{E} \mid \mathsf{reaches}(l',l'')\}} T_{\langle l'',l\rangle}(\check{T}_{\langle l',l''\rangle}(a)) & \langle l', l\rangle \notin \mathcal{E} \end{cases}$

where the condition $\mathsf{reaches}(l, l')$ stands for the existence of a sequence of labels $l_1, \ldots, l_k$ with $l_1 = l$ and $l_k = l'$ s.t. $\langle l_i, l_{i+1}\rangle \in \mathcal{E}$, for all $i \in \{1, \ldots, k-1\}$. The set $\mathcal{E}'_0$ and the transfer functions $\hat{T}'$ are defined in a similar fashion.

The results of the previous sections extend immediately to program skeletons.

Lemma 2. *Let $\langle S, \boldsymbol{c}_I\rangle$ be a certified solution for I s.t $\mathsf{dom}(S) \subseteq \mathcal{N}_0$. Then $\langle \hat{S}, \hat{\boldsymbol{c}}_{\hat{I}}\rangle = \langle S, \boldsymbol{c}_{I|\mathcal{N}_0}\rangle$ is a certified solution of $\hat{I} = \langle A, \hat{T}_e, f\rangle$.*

Proposition 5. *Let* $\langle \hat{S}, \hat{\boldsymbol{c}}_{\hat{I}} \rangle = \langle S, \boldsymbol{c}_{I|\mathcal{N}_0} \rangle$ *be a certified solution of* $\hat{I} = \langle A, \hat{T}_e, f \rangle$. *Suppose that for every* $\langle l_1, l_2 \rangle \in \mathcal{E}'_0$ *and* $a \in A$:

- *if* $f = \uparrow$ *then* $\mathsf{justif}(l_1, l_2) :\vdash \hat{S}(l_1) \sqcap \hat{T}_{\langle l_1, l_2 \rangle}(a) \sqsubseteq \hat{T'}_{\langle l_1, l_2 \rangle}(a)$;
- *if* $f = \downarrow$ *then* $\mathsf{justif}(l_1, l_2) :\vdash \hat{T'}_{\langle l_1, l_2 \rangle}(a) \sqsubseteq \hat{S}(l_2) \sqcap \hat{T}_{\langle l_1, l_2 \rangle}(a)$

Then every certified labeling $\langle \mathsf{annot}, \boldsymbol{c} \rangle$ *for* G *such that* $\mathsf{dom}(\mathsf{annot}) \subseteq \mathcal{N}_0$ *can be transformed into a certified labeling* $\langle \mathsf{annot}', \boldsymbol{c}' \rangle$ *for* G', *where* $\mathsf{annot}'(l)$ *is defined as* $\mathsf{annot}(l) \sqcap S(l)$ *for all* $l \in \mathsf{dom}(\mathsf{annot}') = \mathsf{dom}(\mathsf{annot}) \cap \mathcal{N}'$.

Proposition 5 can be used to prove preservation of proof obligations for non-optimizing compilers. Indeed, non-optimizing compilation transforms a graph representation of a program by splitting each node into a subgraph of more basic nodes, preserving the overall program structure. Thus, one can coalesce back the generated subgraphs into a skeleton structure similar to the source program. If we assume that transfer functions of the skeleton representation are equal to those of the source program (it is not sufficient that the functions are equivalent w.r.t. $\sqsubseteq$; equality is essential), then proof obligations are preserved and certificates can be reused without modification.

5.4 Example

We now instantiate the results of this section on the example of a fast exponentiation algorithm. Its representation as a (labeled) graph is given in Figure 8.a; labels are either assignments of the form $x{:=}e$, in which case the node has exactly one successor, or conditional statements of the form $b?$, in which case the node has exactly two successor nodes, respectively corresponding to the true and false branch of the condition.

The certificate infrastructure is built over a weakest precondition calculus over first-order formulae. Thus, the backward transfer functions are defined, for any assertion ϕ, as $T_{\langle l,l' \rangle}(\phi) = \phi[\%_x]$ in case the node l contains the assignment $x{:=}e$, and as $b \Rightarrow \phi$ or $\neg b \Rightarrow \phi$ respectively for the positive and negative branch of a jump statement conditioned by the boolean expression b. We assume given a certificate of functional correctness for the program, i.e. we assume given a certified solution $\langle \mathsf{annot}, \boldsymbol{c} \rangle$ of $I = \langle A, \{T_e\}, \uparrow \rangle$, where annot (given in blue in the Figure) is the partial labeling s.t. the precondition is trivial, $\mathsf{annot}(l_1) = \mathsf{true}$, the invariant is $\mathsf{annot}(l_2) = c \times x'^{y'} = x^y$ and the postcondition is $\mathsf{annot}(l_7) = x'{=}x^y$.

The first transformation applied is loop unrolling, shown in Figure 8.b. Formally, it consists on duplicating a subset of nodes as defined in Section 5.2. In the example, nodes l_2, l_3, l_4 and l_5 are respectively duplicated into the nodes l'_2, l'_3, l'_4, l'_5 and a new subset of edges is defined accordingly. A certified labeling $\langle \mathsf{annot}^+, \boldsymbol{c}^+ \rangle$, where $\mathsf{annot}^+(l'_2) = \mathsf{annot}(l_2)$, can be generated for the program in Figure 8.b, by application of Proposition 4.

Next, suppose that we know (e.g. from the execution context) that the program is called with an even y; such knowledge is formalized by a precondition $y = 2 \times p$. Then, one can consider a forward abstract interpretation that analyses

parity of variables and which variables are modified. A certifying analyzer for such an abstract interpretation exists by Proposition 1 and will produce a certified solution $\langle S, \boldsymbol{c}^S \rangle$ (given in green in the picture), such that $S(l_1) \stackrel{\text{def}}{=} y = 2 \times p$, $S(l) \stackrel{\text{def}}{=} y' = 2 \times p \wedge x = x'$ for $l \in \{l'_2, l'_3, l'_5\}$ and $S(l) = \mathsf{true}$ in any other case.

Figure 8.c contains an optimized version of the program of Figure 8.b, where jump statements whose conditions can be determined statically have been eliminated (nodes l'_2 and l'_3) and unreachable nodes have been removed (node l'_4), and where assignments have been simplified by propagating the results of the analysis (node l'_5). By Proposition 2, one can build a certificate for the optimized program, with labeling $\mathsf{annot}'(l) = \mathsf{annot}(l) \sqcap S(l)$ for all nodes $l \in \mathsf{dom}(\mathsf{annot})$ (in blue in the figure), provided there exists, for every $a \in A$ and for every modified edge, i.e. for every $\langle l, l' \rangle \in \{\langle l'_2, l'_3 \rangle, \langle l'_3, l'_5 \rangle, \langle l'_5, l_2 \rangle\}$, a certificate:

$$\mathsf{justif}_{\langle l,l' \rangle} :\vdash y' = 2 \times p \wedge x = x' \sqcap T_{\langle l,l' \rangle}(a) \sqsubseteq T'_{\langle l,l' \rangle}(a)$$

The remaining certificates $\mathsf{justif}(l, l')$ for $\langle l, l' \rangle \notin \{\langle l'_2, l'_3 \rangle, \langle l'_3, l'_5 \rangle, \langle l'_5, l_2 \rangle\}$ are trivially generated since $T'_{\langle l,l' \rangle} = T_{\langle l,l' \rangle}$.

A further simple transformation consists of coalescing the nodes l'_2, l'_3 and l'_5 to simplify the graph representation. Formally, we use the program skeletons to cluster the sub-graph constituted by the nodes l'_2, l'_3 and l'_5 into a single node. Then, we define the transfer function $\hat{T}_{\langle l'_2, l_2 \rangle} = T'_{\langle l'_5, l_2 \rangle}$ (formally, one should have $T_{\langle l'_2, l_2 \rangle} = T'_{\langle l'_2, l'_3 \rangle} \circ T'_{\langle l'_3, l'_5 \rangle} \circ T'_{\langle l'_5, l_2 \rangle}$ but $T'_{\langle l'_2, l'_3 \rangle}$ and $T'_{\langle l'_3, l'_5 \rangle}$ are the identity function). Hence, by a trivial application of Proposition 5, there exists a certified solution $\langle \hat{\mathsf{annot}}, \hat{\boldsymbol{c}} \rangle$, for the collapsed program representation $\langle \mathcal{N}_0, \mathcal{E}_0, l_{\mathsf{sp}} \rangle$, s.t. $\hat{\mathsf{annot}}(l) = \mathsf{annot}(l)$ for all $l \in \mathcal{N}_0$.

Finally, we perform liveness analysis on program variables and remove assignments to dead variables. The resulting program is given in Figure 8.d. The remaining of this subsection is devoted to an explanation of the analysis, and to a justification of the transformation.

Assuming a standard program semantics, we say that a variable is live at a certain program point if its value will be needed in the future. An intentional definition of liveness would classify a variable x as live at a program node l whenever there is a path from l that reaches a boolean or arithmetic expression referring to x, without traversing an assignment to x. We prefer to use a more extensional interpretation of liveness, as in Benton's Relational Hoare Logic [9], identifying a declaration of a set of live variables as a relational proposition. To this end, we generalize the abstract domain A of the certificate infrastructure to include relational propositions. The extension consists on partitioning the domain of variables by attaching to each of them an index $\langle 1 \rangle$ or $\langle 2 \rangle$. The set of transfer functions is also modified accordingly; for instance, the substitution $\phi[^{e}/_{x}]$ corresponding to the assignment $x{:=}e$ at node l, is replaced by the substitution $\phi[^{e_{\langle 1 \rangle}}/_{x_{\langle 1 \rangle}}][^{e_{\langle 2 \rangle}}/_{x_{\langle 2 \rangle}}]$, where $e_{\langle i \rangle}$ is the result of indexing every variable occurring at e with $\langle i \rangle$.

Then, we define $\gamma(X) = \bigwedge_{v \in X} v_{\langle 1 \rangle} = v_{\langle 2 \rangle}$ as an interpretation of the fact that all variables in X are live. In order to generate a certificate for the optimized program, we apply Proposition 3, using as composition operator over relational

propositions the function $\centerdot$ defined as $\phi \centerdot \psi = \exists x^1, \ldots, x^k.\ \phi[{}^{x^1_{\langle 2 \rangle}}\!/_x] \ldots [{}^{x^k_{\langle 2 \rangle}}\!/_x] \wedge \psi[{}^{x^1_{\langle 1 \rangle}}\!/_x] \ldots [{}^{x^k_{\langle 1 \rangle}}\!/_x]$ where $\{x^1, \ldots, x^k\}$ are the set of variables in ϕ or ψ. For readability, if ϕ is a non-relational proposition, $\gamma(X) \centerdot \phi$ is denoted as the equivalent proposition $\exists y_1, \ldots, y_m.\ \phi$ where $\{y_1, \ldots, y_m\} = \mathsf{Var} - X$.

By Proposition 1, we can assume the existence of the certified solution $\langle \gamma \circ \mathsf{live}, \mathbf{c}'' \rangle$ s.t. $\mathsf{live}(l_1) = \{x, y\}$, $\mathsf{live}(l'_2) = \{x, y, c\}$ and $\mathsf{live}(l) = \{x, y, c, x', y'\}$ for $l \notin \{l_1, l'_2\}$. Since node l_1 contains an assignment to variables x' and y' and this variables are not live in node l'_2, we may safely simplify the statement by removing such assignments. From Proposition 3 we can transform the current certified solution by assuming the certificate

$$\mathsf{justif}(l_1, l'_2) :\vdash \gamma(\mathsf{live}(l_1)) \centerdot T_{\langle l_1, l'_2 \rangle}(\phi) \sqsubseteq T'_{\langle l_1, l'_2 \rangle}(\gamma(\mathsf{live}(l'_2)) \centerdot \phi)$$

whose goal is equivalent to $\vdash \phi[{}^1\!/_c][{}^x\!/_{x'}][{}^y\!/_{y'}] \sqsubseteq (\exists x', y'.\ \phi)[{}^1\!/_c]$.

6 Concurrency

This section generalizes the results on certificate transformation obtained for sequential programs. Since our goal is to show that the results of Section 4 and Section 5 scale to a concurrent setting, we base ourselves on a simple verification framework, without considering issues of usability. Thus, we adopt a verification infrastructure presented similar to Owicki-Gries logic [19], in the sense that verification is split in two independent tasks: local correctness, and global stability; however, we do not attempt to minimize the number of proof obligations by considering atomically executable code fragments.

Consider a program G, an abstract interpretation $I^\sharp = \langle A^\sharp, \{T^\sharp_e\}, f^\sharp \rangle$, a certificate infrastructure $\langle I, \mathcal{P} \rangle$ with $I = \langle A, \{T_e\}, f \rangle$ and a concretization function $\gamma : A^\sharp \to A$. The representation of a concurrent program remains a single graph, abstracting a simplified setting in which every concurrent process executes the same program description. In a concurrent environment we must extend the notion of solution of previous section. Given a labeling S, in addition to be a solution of an abstract interpretation I as defined in Section 3, namely local solution, we require S to be globally stable. We say that a condition a at node l is globally stable if a concurrent component executing at any other node l' will not invalidate a. We abstract this idea by requiring for every node l and edge $\langle l_1, l'_1 \rangle$, that the validity of $S(l_1)$ ensures the stability of the value $S(l)$ along the application of the transfer function $T_{\langle l_1, l'_1 \rangle}$.

Definition 9 (globally stable solution). *A labeling S, is a globally stable solution of $I^\sharp$ if it is a solution of $I^\sharp$ and for every edge $\langle l_1, l'_1 \rangle$ and node l the following condition holds:*

- $f = \uparrow$ *and* $S(l_1) \sqcap S(l) \sqsubseteq^\sharp T^\sharp_{\langle l_1, l'_1 \rangle}(S(l))$ *or,*
- $f = \downarrow$ *and* $T^\sharp_{\langle l'_1, l_1 \rangle}(S(l'_1) \sqcap S(l)) \sqsubseteq^\sharp S(l)$.

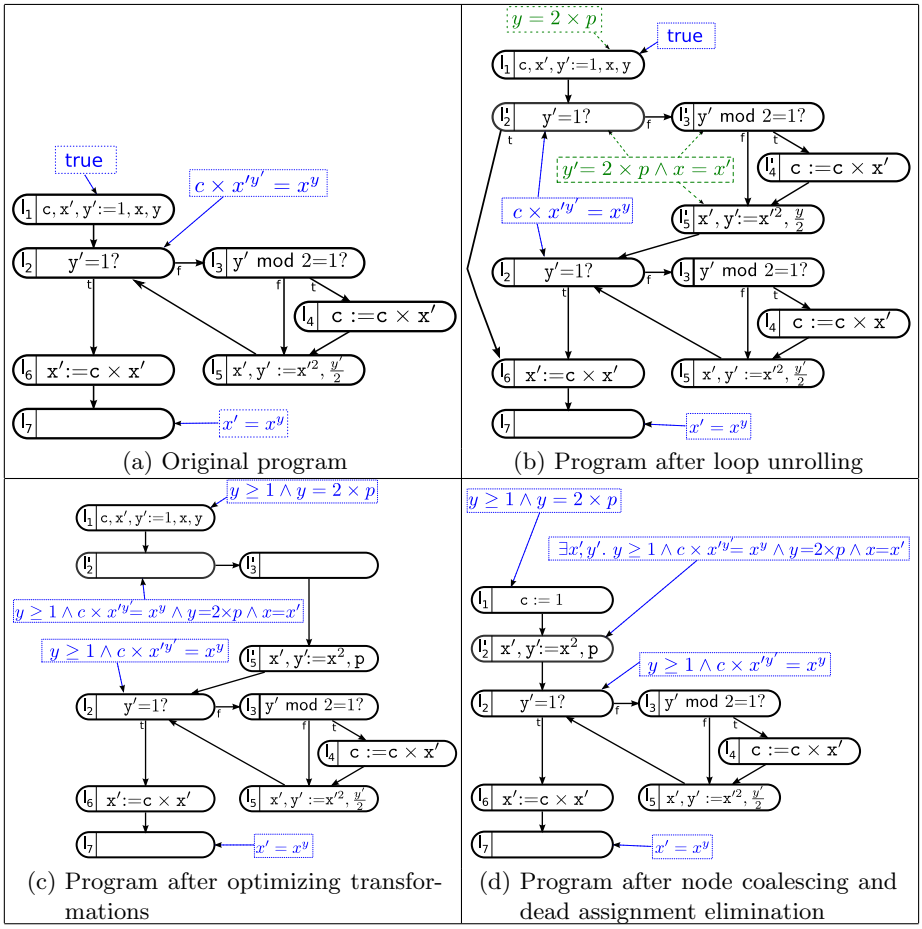

Fig. 8. Certificate translation example

To give an intuition of this definition, consider the statement $y{:=}2 \times x$ at node l_1 with successor l'_1. A labeling S such that $S(l_1) = x \geq 0$ and $S(l'_1) = y \geq 0$, satisfies the condition of being a solution of a weakest precondition calculus at node l_1. In a concurrent environment we must also verify that the conditions $x \geq 0$ and $y \geq 0$ are not invalidated by any other statement. For instance, if node l_2 contains the statement $x{:=}z^y$, we are due to ensure that it does not invalidate neither $x \geq 0$ nor $y \geq 0$, modulo the validity of $S(l_2)$. Being the latter task trivial, the former is feasible assuming $S(l_2) = \mathsf{even}(y) \wedge y \geq 0$. Respectively, $S(l_2)$ must also be proved stable with respect to the assignment at node l_1.

Definition 10 (certified globally stable solution). *A certified globally stable solution for I is a triple $\langle S, \boldsymbol{c}, \boldsymbol{c}' \rangle$ where $\langle S, \boldsymbol{c} \rangle$ is a certified solution of I and for all $\langle l_1, l'_1 \rangle \in \mathcal{E}$ and $l \in \mathcal{N}$,*

– $f = \uparrow$ *and* $\boldsymbol{c}'(l_1, l_1', l) : \vdash S(l_1) \sqcap S(l) \sqsubseteq T_{\langle l_1, l_1' \rangle}(S(l))$ *or,*
– $f = \downarrow$ *and* $\boldsymbol{c}'(l_1, l_1', l) : \vdash T_{\langle l_1', l_1 \rangle}(S(l_1') \sqcap S(l)) \sqsubseteq S(l)$.

Certifying analyzers can be extended to a concurrent setting.

Lemma 3 (Certifying Analyzers). *Consider a solution S for the abstract interpretation $I^\sharp$. Assume for every $a, a' \in A^\sharp$ s.t. $a \sqsubseteq^\sharp a'$, the certificates* $\mathsf{monot}_\gamma(a, a')$ *and* cons *defined in Proposition 1, and the certificate* $\mathsf{distrib}_{(\gamma,\sqcap)}$ *in Fig. 5. Then, one can compute $\boldsymbol{c}'$ s.t. $\langle \gamma \circ S, \boldsymbol{c}, \boldsymbol{c}' \rangle$ is a certified globally stable solution for I.*

Proof.

$$
f = f^\sharp = \uparrow \begin{cases}
\mathsf{hyp}{:=} S(l) \sqcap S(l_1) \sqsubseteq^\sharp T^\sharp_{\langle l, l' \rangle}(S(l_1)) \\
p_1{:=}\mathsf{monot}_\gamma : \vdash \gamma(S(l) \sqcap S(l_1)) \sqsubseteq \gamma(T^\sharp_{\langle l, l' \rangle}(S(l_1))) \\
p_2{:=}\mathsf{distr}_{(\gamma,\sqcap)} : \vdash \gamma(S(l)) \sqcap \gamma(S(l_1)) \sqsubseteq \gamma(S(l) \sqcap S(l_1)) \\
p_3{:=}\mathsf{trans}(p_2, p_1) : \vdash \gamma(S(l)) \sqcap \gamma(S(l_1)) \sqsubseteq \gamma(T^\sharp_{\langle l, l' \rangle}(S(l_1))) \\
p_4{:=}\mathsf{cons} : \vdash \gamma(T^\sharp_{\langle l, l' \rangle}(S(l_1))) \sqsubseteq T_{\langle l, l' \rangle}(\gamma \circ S(l_1)) \\
p_5{:=}\mathsf{trans}(p_3, p_4) : \vdash \gamma \circ S(l) \sqcap \gamma \circ S(l_1) \sqsubseteq T_{\langle l, l' \rangle}(\gamma \circ S(l_1))
\end{cases}
$$

$$
f = \downarrow, f^\sharp = \uparrow \begin{cases}
\mathsf{hyp}{:=} S(l) \sqcap S(l_1) \sqsubseteq^\sharp T^\sharp_{\langle l, l' \rangle}(S(l_1)) \\
p_1{:=}\mathsf{monot}_\gamma : \vdash \gamma(S(l) \sqcap S(l_1)) \sqsubseteq \gamma(T^\sharp_{\langle l, l' \rangle}(S(l_1))) \\
p_2{:=}\mathsf{distr}_{(\gamma,\sqcap)} : \vdash \gamma(S(l)) \sqcap \gamma(S(l_1)) \sqsubseteq \gamma(S(l) \sqcap S(l_1)) \\
p_3{:=}\mathsf{trans}(p_2, p_1) : \vdash \gamma(S(l)) \sqcap \gamma(S(l_1)) \sqsubseteq \gamma(T^\sharp_{\langle l, l' \rangle}(S(l_1))) \\
p_4{:=}\mathsf{monot}_T(p_3) : \vdash T_{\langle l, l' \rangle}(\gamma(S(l)) \sqcap \gamma(S(l_1))) \sqsubseteq T_{\langle l, l' \rangle}(\gamma(T^\sharp_{\langle l, l' \rangle}(S(l_1)))) \\
p_5{:=}\mathsf{cons} : \vdash T_{\langle l, l' \rangle}(\gamma(T^\sharp_{\langle l, l' \rangle}(S(l_1)))) \sqsubseteq \gamma \circ S(l_1) \\
p_6{:=}\mathsf{trans}(p_4, p_5) : \vdash T_{\langle l, l' \rangle}(\gamma(S(l)) \sqcap \gamma(S(l_1))) \sqsubseteq \gamma \circ S(l_1)
\end{cases}
$$

$$
f = f^\sharp = \downarrow \begin{cases}
\mathsf{hyp}{:=} T^\sharp_{\langle l', l \rangle}(S(l') \sqcap S(l_1)) \sqsubseteq^\sharp S(l_1) \\
p_1{:=}\mathsf{monot}_\gamma : \vdash \gamma(T^\sharp_{\langle l', l \rangle}(S(l') \sqcap S(l_1))) \sqsubseteq \gamma(S(l_1)) \\
p_2{:=}\mathsf{cons} : \vdash T_{\langle l', l \rangle}(\gamma(S(l') \sqcap S(l_1))) \sqsubseteq \gamma(T^\sharp_{\langle l', l \rangle}(S(l') \sqcap S(l_1))) \\
p_3{:=}\mathsf{trans}(p_2, p_1) : \vdash T_{\langle l', l \rangle}(\gamma(S(l') \sqcap S(l_1))) \sqsubseteq \gamma(S(l_1)) \\
p_4{:=}\mathsf{distr}_{(\gamma,\sqcap)} : \vdash \gamma(S(l')) \sqcap \gamma(S(l_1)) \sqsubseteq \gamma(S(l') \sqcap S(l_1)) \\
p_5{:=}\mathsf{monot}_T(p_4) : \vdash T_{\langle l', l \rangle}(\gamma(S(l')) \sqcap \gamma(S(l_1))) \sqsubseteq T_{\langle l', l \rangle}(\gamma(S(l') \sqcap S(l_1))) \\
p_6{:=}\mathsf{trans}(p_5, p_3) : \vdash T_{\langle l', l \rangle}(\gamma(S(l')) \sqcap \gamma(S(l_1))) \sqsubseteq \gamma(S(l_1))
\end{cases}
$$

$$
f = \uparrow, f^\sharp = \downarrow \begin{cases}
\mathsf{hyp}{:=} T^\sharp_{\langle l', l \rangle}(S(l') \sqcap S(l_1)) \sqsubseteq^\sharp S(l_1) \\
p_1{:=}\mathsf{monot}_\gamma : \vdash \gamma(T^\sharp_{\langle l', l \rangle}(S(l') \sqcap S(l_1))) \sqsubseteq \gamma(S(l_1)) \\
p_2{:=}\mathsf{monot}_T(p_1) : \vdash T_{\langle l', l \rangle}((\gamma(T^\sharp_{\langle l', l \rangle}(S(l') \sqcap S(l_1))))) \sqsubseteq T_{\langle l', l \rangle}(\gamma(S(l_1))) \\
p_3{:=}\mathsf{cons} : \vdash \gamma((S(l') \sqcap S(l_1))) \sqsubseteq T_{\langle l', l \rangle}((\gamma(T^\sharp_{\langle l', l \rangle}(S(l') \sqcap S(l_1))))) \\
p_4{:=}\mathsf{distr}_{(\gamma,\sqcap)} : \vdash \gamma \circ S(l') \sqcap \gamma \circ S(l_1) \sqsubseteq \gamma(S(l') \sqcap S(l_1)) \\
p_5{:=}\mathsf{trans}(p_4, \mathsf{trans}(p_3, p_2)) : \vdash \gamma \circ S(l') \sqcap \gamma \circ S(l_1) \sqsubseteq T_{\langle l', l \rangle}(\gamma(S(l_1)))
\end{cases}
$$

Let $G' = \langle \mathcal{N}', \mathcal{E}', l_{\mathsf{sp}} \rangle$ be a program transformed from G s.t. $\mathcal{N}' \subseteq \mathcal{N}$ and $\mathcal{E}' \subseteq \mathcal{E}$ with associated certificate infrastructure $\langle I', \mathcal{P} \rangle$, where $I' = \langle A, \{T_e'\}, f \rangle$. The following proposition generalizes certificate transformation for sequential programs (Proposition 2) for a concurrent setting.

Proposition 6 (Existence of certificate transformers). *Assume the existence of the certificates* $\mathsf{assoc}_\sqcap$, $\mathsf{commut}_\sqcap$ *and* $\mathsf{distr}_{(T,\sqcap)}$ *defined in Fig. 5. Let*

$\langle R, \boldsymbol{c}_R, \boldsymbol{c}'_R \rangle$ *be a certified globally stable solution of* I *s.t. for every* $\langle l_1, l_2 \rangle \in \mathcal{E}$ *and* $a \in A$ *we have the certificates:*

- $\mathsf{justif}(l_1, l_2) :\vdash T'_{\langle l_1,l_2\rangle}(a) \sqsubseteq R(l_2) \sqcap T_{\langle l_1,l_2\rangle}(a)$, *if* $f = \downarrow$; *or*
- $\mathsf{justif}(l_1, l_2) :\vdash R(l_1) \sqcap T_{\langle l_1,l_2\rangle}(a) \sqsubseteq T'_{\langle l_1,l_2\rangle}(a)$ *if* $f = \uparrow$.

Then one can transform every certified globally stable labeling $\langle S, \boldsymbol{c}, \boldsymbol{c}' \rangle$ *for* G *into a certified labeling* $\langle S', \boldsymbol{c}_2, \boldsymbol{c}'_2 \rangle$ *for* G', *where for all* $l \in \mathcal{N}'$, *we define* $S'(l)$ *equal to* $S(l) \sqcap R(l)$.

Proof. Building the certificate $\boldsymbol{c}_{S'}$ *is exactly the same procedure as in section 5. The case for the certificate* $\boldsymbol{c}'_{S'}$ *for non-interference follows:*

case $f = \uparrow$, *let* $T = T_{\langle l_1,l'_1\rangle}$ *and* $T' = T'_{\langle l_1,l'_1\rangle}$ *in:*

$$
\begin{aligned}
p_1 &:= \mathsf{justif} :\vdash R(l_1) \sqcap T(R(l_2)) \sqsubseteq T'(R(l_2)) \\
p_2 &:= \mathsf{weak}_\sqcap :\vdash R(l_1) \sqcap R(l_2) \sqcap T(R(l_2)) \sqsubseteq T'(R(l_2)) \\
p_3 &:= \boldsymbol{c}'_R :\vdash R(l_1) \sqcap R(l_2) \sqsubseteq T(R(l_2)) \\
p_4 &:= \mathsf{intro}_\sqcap(p_3, \mathsf{axiom}) :\vdash R(l_1) \sqcap R(l_2) \sqsubseteq T(R(l_2)) \sqcap R(l_1) \sqcap R(l_2) \\
p_5 &:= \mathsf{trans}(p_4, p_2) :\vdash R(l_1) \sqcap R(l_2) \sqsubseteq T'(R(l_2)) \\
p_6 &:= \boldsymbol{c}'_S :\vdash S(l_1) \sqcap S(l_2) \sqsubseteq T(S(l_2)) \\
p_7 &:= \mathsf{justif} :\vdash R(l_1) \sqcap T(S(l_2)) \sqsubseteq T'(S(l_2)) \\
p_8 &:= \mathsf{intro}_\sqcap(\mathsf{weak}_\sqcap(p_6), \mathsf{axiom}) :\vdash S(l_1) \sqcap S(l_2) \sqcap R(l_1) \sqsubseteq T(S(l_2)) \sqcap R(l_1) \\
p_9 &:= \mathsf{trans}(p_8, p_7) :\vdash S(l_1) \sqcap S(l_2) \sqcap R(l_1) \sqsubseteq T'(S(l_2)) \\
p_{10} &:= \mathsf{intro}_\sqcap(\mathsf{weak}_\sqcap(p_9), \mathsf{weak}_\sqcap(p_5)) :\vdash S'(l_1) \sqcap S'(l_2) \sqsubseteq T'(S(l_2)) \sqcap T'(R(l_2)) \\
p_{11} &:= \mathsf{distrib}_T :\vdash T'(S(l_2)) \sqcap T'(R(l_2)) \sqsubseteq T'(S'(l_2)) \\
\boldsymbol{c}'_S &:= \mathsf{trans}(p_{10}, p_{11}) :\vdash S'(l_1) \sqcap S'(l_2) \sqsubseteq T'(S'(l_2))
\end{aligned}
$$

case $f = \downarrow$, *let* $T = T_{\langle l'_1,l_1\rangle}$ *and* $T' = T'_{\langle l'_1,l_1\rangle}$ *in:*

$$
\begin{aligned}
p_1 &:= \boldsymbol{c}'_R :\vdash T(R(l'_1) \sqcap R(l_2)) \sqsubseteq R(l_2) \\
p_2 &:= \boldsymbol{c}'_S :\vdash T(S(l'_1) \sqcap S(l_2)) \sqsubseteq S(l_2) \\
p_3 &:= \mathsf{intro}_\sqcap(\mathsf{weak}(p_1), \mathsf{weak}(p_2)) :\vdash T(S(l'_1) \sqcap S(l_2)) \sqcap T(R(l'_1) \sqcap R(l_2)) \sqsubseteq S'(l_2) \\
p_4 &:= \mathsf{distr}_{(T,\sqcap)} T(S'(l'_1) \sqcap S'(l_2)) T(S(l'_1) \sqcap S(l_2)) \sqcap T(R(l'_1) \sqcap R(l_2)) \\
p_5 &:= \mathsf{trans}(p_4, p_3) :\vdash T(S'(l'_1) \sqcap S'(l_2)) \sqsubseteq S'(l_2) \\
p_6 &:= \mathsf{weak}(p_5) :\vdash T(S'(l'_1) \sqcap S'(l_2)) \sqcap (R(l_1)) \sqsubseteq S'(l_2) \\
p_7 &:= \mathsf{justif} :\vdash T'(S'(l'_1) \sqcap S'(l_2) \sqsubseteq T(S'(l'_1) \sqcap S'(l_2)) \sqcap (R(l_1)) \\
\boldsymbol{c}'_S &:= \mathsf{trans}(p_7, p_6) :\vdash T'(S'(l'_1) \sqcap S'(l_2)) \sqsubseteq S'(l_2)
\end{aligned}
$$

7 Hybrid certificates

In the preceding sections, we have considered scenarios where program analyses are used to justify program transformations. Here we look at another scenario where preliminary program analyses are used to gain valuable information (e.g. on its heap or on its control flow graph) that is exploited by another, main, verification task to increase either its precision or its efficiency. Such a scenario is rather common in verification of object-oriented programs. For example, a

semantically sound weakest precondition for a field assignment $x.f{:=}e$ must consider the case of normal execution, as well as the case when x is a null pointer and an exception is raised. However, it is sound to use a simplified weakest precondition considering only the normal case, provided a safety analysis result ensures that the variable x is not a null pointer,

We focus on the case where the preliminary analysis can be certified, and define hybrid certificates, that consist of a certified solution of the preliminary analysis, and a certificate of the program specification in a variant of the main verification framework. Thus, we consider certificate infrastructures $\langle I^\sharp, \mathcal{P}^\sharp\rangle$ and $\langle I, \mathcal{P}\rangle$ for program $G = \langle \mathcal{N}, \mathcal{E}, l_{\mathsf{sp}}\rangle$, with $I^\sharp = \langle A^\sharp, \{T_e^\sharp\}, f^\sharp\rangle$ and $I = \langle A, \{T_e\}, f\rangle$, and let $\gamma : A^\sharp \to A$ be a monotone concretization function.

Definition 11 (Hybrid certified solution). *An hybrid certified solution of $\langle I^\sharp, I\rangle$ is a pair $\langle p^\sharp, p\rangle$ where $p^\sharp$ and p are certified solutions of $I^\sharp$ and I respectively.*

We apply the results of the previous section to conclude that every hybrid certified solution can be transformed into a certified solution, from which the soundness of hybrid verification methods follows (assuming soundness of the primary verification method).

Corollary 1. *Every hybrid certified solution $\langle\langle S^\sharp, \boldsymbol{c}^\sharp\rangle, \langle S, \boldsymbol{c}\rangle\rangle$, can be transformed into a certified solution $\langle S', \boldsymbol{c}'\rangle$ for $I = \langle A, \{T'_e\}, f\rangle$, s.t. $S'(l) = S(l) \sqcap \gamma(S^\sharp(l))$ for all $l \in \mathsf{dom}(S)$, provided we are given, for all $\langle l, l'\rangle \in \mathcal{E}$ and $a \in A$, the certificates*

- $\mathsf{justif}(l, l') :\vdash \gamma(S^\sharp(l)) \sqcap T_{\langle l,l'\rangle}(a) \sqsubseteq T'_{\langle l,l'\rangle}(a)$ *if* $f = \uparrow$, *or*
- $\mathsf{justif}(l, l') :\vdash T'_{\langle l,l'\rangle}(a) \sqsubseteq \gamma(S^\sharp(l')) \sqcap T_{\langle l,l'\rangle}(a)$ *if* $f = \downarrow$

and $\mathsf{cons}(x) :\vdash \phi(x)$, *where* $\phi(x)$ *is defined in Figure 2.*

8 Related work

Certified solutions. Abstraction Carrying Code (ACC) is an instance of Proof Carrying Code where programs come with a solution in an abstract interpretation that can be used to specify the consumer policy [2]. ACC is closely related to our notion of certified solution; in fact, one may view the latter as a natural extension of ACC to settings where the pre-order relation is either undecidable, or expensive to compute, and where the use of certificates is required in order to check solutions. Abstraction Carrying Code offers the possibility to generate and update certificates incrementally [1]. Besson *et al* [10] have recently developed a program analysis framework in which certificates are used to verify inclusions between elements of the abstract domain of polyhedra. Their analysis is also an instance of a certified solution. Rival [20, 21] proposed a method to translate the result of a static analysis along program compilation. Result validation is restricted to post-fixpoint checking, i.e. there is no notion of certificate.

Certifying analyzers. We are aware of two previous works on certifying, or proof-producing, program analyses. Both consider the backwards case. Seo, Yang and Yi [23] consider a generic backwards abstract interpretation for a simple imperative language and provide an algorithm that automatically constructs safety proofs in Hoare logic from abstract interpretation results. Chaieb [11] considers a flow chart language equipped with a weakest precondition calculus, and provides sufficient conditions of the existence of certificates for solutions of backwards abstract interpretations. The technique was applied in the context of a certified PCC infrastructure [24].

Certificate translation. Müller and co-workers [3, 16] define a proof transforming compiler for sequential Java. They consider Hoare logics for source and bytecode programs, and transform a correct derivation for a Java program into a correct derivation for the JVM program obtained by non-optimizing compilation.

Saabas and Uustalu [22] develop type-based methods to establish the existence of certifying analyzers and certificate transformers. They illustrate the feasibility of their method by explaining in detail two particular transformations: common subexpression elimination and dead variable elimination. They demonstrate the correctness of both transformations, by derivability of Hoare logic proofs, and provide an algorithm to transform a Hoare proof of the original program to a Hoare proof of the transformed program.

Hybrid certificates. Grégoire and Sacchini [15] have proved the soundness in Coq of a hybrid verification condition generator for JVM programs; they use a null pointer analysis to reduce the number of proof obligations. In a similar way, Barthe, Pichardie and Rezk [7] rely on hybrid verification techniques for verifying information flow in JVM programs. More generally, hybrid verification is heavily used, both for type based analyses [17] and functional verification [4].

9 Conclusion

We have provided a crisp formalization of certificate translation in a mild extension of abstract interpretation in which solutions carry a certificate of their correctness. Our formalization allows us to give a rational reconstruction of our earlier work, to extend our earlier results to novel settings, and more generally to establish the scalability of certificate translation. A further benefit of the formalization is that it can be used to justify hybrid certificates.

Our next goal is to extend our results to relational program logics, which have recently emerged as a means to show information flow policies for programs. In parallel, we intend to refine our results on concurrent programs to verification methods that mitigate the explosion of verification conditions. Such verification methods should be expressible in our framework, using program skeletons to cluster atomically executable subsets of adjacent nodes into single nodes. Finally, we are also interested in the problem of transforming standard certificates into hybrid certificates. We conjecture that such a transformation is definable for certificates in a suitable normal form, and can be used for trimming proofs.

References

1. E. Albert, P. Arenas, and G. Puebla. Incremental certificates and checkers for abstraction-carrying code. In *Workshop on the Issues in the Theory of Security*, March 2006.
2. E. Albert, G. Puebla, and M. V. Hermenegildo. Abstraction-carrying code. In *Logic for Programming Artificial Intelligence and Reasoning*, number 3452 in Lecture Notes in Computer Science, pages 380–397. Springer-Verlag, 2005.
3. F. Y. Bannwart and P. Müller. A program logic for bytecode. In F. Spoto, editor, *Bytecode Semantics, Verification, Analysis and Transformation*, volume 141 of *Electronic Notes in Theoretical Computer Science*, pages 255–273. Elsevier, 2005.
4. M. Barnett, K. R. M. Leino, and W. Schulte. The Spec# programming system: An overview. In G. Barthe, L. Burdy, M. Huisman, J.-L. Lanet, and T. Muntean, editors, *Construction and Analysis of Safe, Secure and Interoperable Smart Devices: Proceedings of the International Workshop CASSIS 2004*, volume 3362 of *Lecture Notes in Computer Science*, pages 151–171. Springer-Verlag, 2005.
5. G. Barthe, B. Grégoire, C. Kunz, and T. Rezk. Certificate translation for optimizing compilers. In *Static Analysis Symposium*, number 4134 in Lecture Notes in Computer Science, Seoul, Korea, August 2006. Springer-Verlag.
6. G. Barthe, B. Grégoire, and M. Pavlova. Preservation of proof obligations for java. Draft paper, 2007.
7. G. Barthe, D. Pichardie, and T. Rezk. A certified lightweight non-interference java bytecode verifier. In R. De Niccola, editor, *European Symposium on Programming*, volume 4421 of *Lecture Notes in Computer Science*, pages 125 – 140. Springer-Verlag, 2007.
8. G. Barthe, T.Rezk, and A. Saabas. Proof obligations preserving compilation. In T. Dimitrakos, F. Martinelli, P. Ryan, and S. Schneider, editors, *Proceedings of FAST'05*, volume 3866 of *Lecture Notes in Computer Science*, pages 112–126. Springer-Verlag, 2005.
9. N. Benton. Simple relational correctness proofs for static analyses and program transformations. In *Principles of Programming Languages*, pages 14–25. ACM Press, 2004.
10. Frédéric Besson, Thomas Jensen, David Pichardie, and Tiphaine Turpin. Result certification for relational program analysis. Technical report, IRISA, 2007.
11. A. Chaieb. Proof-producing program analysis. In Kamel Barkaoui, Ana Cavalcanti, and Antonio Cerone, editors, *ICTAC*, volume 4281 of *Lecture Notes in Computer Science*, pages 287–301. Springer, 2006.
12. MobiusConsortium. Deliverable 4.1: Scenarios for proof-carrying code. Available online from `http://mobius.inria.fr`, 2006.
13. P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Principles of Programming Languages*, pages 238–252, 1977.
14. P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *Principles of Programming Languages*, pages 269–282, 1979.
15. B. Grégoire and J. Sacchini. Combining a verification condition generator for a bytecode language with static analyses. In *TGC'07*. Lecture Notes in Computer Science, 2007. To appear.
16. P. Müller and M. Nordio. Proof-transforming compilation of programs with abrupt termination. Technical Report 565, ETH Zurich, 2007.

17. A. C. Myers. JFlow: Practical mostly-static information flow control. In *Principles of Programming Languages*, pages 228–241. ACM Press, 1999. Ongoing development at `http://www.cs.cornell.edu/jif/`.
18. G. C. Necula. Proof-carrying code. In *Principles of Programming Languages*, pages 106–119, New York, NY, USA, 1997. ACM Press.
19. S. Owicki and D. Gries. An axiomatic proof technique for parallel programs. *Acta Informatica Journal*, 6:319–340, 1975.
20. X. Rival. Abstract Interpretation-Based Certification of Assembly Code. In L. D. Zuck, P. C. Attie, A. Cortesi, and S. Mukhopadhyay, editors, *Verification, Model Checking and Abstract Interpretation*, volume 2575 of *Lecture Notes in Computer Science*, pages 41–55, 2003.
21. X. Rival. Symbolic Transfer Functions-based Approaches to Certified Compilation. In *Principles of Programming Languages*, pages 1–13. ACM Press, 2004.
22. A. Saabas and T. Uustalu. Type systems for optimizing stack-based code. In M. Huisman and F. Spoto, editors, *Bytecode Semantics, Verification, Analysis and Transformation*, volume 190(1) of *Electron. Notes in Theor. Comput. Sci.*, pages 103–119. Elsevier, 2007.
23. S. Seo, H. Yang, and K. Yi. Automatic Construction of Hoare Proofs from Abstract Interpretation Results. In A. Ohori, editor, *Proceedings of APLAS'03*, volume 2895 of *Lecture Notes in Computer Science*, pages 230–245. Springer-Verlag, 2003.
24. M. Wildmoser, Amine Chaieb, and Tobias Nipkow. Bytecode analysis for proof carrying code. In *Bytecode Semantics, Verification, Analysis and Transformation*, 2005.

APPENDIX

A Verification Infrastructure Soundness

Due to space constraints, we have omitted in the paper a formalization of program semantics and the soundness of certificate infrastructures. In this appendix, we briefly provide sufficient conditions for a verification infrastructure to be sound. We begin by specifying the semantics of programs as a transition relation on states. The definition of states implicitly considers sequential programs, but we allow the semantics to be non-deterministic. Hereafter, we let $G = \langle \mathcal{N}, \mathcal{E}, l_{\mathsf{sp}} \rangle$ be a graph representation of a program and $I = \langle A, \{T_e\}, f \rangle$ an abstract interpretation of G.

A.1 Certified Solutions

Definition 12 (States, semantics). *Let* Env *be an abstract set of environments. The set of states is defined as* $\mathsf{State} = \mathcal{N} \times \mathsf{Env}$. *The semantics of program* G *is given by an abstract relation* $\rightsquigarrow \subseteq \mathsf{State} \times \mathsf{State}$.

For every abstract domain A, we assume that $\models_A \subseteq \mathsf{Env} \times A$ is a satisfaction relation, s.t. $\sqsubseteq$ is an approximation order, i.e., that for all $\eta \in \mathsf{Env}$, $a_1, a_2 \in A$, if $\models_A \eta : a_1$ and $a_1 \sqsubseteq a_2$ then $\models_A \eta : a_2$. In the following, we write simply $\models$ omitting the subscript A.

Definition 13 (Consistency). *We say that* I *is consistent with the semantics of* G *w.r.t.* $\models$ *iff for all states* $\langle l, \eta \rangle, \langle l', \eta' \rangle \in \mathsf{State}$ *such that* $\langle l, \eta \rangle \rightsquigarrow \langle l', \eta' \rangle$, *and for all* $a \in A$:

- *if* $f = \downarrow$ *and* $\models \eta : a$, *then* $\models \eta' : T_e(a)$;
- *if* $f = \uparrow$ *and* $\models \eta : T_e(a)$, *then* $\models \eta' : a$.

Lemma 4. *Let* S *be a solution of the abstract interpretation* $I = \langle A, \{T_e\}, f \rangle$ *and assume* I *consistent with the semantics of* G. *Then, if* $\langle l, \eta \rangle \rightsquigarrow^k \langle l', \eta' \rangle$ *and* $\models \eta : S(l)$ *then* $\models \eta' : S(l')$.

Proof. *The proof is by natural induction on* k.

Corollary 2. *Let* $\langle \mathsf{annot}, \boldsymbol{c} \rangle$ *be a certified partial labeling of* $\langle I, \mathcal{P} \rangle$ *and assume* I *consistent with the semantics of* G. *Then, if* $\langle l_{\mathsf{sp}}, \eta \rangle \rightsquigarrow^* \langle l_o, \eta' \rangle$ *with* $l_o \in \mathcal{O}$ *and* $\models \eta : \mathsf{annot}(l_{\mathsf{sp}})$ *then* $\models \eta' : \mathsf{annot}(l_o)$.

Proof. *This result follows from Lemma 4 and Lemma 1.*

A.2 Code Duplication

In Section 5.2 we have defined a program transformation that consists on duplicating some of the original nodes in G, and updating the set of labels accordingly. Let $G^+ = \langle \mathcal{N}^+, E^+, l_{\mathsf{sp}} \rangle$ be a result of duplicating nodes in program G. In the following, we define the extended semantics relation for program G^+ and show the soundness of the abstract interpretation I^+.

Definition 14. *The extended set of states is defined as* $\mathsf{State}^+ = \mathcal{N}^+ \times \mathsf{Env}$. *The semantics relation* $\overset{+}{\rightsquigarrow} \subseteq \mathsf{State}^+ \times \mathsf{State}^+$ *is defined from* $\rightsquigarrow$ *s.t.* $\langle \bar{l}, \eta \rangle \overset{+}{\rightsquigarrow} \langle \bar{l}', \eta' \rangle$ *iff* $\langle l, \eta \rangle \rightsquigarrow \langle l', \eta' \rangle$, *with* $\langle \bar{l}, \bar{l}' \rangle \in (\{l, l^+\} \times \{l', l'^+\}) \cap E^+$.

The following result implies as a corollary the soundness of $\langle I^+, \mathcal{P} \rangle$ from the soundness of $\langle I, \mathcal{P} \rangle$.

Lemma 5. *For all* $\eta, \eta' \in \mathsf{Env}$ *and* $\bar{l}, \bar{l}' \in \mathcal{N}^+$, *s.t.* $\langle \bar{l}, \eta \rangle \overset{+}{\rightsquigarrow}^* \langle \bar{l}', \eta' \rangle$ *we have that* $\langle l, \eta \rangle \rightsquigarrow^* \langle l', \eta' \rangle$.

Proof. The proposition

$$\forall \eta, \eta' \in \mathsf{Env}.\ \forall \bar{l}, \bar{l}' \in \mathcal{N}^+.\ \langle \bar{l}, \eta \rangle \overset{+}{\rightsquigarrow}^k \langle \bar{l}', \eta' \rangle \Rightarrow \langle l, \eta' \rangle \rightsquigarrow^k \langle l, \eta' \rangle$$

is proved by definition of $\overset{+}{\rightsquigarrow}$ *and natural induction over* k.

A.3 Program Skeletons

In Section 5.3 we have proposed to cluster adjacent nodes with the aim of considering a broader set of program transformations. This new representation must come accompanied with a corresponding semantics and a set of derived abstract interpretations. Although it is not essential for the results of the paper, we can show that the consistency of an abstract interpretation is propagated along node clustering.

Definition 15. *The semantics of program* $\hat{G}$ *is defined as a relation* $\hat{\rightsquigarrow} \subseteq \mathsf{State}_0 \times \mathsf{State}_0$, *where* State_0 *is a restriction of* State *over the set of nodes* $\mathcal{N}_0$, *such that* $\langle l, \eta \rangle \hat{\rightsquigarrow} \langle l', \eta' \rangle$ *iff* $\exists k \in \mathbb{N}.\ \langle l, \eta \rangle \rightsquigarrow^k \langle l', \eta' \rangle$.

Lemma 6. *Let* I *be an abstract interpretation of program* G, *and* I^+ *the corresponding derived abstract interpretation for* G^+. *Then the consistency of* $\hat{I}$ *w.r.t.* $\hat{\rightsquigarrow}$ *follows from the consistency of* I *w.r.t.* $\rightsquigarrow$.

Proof.

case $f = \uparrow$**:** *We have to show that for all* $\langle l, l' \rangle \in \mathcal{E}_0$, $a \in A$, *if* $\langle l, \eta \rangle \hat{\rightsquigarrow} \langle l', \eta' \rangle$ *and* $\models \eta : \hat{T}_{\langle l, l' \rangle}(a)$ *then* $\models \eta' : a$. *Equivalently, we show by induction on* k *that* $\langle l, \eta \rangle \rightsquigarrow^k \langle l', \eta' \rangle$ *and* $\models \eta : \check{T}_{\langle l, l' \rangle}(a)$ *implies* $\models \eta' : a$. *The base case*

is straightforward by definition of $\check{T}_{\langle l,l'\rangle}$ *in case* $\langle l,l'\rangle \in \mathcal{E}$. *In the inductive step, we have* $\langle l,\eta\rangle \rightsquigarrow \langle l'',\eta''\rangle \rightsquigarrow^k \langle l',\eta'\rangle$ *and*

$$\models \eta : \bigsqcap_{\{\langle l,l''\rangle \mid \mathsf{reaches}(l'',l')\}} T_{\langle l,l''\rangle}(\check{T}_{\langle l'',l'\rangle}(a))$$

Since $\sqsubseteq$ *is an approximation relation we have* $\models \eta : T_{\langle l,l''\rangle}(\check{T}_{\langle l'',l'\rangle}(a))$ *which by consistency of* T *with respect to* $\rightsquigarrow$ *implies* $\models \eta'' : \check{T}_{\langle l'',l'\rangle}(a)$. *By I.H.,* $\models \eta' : a$ *follows.*

case $f = \downarrow$**:** *We show by induction on* k *that* $\models \eta : a$ *and* $\langle l,\eta\rangle \rightsquigarrow^k \langle l',\eta'\rangle$ *then* $\models \eta' : \check{T}_{\langle l,l'\rangle}(a)$. *The base case is straightforward. For the inductive step, consider the sequence* $\langle l,\eta\rangle \rightsquigarrow^k \langle l'',\eta''\rangle \rightsquigarrow \langle l',\eta'\rangle$. *It follows from* $\models \eta : a$ *and inductive hypothesis that* $\models \eta'' : \check{T}_{\langle l,l''\rangle}(a)$ *and then, by consistency of* I *with respect to* $\rightsquigarrow$ *we have* $\models \eta' : T_{\langle l'',l'\rangle}(\check{T}_{\langle l,l''\rangle}(a))$. *Then,* $\models \eta' : \check{T}_{\langle l,l'\rangle}(a)$ *follows from definition of* $\check{T}_{\langle l,l'\rangle}$ *and the fact that* $\sqsubseteq$ *is an approximation relation.*

A.4 Concurrency

Definition 16. – *We define* $G^{\shuffle} = \langle \mathcal{N}^{\shuffle}, \mathcal{E}^{\shuffle}, l_{\mathsf{sp}}{}^{\shuffle}\rangle$ *the concurrent representation of program* $G = \langle \mathcal{N}, \mathcal{E}, l_{\mathsf{sp}}\rangle$, *where*

- $\mathcal{N}^{\shuffle} = \mathcal{N}^k$
- $\mathcal{E}^{\shuffle} = \{\langle l_1 \dots l_k, l'_1 \dots l'_k\rangle \mid \exists i.\ 0 < i \leq k \wedge \langle l_i, l'_i\rangle \in \mathcal{E} \wedge \forall j \neq i.\ l_j = l'_j\}$
- $l_{\mathsf{sp}} \dots l_{\mathsf{sp}}$

– *The set of concurrent states is defined as* $\mathsf{State}^{\shuffle} = \mathcal{N}^{\shuffle} \times \mathsf{Env}$.

– *The semantics relation* $\overset{\shuffle}{\rightsquigarrow} \subseteq \mathsf{State}^{\shuffle} \times \mathsf{State}^{\shuffle}$ *is s.t. for all* $\langle \boldsymbol{l},\eta\rangle$ *and* $\langle \boldsymbol{l}',\eta'\rangle$ *in* $\mathsf{State}^{\shuffle}$, $\langle \boldsymbol{l},\eta\rangle \rightsquigarrow \langle \boldsymbol{l}',\eta'\rangle$ *whenever* $\boldsymbol{l} = \langle l_1 \dots l_i \dots l_k\rangle$, $\boldsymbol{l}' = \langle l_1 \dots l'_i \dots l_k\rangle$ *and* $\langle l_i,\eta\rangle \overset{\shuffle}{\rightsquigarrow} \langle l'_i,\eta'\rangle$.

– *Similarly, if* $\boldsymbol{l} = \langle l_1 \dots l_i \dots l_k\rangle$ *and* $\boldsymbol{l}' = \langle l_1 \dots l'_i \dots l_k\rangle$, *we define the transfer function* $T_{\langle \boldsymbol{l},\boldsymbol{l}'\rangle} = T_{\langle l_i,l'_i\rangle}$.

Given a labeling $S : \mathcal{N} \to A$, we define the labeling $S^{\shuffle}$ s.t. $S^{\shuffle}(\boldsymbol{l}) = S(l_1) \sqcap \dots \sqcap S(l_k)$ for all $\boldsymbol{l} = l_1 \dots l_k$ in $\mathcal{N}^{\shuffle}$. The consistency of $I^{\shuffle}$ w.r.t. $\overset{\shuffle}{\rightsquigarrow}$ follows directly from the consistency of I w.r.t $\rightsquigarrow$.

Lemma 7. *Let* S *be a globally stable solution of* $I = \langle A, \{T_e\}_{e\in\mathcal{E}}, f\rangle$, *then* $S^{\shuffle}$ *is a solution of* $I^{\shuffle} = \langle A, \{T^{\shuffle}_e\}_{e\in\mathcal{E}^{\shuffle}}, f\rangle$.

Proof. Consider first the case $f = \downarrow$, *and the edge* $\langle \boldsymbol{l},\boldsymbol{l}'\rangle$ *s.t.* $\boldsymbol{l} = l_1 \dots l_i \dots l_k$ *and* $\boldsymbol{l}' = l_1 \dots l'_i \dots l_k$. *Since* $\langle \boldsymbol{l}',\boldsymbol{l}\rangle$ *is an arbitrary edge in* $\mathcal{E}^{\shuffle}$ *it is sufficient enough to show that* $T_{\langle \boldsymbol{l}',\boldsymbol{l}\rangle}(S^{\shuffle}(\boldsymbol{l}')) \sqsubseteq S^{\shuffle}(\boldsymbol{l})$ *or, what is the same,* $T_{\langle l'_i,l_i\rangle}(S^{\shuffle}(\boldsymbol{l}')) \sqsubseteq S^{\shuffle}(\boldsymbol{l})$. *Consider first the node* l_i. *We know by definition of* $S^{\shuffle}$ *and monotonicity of* T *that* $T_{\langle l'_i,l_i\rangle}(S^{\shuffle}(\boldsymbol{l}')) \sqsubseteq T_{\langle l'_i,l_i\rangle}(S(l'_i))$ *and then, since* $T_{\langle l'_i,l_i\rangle}(S(l'_i)) \sqsubseteq S(l_i)$, *it follows by transitivity that* $T_{\langle l'_i,l_i\rangle}(S^{\shuffle}(\boldsymbol{l}')) \sqsubseteq S(l_i)$. *Consider now any other node* l_j *different from* l_i. *We can show that* $T_{\langle l'_i,l_i\rangle}(S^{\shuffle}(\boldsymbol{l}')) \sqsubseteq S(l_j)$, *since*

$T_{\langle l'_i,l_i\rangle}(S^{\uplus}(\boldsymbol{l'})) \sqsubseteq T_{\langle l'_i,l_i\rangle}(S(l_j) \sqcap S(l'_i))$ *by monotonicity of* T *and since global stability implies* $T_{\langle l'_i,l_i\rangle}(S(l_j) \sqcap S(l'_i)) \sqsubseteq S(l_j)$. *Then it follows that* $T_{\langle l'_i,l_i\rangle}(S^{\uplus}(\boldsymbol{l'})) \sqsubseteq S(l_1) \sqcap \ldots \sqcap S(l_k)$. *For the case* $f = \uparrow$ *the proof is similar, but we must assume the distributivity of transfer functions with respect to the operator* $\sqcap$.