Sound up-to techniques and Complete abstract domains

Filippo Bonchi
University of Pisa
Pisa, Italy
filippo.bonchi@unipi.it

Pierre Ganty
IMDEA Software Institute
Pozuelo de Alarcón, Spain
pierre.ganty@imdea.org

Roberto Giacobazzi*
IMDEA Software Institute
Pozuelo de Alarcón, Spain
and University of Verona
Verona, Italy
roberto.giacobazzi@imdea.org

Dusko Pavlovic†
University of Hawaii
Honolulu, Hawaii, USA
dusko@hawaii.edu

Abstract

Abstract interpretation is a method to automatically find invariants of programs or pieces of code whose semantics is given via least fixed-points. Up-to techniques have been introduced as enhancements of coinduction, an abstract principle to prove properties expressed via greatest fixed-points.

While abstract interpretation is always sound by definition, the soundness of up-to techniques needs some ingenuity to be proven. For completeness, the setting is switched: up-to techniques are always complete, while abstract domains are not.

In this work we show that, under reasonable assumptions, there is an evident connection between sound up-to techniques and complete abstract domains.

Keywords abstract interpretation, complete abstract domains, coinduction up-to, sound up-to techniques, cross-fertilization

ACM Reference Format:
Filippo Bonchi, Pierre Ganty, Roberto Giacobazzi, and Dusko Pavlovic. 2018. Sound up-to techniques and Complete abstract domains. In *Proceedings of 33rd Annual ACM/IEEE Symposium on Logic in Computer Science (LICS), July 9–12, 2018, Oxford, United Kingdom.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3209108.3209169

1 Introduction

Abstract interpretation [8] is a general method for approximating invariants of dynamic systems. The key idea is that the analysis, or the possibility of proving invariance properties of a system, can be reduced to compute an approximate semantics of the system under inspection. Any monotone function b on a monotone lattice C can be approximated in a sound way by a function $\overline{b}$ on an smaller lattice A and an approximate invariant can be always obtained by computing the least fixed point of $\overline{b}$. Instances of abstract interpretation include sound-by-construction methodologies for the design of static program analyses, type analysis [6], and model checkers [11]. Examples of successful industrialisation cases of abstract interpretation in the context of program analysis are Astrée [12], a static program analyser aiming at proving the absence of run-time errors in industrial-size safety-critical C programs, Julia, for the analysis of Java and Android code for web applications, Clousot, developed at Microsoft Research [16] for statically checking Code Contracts in .Net, Infer and Zoncolan, at Facebook Inc. for scalable information-flow analysis [25].

One of the main limitation of abstract interpretation is *completeness*: the least fixed point of the approximate $\overline{b}$ is not always a faithful representation of the one of b, hence the latter could satisfy some properties not satisfied by the former. Computing on A rather than on C can thus lead to false alarms. Completeness should be intended as absence of false alarms.

This was first observed by Cousot and Cousot [10], where they also show a strategy to prove completeness of abstract domains: one can more easily prove a sufficient condition, that we call here *full completeness*, but that is known under difference names, like backward completeness [18] or (stepwise) completeness [19]. Interestingly enough, Giacobazzi et al. [19] observed that both completeness and full completeness can be regarded not as a property of $\overline{b}$, but rather of b and A. A symmetric condition, called *forward completeness*, was later introduced in [18] and used for an efficient simulation equivalence algorithm [34]. The key insight here is that when b is a left adjoint, then full completeness coincides with forward completeness of its right adjoint.

The rationale behind coinductive up-to techniques is apparently dual. Suppose we have a characterisation of an object of interest as a greatest fixed-point of some function. For instance, behavioural equivalence in CCS [24] is the greatest fixed-point of a monotone function b on the lattice of relations, describing the standard bisimulation game. This means that to prove two CCS terms equivalent, it suffices to exhibit a relation R that relates them, and which is a *b-simulation*, i.e., $R \subseteq b(R)$.

Alternatively, one can look for a relation R which is a b-simulation *up to some function a*, i.e., $R \subseteq b(a(R))$. However, not every function a can safely be used: a should be *sound* for b, meaning that any b-simulation up to a should be contained in a b-simulation. A similar phenomemon occurs in abstract interpretation where not all abstract domains are complete.

Since their introduction [24], coinduction up-to techniques were proved useful, if not essential, in numerous proofs about concurrent systems (see [31] for a list of references); it has been used to obtain decidability results [5], and more recently to improve standard automata algorithms [4]. It is worth to make clear at this point

*Partially supported by AFOSR.
†Partially supported by AFOSR.

LICS '18, July 9–12, 2018, Oxford, United Kingdom

ACM ISBN 978-1-4503-5583-4/18/07.
https://doi.org/10.1145/3209108.3209169

that, while abstract interpretation was originally intended as a fully automated approach to program analysis, coinduction up-to has always been seen as a proof principle: this explains the increasing spread of up to techniques amongst proof assistants (see e.g. [14]).

Since proving soundness of these techniques is rather complicated and error prone – a famous example of an unsound technique is that of weak bisimulation up to weak bisimilarity – Sangiorgi introduced the sufficient condition of respectfulness [35]. This was later refined by Pous with the notion of *compatibility* [28, 31].

In this paper we relate abstract interpretation with coinduction up-to. Our key observation (Remark 5.5) is that the notions of forward completeness and compatibility coincide. Using the key insight of Giacobazzi et al. [19] saying that when b is a left adjoint, full completeness of b coincides with compatibility of its right adjoint, we prove that the same abstraction can play the role of a complete abstract domain for abstract interpretation and a sound up-to technique for coinduction.

As a noteworthy example of this correspondence, we show in this paper an abstract-interpretation based analysis of the well-known Hopcroft and Karp's algorithm [1, 21] for checking language equivalence of deterministic automata. In this case the optimisation allowing the reduction of the state space, which is known from up-to techniques can also be derived by abstract interpretation.

Finally the connection between complete abstract interpretations and sound up-to techniques allows some technology transfer. As a proof of concept, we introduce in abstract interpretation the notion of companion, already known in coinduction up-to, that provides a way to simplify the checking of completeness for a generic abstraction. This also leads to the definition of a new and weaker notion of completeness for abstract interpretation which is *local* but sufficient to prove the absence of false alarms in program analysis.

The reader will find detailed proofs in a technical report [3].

2 Preliminaries and notation

We use $(L, \sqsubseteq)$, $(L_1, \sqsubseteq_1)$, $(L_2, \sqsubseteq_2)$ to range over complete lattices and x, y, z to range over their elements. We omit the ordering $\sqsubseteq$ whenever unnecessary. As usual $\bigsqcup$ and $\bigsqcap$ denote least upper bound and greatest lower bound, $\sqcup$ and $\sqcap$ denote join and meet, $\top$ and $\bot$ top and bottom.

Hereafter we always consider monotone maps so we will often omit to specify that they are monotone. Monotone maps form a complete lattice with their natural point-wise order: whenever $f, g\colon L_1 \to L_2$ then $f \sqsubseteq g$ iff for any $x \in L_1$: $f(x) \sqsubseteq_2 g(x)$. Obviously, the identity $id\colon L \to L$ and the composition $f \circ g\colon L_1 \to L_3$ of two monotone maps $g\colon L_1 \to L_2$ and $f\colon L_2 \to L_3$ are monotone.

Given a monotone map $f\colon L \to L$, $x \in L$ is said to be a *post-fixed point*[1] iff $x \sqsubseteq f(x)$ and a *pre-fixed point* iff $f(x) \sqsubseteq x$. A *fixed point* iff $x = f(x)$. Pre, post and fixed points form complete lattices, denoted by $Pre(f)$, $Post(f)$ and $Fix(f)$, respectively. We write μf and νf for the least and greatest fixed-point.

For a map $f\colon L \to L$, we inductively define $f^0 = id$ and $f^{n+1} = f \circ f^n$. We fix $f^\uparrow = \bigsqcup_{i\in\mathbb{N}} f^i$ and $f^\downarrow = \bigsqcap_{i\in\mathbb{N}} f^i$. A monotone map $f\colon L \to L$ is an *up-closure* operator if $x \sqsubseteq f(x)$ and $ff(x) \sqsubseteq f(x)$. It is a *down-closure* operator if $f(x) \sqsubseteq x$ and $f(x) \sqsubseteq ff(x)$. For any f, $f^\uparrow$ is an up-closure and $f^\downarrow$ is a down-closure [9].

[1] It is also common to find in literature the reversed definitions of post and pre-fixed point. Here we adopted the terminology of Davey and Priestley [15].

Given $l\colon L_1 \to L_2$ and $r\colon L_2 \to L_1$, we say that l is the *left adjoint* of r, or equivalently that r is the *right adjoint* of l, written $(L_1, \sqsubseteq_1) \underset{l}{\overset{r}{\leftrightarrows}} (L_2, \sqsubseteq_2)$, exactly when we have $l(x) \sqsubseteq_2 y$ iff $x \sqsubseteq_1 r(y)$ for all $x \in L_1$ and $y \in L_2$. Moreover, $r \circ l$ is an up-closure operator and $l \circ r$ a down-closure operator. When $l \circ r = id$ we say that l, r form a *Galois insertion* (GI), hereafter denoted as $(L_1, \sqsubseteq_1) \underset{l}{\overset{r}{\leftrightarrows}} (L_2, \sqsubseteq_2)$. Closure operators and Galois insertions are bijective correspondence: given an up-closure operator $f\colon L \to L$, the functions $l\colon L \to Pre(f)$, defined as $l(x) = \bigsqcap\{y \mid x \sqsubseteq y \sqsupseteq f(y)\}$ is the left adjoint of $r\colon Pre(f) \to L$, defined as $r(x) = x$.

For a monotone map $b\colon L \to L$ on a complete lattice L, the Knaster-Tarski fixed-point theorem characterises μb as the least upper bound of all pre-fixed points of b and νb as the greatest lower bound of all its post-fixed points:

$$\mu b = \bigsqcap\{x \mid b(x) \sqsubseteq x\} \qquad \nu b = \bigsqcup\{x \mid x \sqsubseteq b(x)\} \ .$$

This immediately leads to the *induction* and *coinduction* proof principles, illustrated below, on the left and on the right, respectively [26].

$$\frac{\exists x,\ b(x) \sqsubseteq x \sqsubseteq f}{\mu b \sqsubseteq f} \qquad \frac{\exists x,\ i \sqsubseteq x \sqsubseteq b(x)}{i \sqsubseteq \nu b} \tag{1}$$

Another fixed-point theorem, usually attributed to Kleene, plays an important role in our exposition. It characterises μb and νb as the least upper bound, respectively the greatest lower bound, of the chains

$$\bot \sqsubseteq b(\bot) \sqsubseteq bb(\bot) \sqsubseteq \ldots \qquad \top \sqsupseteq b(\top) \sqsupseteq bb(\top) \sqsupseteq \ldots \tag{2}$$

In short,

$$\mu b = \bigsqcup_{i\in\mathbb{N}} b^i(\bot) \qquad \nu b = \bigsqcap_{i\in\mathbb{N}} b^i(\top)$$

The assumptions are stronger than for Knaster-Tarski: for the left-most statement, it requires the map b to be *Scott-continuous* (i.e.., it preserves $\bigsqcup$ of directed chains) and, for the rightmost *Scott-cocontinuous* (similar but for $\bigsqcap$). Observe that every left adjoint is continuous (it preserves aribtrary $\bigsqcup$) and every right adjoint is cocontinuous.

Coinduction up-to can be thought as an optimisation of the principle in (1), right. Abstract interpretation as an optimisation of the chain in (2), left. In both cases, the optimisation is given by an up-closure.

3 Coinduction up-to

In order to motivate up-to techniques we illustrate how coinduction can be exploited to check language equivalence of automata.

3.1 Coinduction for Deterministic Automata

A deterministic automaton on the alphabet A is a triple (X, o, t), where X is a set of states, $o\colon X \to 2 = \{0, 1\}$ is the output function, determining if a state x is final ($o(x) = 1$) or not ($o(x) = 0$) and $t\colon X \to X^A$ is the transition function which returns the next state, for each letter $a \in A$.

Every automaton (X, o, t) induces a function $[\![-]\!]\colon X \to 2^{A^*}$ defined for all $x \in X$, $a \in A$ and $w \in A^*$ as $[\![x]\!](\varepsilon) = o(x)$ and $[\![x]\!](aw) = [\![t(x)(a)]\!](w)$. Two states $x, y \in X$ are said to be *language equivalent*, in symbols $x \sim y$, iff $[\![x]\!] = [\![y]\!]$. Alternatively, language equivalence can be defined coinductively as the greatest fixed-point

Naive (x_1, x_2)

```
(1) R := ∅;  todo := ∅
(2) insert (x1, x2) into todo
(3) while todo is not empty do
   (3.1)  extract (x1', x2') from todo
   (3.2)  if (x1', x2') ∈ R then goto (3)
   (3.3)  if o(x1') ≠ o(x2') then return false
   (3.4)  for all a ∈ A,
              insert (t(x1')(a), t(x2')(a)) into todo
   (3.5)  insert (x1', x2') into R
(4) return true
```

Figure 1. Naive algorithm checking language equivalence of states $x_1, x_2 \in X$ for a deterministic automaton (X, o, t).

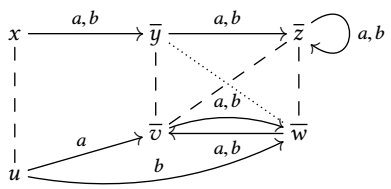
of a map b on Rel_X, the lattice of relations over X. For all $R \subseteq X^2$, $b\colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ is defined as

$$b(R) = \{(x, y) \mid o(x) = o(y) \text{ and, for all } a \in A, (t(x)(a), t(y)(a)) \in R\} \ . \quad (3)$$

Indeed, one can check that b is monotone and that $vb = \sim$.

Thanks to this characterisation, one can prove $x \sim y$ by mean of the coinduction proof principle illustrated in (1). To this end, one provides a relation R that is a *b-simulation*: a post fixed-point of b. Besides being a b-simulation, R must satisfy $\{(x, y)\} \subseteq R$.

For an example, consider the following deterministic automaton, where final states are over lined and the transition function is represented by labeled arrows. The relation consisting of dashed and dotted lines is a b-simulation showing that $x \sim u$.



(4)

Figure 1 illustrates an algorithm, called Naive, that takes in input a deterministic automaton (X, o, t) and a pair of states (x_1, x_2). It attempts to build a bisimulation R containing (x_1, x_2): if it succeeds, then $x_1 \sim x_2$ and returns true, otherwise returns false.

The worst case complexity of the algorithm Naive is linear in the size of the computed bisimulation R. Therefore, it is quadratic with respect to the number of states in X. An optimised version of Naive, the well known Hopcroft and Karp algorithm [1, 21], can be given by means of up-to techniques.

3.2 Up-to techniques

Coinduction allows to prove $i \sqsubseteq vb$ for a given map $b\colon C \to C$ on a complete lattice C and some $i \in C$. Up-to techniques have been introduced by Milner [24] as an enhancement for coinduction. In a nutshell, an *up-to technique* is a monotone map $a\colon C \to C$. A *b-simulation up to a* is a post-fixed point of ba, that is an x such that $x \sqsubseteq ba(x)$. An up-to technique a is said to be *sound* w.r.t. b (or b-sound, for short) if the following *coinduction up to* principle holds.

$$\frac{\exists x,\ i \sqsubseteq x \sqsubseteq ba(x)}{i \sqsubseteq vb} \quad (5)$$

An equivalent formulation can be given as follows.

Lemma 3.1. *a is b-sound iff $vba \sqsubseteq vb$.*

Remark 3.2 (Completeness of up-to technique)**.** Observe that, according to the above definition an up-to technique a might not be *complete*: it may exist an i such that $i \sqsubseteq vb$ for which there is no x satisfying $i \sqsubseteq x \sqsubseteq ba(x)$. However, if a is an up-closure operator, then $vb \sqsubseteq a(vb)$ and using monotonicity of b, one obtains that $i \sqsubseteq vb = b(vb) \sqsubseteq b(a(vb))$. The question of completeness for up-to techniques has never been raised because they have always been considered up-closure operators (e.g., up-to equivalence, up-to congruence). The main reason for considering arbitrary monotone maps rather than just up-closure operators, comes from the fact that the former allows for more modular proofs of their soundness. This is discussed in more details in Remark 5.6.

3.3 Hopcroft and Karp's algorithm from the standpoint of up-to techniques

For an example of up-to technique, take the function $e\colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ mapping every relation $R \subseteq X^2$ to its equivalence closure. We will see in Section 5.1, that e is sound for the map b defined in (3). A *b-simulation up to e* is a relation R such that $R \subseteq b(e(R))$. Consider the automaton in (4) and the relation R containing only the dashed lines: since $t(x)(b) = y$, $t(u)(b) = w$ and $(y, w) \notin R$, then $(x, u) \notin b(R)$. This means that R is *not* a b-simulation; however it is a b-simulation up to e, since (y, w) belongs to $e(R)$ and (x, u) to $b(e(R))$.

This example shows that b-simulations up-to e can be smaller than plain b-simulations: this idea is implicitly exploited in the Hopcroft and Karp's algorithm [1, 21] to check language equivalence of deterministic automata. This algorithm can be thought as an optimisation of Naive, where line (3.2) is replaced with the following[2].

```
(3.2)   if (x1', x2') ∈ e(R) then goto (3)
```

This optimised algorithm skips any pair which is in the equivalence closure of R: during the while loop (3), it always holds that $R \subseteq b(e(R) \cup todo)$. The algorithm returns true only if $R \subseteq be(R)$. This means that R is a b-simulation up to e containing (x_1, x_2).

This simple optimisation allows to reduce the worst case complexity of Naive: the size of the returned relation R cannot be larger than n (the number of states). The case of non-deterministic automata is even more impressive: another up-to technique, called *up-to congruence*, allows for an exponential improvement [4].

Remark 3.3. The partition refinement algorithm by Hopcroft [20] computes language equivalence for deterministic automata by constructing the chain defined in the right of (2) for the b in (3), e.g., $\top \sqsupseteq \{x, u\}\{y, v, w, z\} \sqsupseteq \{x, u\}\{y, v, w, z\}$ for the automaton in (4).

The crucial observation, for showing that this chain stabilises after at-most n iterations is that every element of the chain is an equivalence relation. Somehow, the computation of the chain for the greatest fixed point is already up-to equivalence. This fact will find a deeper explanation at the end of Section 7.2.

[2] In Section 6.1, we will se that there is a little, but significant, difference between this and the original formulation from [1].

4 Abstract Interpretation

We introduce abstract interpretation by showing a simple problem of program analysis.

4.1 A toy program analysis

Consider the following piece of code, where x is an integer value.

$x := 5;$ `while` $x > 0$ `do` $\{\ x := x - 1;\ \}$

We want to prove that after exiting the loop, x has value 0. Our analysis works on the lattice of predicates over the integers, hereafter denoted by $Pred_Z$, and makes use of the function $\ominus 1 : Pred_Z \to Pred_Z$ defined as $P \ominus 1 = \{i - 1 \mid i \in P\}$, for all $P \in \mathrm{Pred}_Z$. We start by annotating the code so to make explicit its control flow.

$x := 5;^1$ `while` $^2x > 0^3$ `do` $\{\ x := x - 1;^4\ \}^5$

We then write the following system of equations where x^j contains the set of possible values that x can have at the position j.

$$x^1{=}\{5\},\ x^2{=}x^1 \cup x^4,\ x^3{=}x^2 \cap [1,\infty),\ x^4{=}x^3 \ominus 1,\ x^5{=}x^2 \cap (-\infty, 0]$$

For x^2, we obtain the equation $x^2 = \{5\} \cup ((x^2 \cap [1,\infty)) \ominus 1)$ that has as smallest solution μb where $b : Pred_Z \to Pred_Z$ is defined as

$$b(P) = \{5\} \cup ((P \cap [1,\infty)) \ominus 1) \tag{6}$$

for all predicates P. Our initial aim is to check whether $x^5 = x^2 \cap (-\infty, 0] = \mu(b) \cap (-\infty, 0] \subseteq \{0\}$. That is $\mu b \subseteq [0,\infty)$.

We proceed by computing μb as in the left of (2):

$$\emptyset \subseteq \{5\} \subseteq \{5,4\} \subseteq \cdots \subseteq \{5,4,3,2,1\} \subseteq \{5,4,3,2,1,0\} \tag{7}$$

Since $\{5,4,3,2,1,0\} \subseteq [0,\infty)$, we have proved our conjecture.

4.2 Abstract domains

We consider the standard Galois insertion-based definition of an abstract domain [8]. Define an *abstract domain* to be a Galois insertion $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ between complete lattices C and A. Sometimes we call an abstract domain the associated up-closure, hereafter denoted by $a : C \to C$ [10]. We will always identify A with $Pre(a)$. The main idea of abstract interpretation is that in order to check whether $\mu b \sqsubseteq f$, for a map $b : C \to C$ and $f \in C$, the computation of μb via the chain in (2) can be carried more efficiently in some abstract domain A. One wants to define some $\overline{b} : A \to A$ representing an approximation of b in A and then check whether $\mu\overline{b} \sqsubseteq f$. Note that, in the latter inequality, the left hand side stands in A, while the right one in C. For this reason, it is always assumed that $f \in A = Pre(a)$, that is $a(f) \sqsubseteq f$.

An approximation $\overline{b}$ is said to be *sound* (w.r.t. b) if $\alpha b \sqsubseteq \overline{b}\alpha$.[3] The terminology is justified by the last point of the following lemma by Cousot and Cousot [10].

Lemma 4.1. *Let $b, a : C \to C$ be a map and a closure operator with associated Galois insertion $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} C$. Let $f \in Pre(a)$.*

1. *Let $x \in C$. Then $x \sqsubseteq_C f$ iff $\alpha(x) \sqsubseteq_A f$ iff $a(x) \sqsubseteq_C f$.*
2. *In particular, $\mu b \sqsubseteq_C f$ iff $\alpha(\mu b) \sqsubseteq_A f$ iff $a(\mu b) \sqsubseteq_C f$.*
3. *If $\overline{b}$ is b-sound, then $\alpha(\mu b) \sqsubseteq_A \mu\overline{b}$.*
4. *If $\overline{b}$ is b-sound, then $\mu\overline{b} \sqsubseteq_A f$ entails that $\mu b \sqsubseteq_C f$.*

For a monotone map b and an up-closure a, we define $\overline{b}^a$ as $\alpha \circ b \circ \gamma$. As explained by the following proposition, this approximation plays a key role.

[3] Soundness is often defined also by the equivalent inequation $\alpha b \gamma \sqsubseteq \overline{b}$.

Proposition 4.2 ([10], Corollary 7.2.0.4)**.** *The map $\overline{b}^a$ is* the best sound approximation*, that is: (1) $\overline{b}^a$ is sound and (2) if $\overline{b}$ is sound, then $\overline{b}^a \sqsubseteq \overline{b}$.*

Therefore, for all abstract domain a, there exists a sound approximation $\overline{b}^a$, that is $\mu\overline{b}^a \sqsubseteq_A f$ implies that $\mu b \sqsubseteq_C f$.

The converse implication is not guaranteed in general. One has to require completeness of the abstract domain: a is *complete* w.r.t. b (or b-complete, for short) iff $\alpha(\mu b) = \mu\overline{b}^a$ [10, 19]. Standard completeness in abstract interpretation is called here b-completeness.

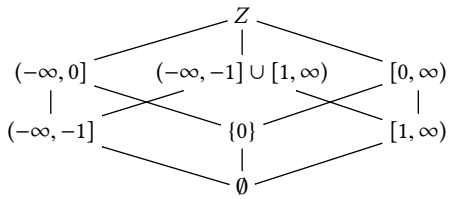
Lemma 4.3. *Let $b, a : C \to C$ be a map and a closure operator. If a is b-complete then for all $f \in Pre(a)$, $\mu\overline{b}^a \sqsubseteq_A f$ iff $\mu b \sqsubseteq_C f$.*

We will often find convenient the following alternative characterization.

Lemma 4.4 ([19], Lemma 3.1)**.** *a is b-complete iff $a(\mu b) = \mu(ab)$.*

4.3 Abstract Interpretation of a toy program

Consider Sign_Z, the abstract domain of signs depicted below: each element is a predicate over Z. The right adjoint $\gamma : \mathrm{Sign}_Z \to \mathrm{Pred}_Z$ is the obvious inclusion and the left adjoint $\alpha : \mathrm{Pred}_Z \to \mathrm{Sign}_Z$ maps any predicate P into the smallest Q in Sign_Z, such that $P \subseteq Q$. For instance, $\alpha(\{5,6\}) = [1,\infty)$. Take $s : \mathrm{Pred}_Z \to \mathrm{Pred}_Z$ as $\gamma \circ \alpha$. As recalled in Section 2, $Pre(s) = \mathrm{Sign}_Z$.



For b defined as in (6), its best sound approximation is $\overline{b}^s(P) = [1,\infty) \sqcup ((P \sqcap [1,\infty))\overline{\ominus 1}^s)$ where $\overline{\ominus 1}^s$ is again defined as $\alpha \circ \ominus 1 \circ \gamma$, e.g., $\overline{\ominus 1}^s([1,\infty)) = [0,\infty)$. The computation of $\mu\overline{b}^s$ is shorter than the one of μb in (7): $\emptyset \sqsubseteq [1,\infty) \sqsubseteq [0,\infty) \sqsubseteq [0,\infty)$. Since $\mu\overline{b}^s \subseteq [0,\infty)$ and since $\overline{b}^s$ is sound, one can conclude that $\mu b \subseteq [0,\infty)$.

Imagine now that one would like to check whether, after the while loop in the toy program, x has a negative value. It is necessary to verify that $\mu b \cap (-\infty, 0] \subseteq (-\infty, -1]$, i.e., $\mu b \subseteq (-\infty, -1]$. After computing $\mu\overline{b}^s = [0,\infty) \not\subseteq (-\infty, -1]$, one would like to conclude that the property does not hold. This deduction cannot be done in general, but only when the abstract domain is complete.

In this case, it is pretty easy to see that Sign_Z is complete: use Lemma 4.4 and observe that $s(\mu b) = s(\{5,4,3,2,1,0\}) = [0,\infty) = \mu(sb)$. However, without knowing the value of μb, proving the completeness is rather complicated. For this reason, in the next section, we will illustrate a sufficient condition entailing completeness.

5 Proving soundness and completeness

Sufficient conditions were introduced to prove soundness of up-to techniques and completeness of abstract domains. Next, we report on several equivalent formulations of such conditions.

Lemma 5.1. *Let $b : C \to C$ be a monotone map and $a : C \to C$ an up-closure operator with $C \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} A$ as associated pair of adjoint maps, i.e., $Pre(a) = A$. The followings are equivalent:*

•1 $ba = aba$;
•2 $ab \sqsubseteq ba$ (EM law);
•3 there exists a $\overline{b} \colon A \to A$ such that $\gamma\overline{b} = b\gamma$ (EM lifting).
The followings are equivalent:
∘1 $ab = aba$;
∘2 $ba \sqsubseteq ab$ (Kl law);
∘3 there exists a $\overline{b} \colon A \to A$ such that $\overline{b}\alpha = \alpha b$ (Kl extension).

These facts are well known and appear in different places in literature: the reader can find a proof in a long version [3]. We call a monotone map $a \colon C \to C$ *compatible* w.r.t. b if it enjoys the property •2 in Lemma 5.1; we call it *fully complete* w.r.t. b, if it enjoys ∘2. Compatibility entails soundness for up-to techniques, while full completeness entails completeness for abstract domains.

Theorem 5.2 ([31], Theorem 3.6.9)**.** *Let $b, a \colon C \to C$ be a map and an up-closure. If a is compatible w.r.t. b, then a is sound w.r.t. b.*

Theorem 5.3 ([10], Theorem 7.1.0.4)**.** *Let $b, a \colon C \to C$ be a map and an up-closure. If a is fully complete w.r.t. b, then a is complete w.r.t. b.*

It is important to remark here that both theorems state sufficient conditions that are not, in general, necessary: we have seen in Section 4.3 that the abstract domain of signs is complete but, as we will see in Section 5.2, it is not fully complete. Indeed, compatibility and full completeness, as characterised by points •3 and ∘3 of Lemma 5.1, require step wise correspondences between b and $\overline{b}$, visualized below,

$$\begin{array}{ccc} A & \xrightarrow{\overline{b}} & A \\ {\scriptstyle\gamma}\downarrow & & \downarrow{\scriptstyle\gamma} \\ C & \xrightarrow[b]{} & C \end{array} \qquad\qquad \begin{array}{ccc} A & \xrightarrow{\overline{b}} & A \\ {\scriptstyle\alpha}\uparrow & & \uparrow{\scriptstyle\alpha} \\ C & \xrightarrow[b]{} & C \end{array}$$

while soundness and completeness require the correspondences just of the fixed points (that can happen to hold for different reasons).

The formulations of compatibility and full completeness provided by points •2 and ∘2 of Lemma 5.1 are more handy to make modular proofs of compatibility and full-completeness, as shown in the next proposition. For compatibility, h, (h_1, h_2) in Proposition 5.4 will play the role of b, while g, (g_1, g_2) the role of the up-to technique a. For abstract domains instead, the situation is reversed: h, (h_1, h_2) will play the role of a, while g, (g_1, g_2) the role of b.

Proposition 5.4 (modularity)**.** *Let $g, h, g_1, g_2, h_1, h_2 \colon C \to C$ be monotone maps on some complete lattice C. Then:*

1. *$id \circ h \sqsubseteq h \circ id$;*
2. *if $g_1 \circ h \sqsubseteq h \circ g_1$ and $g_2 \circ h \sqsubseteq h \circ g_2$, then $(g_1 \circ g_2) \circ h \sqsubseteq h \circ (g_1 \circ g_2)$.*

Moreover:

3. *if $g_1 \circ h \sqsubseteq h \circ g_1$ and $g_2 \circ h \sqsubseteq h \circ g_2$, then $(g_1 \sqcup g_2) \circ h \sqsubseteq h \circ (g_1 \sqcup g_2)$;*
4. *if $g \circ h \sqsubseteq h \circ g$, then $g^{\uparrow} \circ h \sqsubseteq h \circ g^{\uparrow}$.*

Dually:

5. *if $g \circ h_1 \sqsubseteq h_1 \circ g$ and $g \circ h_2 \sqsubseteq h_2 \circ g$, then $g \circ (h_1 \sqcap h_2) \sqsubseteq (h_1 \sqcap h_2) \circ g$;*
6. *if $g \circ h \sqsubseteq h \circ g$, then $g \circ h^{\downarrow} \sqsubseteq h^{\downarrow} \circ g$.*

Remark 5.5. The notion of compatibility is also known in abstract interpretation as *forward completeness*, see [18], which corresponds to require that no loss of precision is introduced by approximate the range of a function in a given abstract domain. This notion have been used for generalising strong preservation to abstract interpretation-based model checking [33].

5.1 Proving soundness of equivalence closure

Recall the monotone map $b \colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ defined in (3) and the up-closure $e \colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ introduced in Section 3.3. In order to prove that the Hopcroft and Karp algorithm is sound one has to rely on the fact that e is sound w.r.t. b. Thanks to Theorem 5.2, one can prove soundness by showing that e is compatible w.r.t. b. The proof of compatibility can be made modular using Proposition 5.4.

The map b can be decomposed as $b = b_* \sqcap f$ where $b_*, f \colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ are defined for all relations R as

$$b_*(R) = \{(x, y) \mid \text{ for all } a \in A,\ (t(x)(a), t(y)(a)) \in R\} \tag{8}$$
$$f(R) = \{(x_1, x_2) \mid o(x_1) = o(x_2)\} \tag{9}$$

and the equivalence closure as $e = (id \sqcup r \sqcup s \sqcup t)^{\uparrow}$ where $r, s, t \colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ are defined as follows.

$$r(R) = \{(x, x) \mid x \in X\} \qquad s(R) = \{(y, x) \mid (x, y) \in R\}$$
$$t(R) = \{(x, z) \mid \exists y \text{ such that } (x, y) \in R \text{ and } (y, z) \in R\}$$

The proof of compatibility of e w.r.t. b can be decomposed by compatibility of e w.r.t. b_* and f and then use Proposition 5.4.5. Furthermore, to prove that e is compatible w.r.t. b_*, one can prove that r, s, t are compatible w.r.t. b_* and then use points 1,3 and 4 of Proposition 5.4. For f, it is immediate to check that $ef \sqsubseteq fe = f$, that is $f(R)$ is an equivalence relation for all $R \in \mathrm{Rel}_X$.

Remark 5.6. Proving compatibility of each of r, s, t is much simpler than proving compatibility of the whole e at once. Observe that while e is an up-closure, the maps r, s, t are *not*. As anticipated in Remark 3.2, this is the main explanation of why it is convenient to consider up-to techniques as arbitrary monotone maps, rather than just up-closures.

Remark 5.7. Some works [17, 19] have studied modularity for proofs of full-completeness of abstract domains, but always focusing on up-closures rather than on monotone maps. This example, together with the results in Section 6, shows that also for abstract domains could be convenient to decompose up-closures into smaller monotone maps. Indeed, the pointwise least-upper bound of closure operators is not necessarily a closure, but a mere monotone map. Thanks to Proposition 5.4.4, one can first makes modular proofs with monotone maps and then transforms them through the operator $(\cdot)^{\uparrow}$ into up-closures.

5.2 Completeness and the domain of signs

Recall the domain of signs in Section 4.3 and $b \colon \mathrm{Pred}_Z \to \mathrm{Pred}_Z$ defined in (6). One would like to prove that s is complete w.r.t. b by mean of Theorem 5.3, namely by proving that s is fully complete. But this approach does not work.

For later use, it is convenient to decompose b as $i \sqcup b^*$ where $i, b^* \colon \mathrm{Pred}_Z \to \mathrm{Pred}_Z$ are defined for all predicates P as follows:

$$i(P) = \{5\} \qquad b^*(P) = (P \cap [1, \infty)) \ominus 1 \ . \tag{10}$$

Observe that s is fully complete w.r.t. i, more generally any abstract domain a is fully complete with any constant function c (that is

$c \sqsubseteq a(c)$), but *not* w.r.t. b^*. To see the latter, take for instance $x = \{3\}$, and observe that $b^*s(x) = [0, \infty)$ and $sb^*(x) = [1, \infty)$, that is

$$b^*s \not\sqsubseteq sb^* \quad . \tag{11}$$

The same x shows that $(i \sqcup b^*)s \not\sqsubseteq s(i \sqcup b^*)$.

6 Relating Abstract Interpretation and Coinduction up-to by adjointness

So far, we have seen that coinduction up-to and abstract interpretation exploit a closure operator a to check, respectively, $i \sqsubseteq \nu b$ and $\mu b \sqsubseteq f$ for some $b\colon C \to C$ and $i, f \in C$. To relate them, hereafter we assume, for coinduction up-to, that $b = b_* \sqcap f$ and, for abstract interpretation, that $b = i \sqcup b^*$ where b^* and b_* are left and right adjoint. Intuitively, the elements of C represent some predicates, or conditions, i and f initial and final conditions, and b^* and b_* predicate transformers mapping a condition into, respectively, its strongest postcondition and weakest precondition. (Note that above and hereafter we implicitly identify $i, f \in C$ with the constant maps $i, f\colon C \to C$).

In this setting, the problems addressed by coinduction up-to and abstract interpretation, namely $i \sqsubseteq \nu(b^* \sqcap f)$ and $\mu(b_* \sqcup i) \sqsubseteq f$, coincide as shown by the following well-known fact.

Proposition 6.1. *Let* $C \underset{b^*}{\overset{b_*}{\leftrightarrows}} C$ *and* $i, f, x \in C$.

$$(b^* \sqcup i)(x) \sqsubseteq x \sqsubseteq f \qquad \textit{iff} \qquad i \sqsubseteq x \sqsubseteq (b_* \sqcap f)(x)$$

The result below follows immediately by Knaster-Tarski.

Corollary 6.2. $\mu(b^* \sqcup i) \sqsubseteq f$ *iff* $i \sqsubseteq \nu(b_* \sqcap f)$.

Our key observation is that, in this case, the sufficient conditions ensuring soundness of up-to techniques –compatibility– and completeness of abstract interpretation –full completeness– are closely related. Indeed, as already noticed in [19, Lemma 4.2], it is straightforward to see that whenever b^* and b_* are adjoint and a is an up-closure:

$$b^*a \sqsubseteq ab^* \quad \text{iff} \quad ab_* \sqsubseteq b_*a \quad . \tag{12}$$

Full completeness of a w.r.t. $i \sqcup b^*$ amounts to $(b^* \sqcup i)a \sqsubseteq a(b^* \sqcup i)$ which, by Proposition 5.4.3, is entailed by

$$b^*a \sqsubseteq ab^* \quad \text{and} \quad i \sqsubseteq ai \quad . \tag{13}$$

Compatibility of a w.r.t. $b_* \sqcap f$ amounts to $a(b_* \sqcap f) \sqsubseteq (b_* \sqcap f)a$ which, by Proposition 5.4.5, is entailed by

$$ab_* \sqsubseteq b_*a \quad \text{and} \quad af \sqsubseteq f \quad . \tag{14}$$

Observe that there is a slight asymmetry in (13) and (14): the condition $i \sqsubseteq ai$ is guaranteed for any $i \in C$, since a is an up-closure. This is not the case for $af \sqsubseteq f$. But the latter condition is anyway necessary to make abstract interpretation meaningful (see e.g. Lemma 4.1). In this setting, whenever a satisfies one of the two equivalent formulations of (12), a can be regarded as both a complete abstract domain for $(i \sqcup b^*)$ and a sound up-to technique for $(b_* \sqcap f)$. This discussion is summarised below.

Assumption 6.3. Consider: (*i*) a complete lattice C, (*ii*) a pair of adjoint $C \underset{b^*}{\overset{b_*}{\leftrightarrows}} C$, (*iii*) a closure operator $a\colon C \to C$, (*iv*) an element $i \in C$, and (*v*) an element $f \in Pre(a)$.

Theorem 6.4. *Under Assumption 6.3, if* a *satisfies one of the two equivalent formulations of* (12)*, then* a *is both a complete abstract domain for* $(i \sqcup b^*)$ *and a sound up-to technique for* $(b_* \sqcap f)$.

6.1 Hopcroft and Karp from the standpoint of abstract interpretation

We now show how the Hopcroft and Karp algorithm [1, 21] can be seen as an instance of complete abstract interpretation, using the technology developed above.

Recall from Section 5.1 that $b\colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ in (3) can be decomposed as $b = b_* \sqcap f$ for b_* and f as in (8) and in (9). The map b_* has a left adjoint $b^*\colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ defined for all relations R as

$$b^*(R) = \{(x_1', x_2') \mid \exists (x_1, x_2) \in R, a \in A \text{ such that } t(x_1)(a) = x_1' \text{ and } t(x_2)(a) = x_2'\} \quad . \tag{15}$$

To sum up we have: $(\mathrm{Rel}_X, \subseteq) \underset{b^*}{\overset{b_*}{\leftrightarrows}} (\mathrm{Rel}_X, \subseteq)$.[4]

We take i as $\{(x_1, x_2)\}$, i.e., the states to prove to be language equivalent. By Corollary 6.2, checking $\{(x_1, x_2)\} \sqsubseteq \sim\ = \nu(b_* \sqcap f)$ is equivalent to checking $\mu(i \sqcup b^*) \sqsubseteq f$. This inequality has a rather intuitive meaning: $\mu(i \sqcup b^*)$ is the sets of pairs of states that are "reachable" from the initial pair i. Clearly, $x_1 \sim x_2$ iff each of these pairs of states is in f.

In Section 5.1, we have shown that the equivalence closure $e\colon \mathrm{Rel}_X \to \mathrm{Rel}_X$ is a sound up-to technique, by proving that $eb_* \sqsubseteq b_*e$ and $ef \sqsubseteq f$. By (12), $b^*e \sqsubseteq eb^*$ and since $i \sqsubseteq ei$, by Proposition 5.4.3, one has that $(i \sqcup b^*)e \sqsubseteq e(i \sqcup b^*)$, that is e is fully complete w.r.t. $(i \sqcup b^*)$. Therefore e is a complete abstract domain for $(i \sqcup b^*)$ and $f \in Pre(e)$.

This provides a novel perspective on the Hopcroft and Karp's algorithm [1]. Its correctness can be established using the least fixed-point of the function $b^* \sqcup i$ abstracted to the lattice of equivalence relations $ERel_X$. We denote this function by $\overline{b^* \sqcup i}^e\colon ERel_X \to ERel_X$ and $(Rel_X, \subseteq) \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} (ERel_X, \subseteq)$ the pair of adjoint associated to e. The map α assigns to every relation its equivalence closure; γ is just the obvious injection. The function $\overline{b^* \sqcup i}^e$ is given by $\alpha \circ (b^* \sqcup i) \circ \gamma$. The algorithm returns true iff $\mu(\overline{b^* \sqcup i}^e) \sqsubseteq f$ as we show later on.

This leads to a slightly different algorithm, illustrated in Fig 2, than the one discussed in Section 3.3: during the while loop (3), it is not checked whether $o(x_1') \neq o(x_2')$ (step (3.3) in Fig 1), but this is done only at the very end for the computed relation R (step (4) in Fig 2). Moreover, after every iteration of the while loop in Fig 2, R is an equivalence relation, while in the other algorithm R is always a mere relation.

Remark 6.5. The original algorithm by Hopcroft and Karp [1, 21] is actually the one in Fig 2: indeed, they use the so called *union-find* data structure for the *equivalence* relation R and they check containment in f only at the end.

To be completely formal, we must say that the algorithm does not compute exactly the chain for $\mu(\overline{b^* \sqcup i}^e)$.

$$\bot \sqsubseteq \overline{b^* \sqcup i}^e(\bot) \sqsubseteq \overline{b^* \sqcup i}^e(\overline{b^* \sqcup i}^e(\bot)) \sqsubseteq \cdots$$

Indeed, at every iteration of the algorithm, only *one* pair of states is removed from *todo* and inserted into R, while in the above chain *many* pairs are added at the same time. However, the final result, i.e., the relation R at step (4), hereafter denoted as $R^{(4)}$ is exactly $\mu(\overline{b^* \sqcup i}^e)$. From this fact and the fact the e is a complete abstract domain for $b^* \sqcup i$ it follows that the algorithm is sound and complete.

[4] This is similar to the $post$ and $\widetilde{pre}$ operator given by Cousot [7, Example 3]

HK (x_1, x_2)

```
(1) R := ∅;  todo := ∅
(2) insert (x1, x2) into todo
(3) while todo is not empty do
   (3.1)  extract (x1', x2') from todo
   (3.2)  if (x1', x2') ∈ R then goto (3)
   (3.3)  for all a ∈ A,
              insert (t(x1')(a), t(x2')(a)) into todo
   (3.4)  insert (x1', x2') into R
   (3.5)  R := e(R);
(4) return R ⊆ f;
```

Figure 2. Hopcroft and Karp's algorithm [1].

In order to prove that $R^{(4)} = \mu(\overline{b^* \sqcup i}^e)$, we first show that $R^{(4)}$ is a fixed point of $\overline{b^* \sqcup i}^e$. Observe that at step (3), it always holds that

$$\overline{b^* \sqcup i}^e(R) = e(R \sqcup todo) \ . \tag{16}$$

This is true after step (2): $R = \emptyset$ and $todo = i$, so $\overline{b^* \sqcup i}^e(\emptyset) = e(b^*(\emptyset) \sqcup i) = e(i) = e(\emptyset \sqcup todo)$. At any iteration, a pair (x_1', x_2') is removed from $todo$ and, if it already belongs to R, the control comes back to (3): in this case (16) is not modified. If it does not, (x_1', x_2') is inserted in R and exactly $b^*(\{(x_1', x_2')\})$ is inserted in $todo$: in this case we need to check that

$$\overline{b^* \sqcup i}^e(R \sqcup \{(x_1', x_2')\}) = e(R \sqcup todo \sqcup b^*(\{(x_1', x_2')\})) \tag{17}$$

It is easy to see that (16) entails (17):

$$\begin{aligned}
&\overline{b^* \sqcup i}^e(R \sqcup \{(x_1', x_2')\}) \\
=&e(b^* \sqcup i(R \sqcup \{(x_1', x_2')\})) && \text{by definition of } \overline{b^* \sqcup i}^e \\
=&e(b^* \sqcup i(R) \sqcup b^*(\{(x_1', x_2')\})) && b^* \text{ is a left adjoint} \\
=&e(e(R \sqcup todo) \sqcup b^*(\{(x_1', x_2')\})) && \text{by (16)} \\
=&e(R \sqcup todo \sqcup b^*(\{(x_1', x_2')\})) && e \text{ is a closure}
\end{aligned}$$

Now, at step (4), $todo$ is empty and, thus by (16), we have that $\overline{b^* \sqcup i}^e(R^{(4)}) = e(R^{(4)}) = R^{(4)}$. This proves that $R^{(4)}$ is a fixed point of $\overline{b^* \sqcup i}^e$.

To prove that $R^{(4)} = \mu\overline{b^* \sqcup i}^e$, is now enough to show that $R^{(4)} \sqsubseteq \mu\overline{b^* \sqcup i}^e$. Let R_j and $todo_j$ be the relations at step (3) at the j-th iteration. Then, a simple inductive argument confirms that $todo^j \sqsubseteq (\overline{b^* \sqcup i}^e)^{j+1}(\bot)$ and $R_j \sqsubseteq (\overline{b^* \sqcup i}^e)^j(\bot)$. Therefore $R^{(4)} = \bigsqcup R_j \sqsubseteq \bigsqcup (\overline{b^* \sqcup i}^e)^j(\bot) = \mu(\overline{b^* \sqcup i}^e)$.

6.2 The domain of signs as an up-to technique

Recall the toy program from Section 4.1. One needs to check whether $\mu(i \sqcup b^*) \sqsubseteq f$ where i and b^* are as in (10) and $f = [0, \infty)$. The right adjoint of b^* is $b_* : Pred_Z \to Pred_Z$ defined for all predicates P as

$$b_*(P) = \bigcup \{Q \mid b^*(Q) \subseteq P\} = ((-\infty, 0] \cup P) \oplus 1 \tag{18}$$

where $\oplus 1 : Pred_Z \to Pred_Z$ is defined as $P \oplus 1 = \{i + 1 \mid i \in P\}$.

Thanks to Corollary 6.2, rather than checking $\mu(i \sqcup b^*) \sqsubseteq f$, one can check $i \sqsubseteq \nu(b_* \sqcap f)$. The latter can be proved by means of coinduction: one has to find a predicate P such that $\{5\} \subseteq P \subseteq b_*(P) \cap [0, \infty)$. For instance, by taking $P = \{5, 4, 3\}$, one has $b_*(P) = (-\infty, 1] \cup \{6, 5, 4\}$ and $b_*(P) \cap [0, \infty) = \{6, 5, 4, 1, 0\}$. Therefore the inclusion does not hold. In order to find a $(b_* \sqcap f)$-simulation P, one can take the least fixed point computed in (7), that is $P = \{5, 4, 3, 2, 1, 0\}$.

One can also reason, more effectively, up-to the abstract domain of signs s (Section 4.3). In this case, $\{5\}$ itself is a $(b_* \sqcap f)$-simulation up to s. Indeed, $s(\{5\}) = [1, \infty)$ and $b_*[1, \infty) \cap f = (((-\infty, 0] \cup [1, \infty)) \oplus 1) \cap [0, \infty) = [0, \infty)$. Obviously $\{5\} \subseteq [0, \infty)$.

To make this a valid proof, one should show first that s is a sound up-to technique. Unfortunately (11) and (12) inform us that $sb_* \not\sqsubseteq b_* s$, namely s is not b_*-compatible. Note that this does not entail that s is not $(b_* \sqcap f)$-compatible, but by taking $x = \{-3\}$ one can easily verify that this is the case, i.e.,

$$s(b_* \sqcap f) \not\sqsubseteq (b_* \sqcap f)s \ .$$

Nevertheless, s is sound w.r.t. $(b_* \sqcap f)$: we will show this in Section 8.

7 Intermezzo

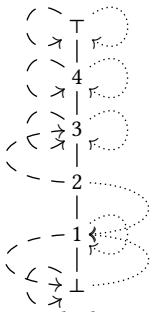
Before continuing with the next achievements, we make two small detours to settle down the concepts seen so far.

7.1 A counterexample to the correspondence of soundness and completeness

In Section 6, we have shown that the conditions in (12) entails both soundness of up-to techniques and completeness of abstract domains. The reader may wonder whether, more generally, it is the case that an up-to technique is sound iff it is a complete abstract domain.

More formally, given Assumption 6.3 is it the case that a is $(b_* \sqcap f)$-sound (as an up-to technique) iff it is $(i \sqcup b^*)$-complete (as an abstract domain)?

The answer is no. Consider the following lattice, with b^* defined by the dashed lines on the left and b_* defined by the dotted lines on the right. It is easy to check that they are adjoint. Take $i = 1$ and $f = 4$.



Let a be the up-closure such that $Pre(a) = \{\top, 4, 3, 2\}$. Then $\mu(i \sqcup b^*) = 1$ and $a(\mu(i \sqcup b^*)) = 2$. Instead $\mu a(i \sqcup b^*) = 3$. Indeed,

$$\bot \sqsubseteq a(i \sqcup b^*)(\bot) = 2 \sqsubseteq a(i \sqcup b^*)(2) = 3 \sqsubseteq a(i \sqcup b^*)(3) = 3 \ .$$

Therefore, by Lemma 4.4, a is not complete w.r.t. $i \sqcup b^*$. However, a is $(b_* \sqcap f)$-sound. Indeed $\nu(b_* \sqcap f)$ is computed as

$$\top \sqsupseteq 4 \sqsupseteq 4 \tag{19}$$

and, similarly, $\nu(b_* \sqcap f)a = 4$.

7.2 Duality

The reader may have got the feeling that coinduction up-to and abstract interpretation are somehow the dual of each other. This is not the case: first, coinduction up-to is a proof technique, exploiting

the Knaster-Tarski fixed point theorem, while abstract interpretation is a computational method relying on Kleene's theorem; second both abstract interpretation and coinduction up-to use as enhancement an up-closure $a\colon C \to C$, while their duals should use down-closures. The latter is explained in some details, below.

The dual of the coinduction up-to looks like

$$\frac{\exists y,\ ba(y) \sqsubseteq y \sqsubseteq x}{\mu b \sqsubseteq x} \tag{20}$$

When a is an up-closure, this principle does not provide any enhancement w.r.t. standard induction (see (1), left): indeed, if $ba(y) \sqsubseteq y$, then also $b(y) \sqsubseteq y$. Instead, when a is an down-closure, the principle might be meaningful.

The dual of abstract interpretation consists in checking $\nu b \sqsubseteq f$ by optimising somehow the computation of the chain of νb in the right of (2). Interestingly enough, all the elements of this chains already belongs to the domain $Pre(a)$, whenever a is a fully-complete up-closure.

Proposition 7.1. *Let $C \underset{b^*}{\overset{b_*}{\leftrightarrows}} C$ and $i, f \in C$. Let $a\colon C \to C$ be an up-closure fully complete w.r.t. b^*. Assume moreover that $a(f) \sqsubseteq f$. For all k,*

$$a(b_* \sqcap f)^k(\top) \sqsubseteq (b_* \sqcap f)^k(\top) \ .$$

This provides an explanation for what we anticipated in Remark 3.3. Indeed, we have seen in Section 5.1 that the equivalence closure $e\colon \mathsf{Rel}_X \to \mathsf{Rel}_X$ is fully complete w.r.t. b_* in (8); the above proposition states that all the elements of the chain (2) for computing $\nu(b_* \sqcap f)$ are in $Pre(e)$, i.e., they are equivalence relations.

It is worth to conclude this detour on duality, by remarking that while abstract interpretation and coinduction up-to naturally emerges in logics, computer science and related fields, their duals, exploiting a down-closure operator, are far less common.

8 The companion

In Section 6.2, we have seen that the domain of signs s is not compatible w.r.t. $(b_* \sqcap f)$. Nevertheless, we will see at the end of this section that s is sound. The strategy that we are going to use to prove this fact exploits recent developments in up-to techniques [22, 27, 29] that, in the next section we will transfer to abstract interpretation. The proof strategy is based on the following observations:

1. The class of sound up-to techniques is downward closed: if $a_1 \sqsubseteq a_2$ and a_2 is b-sound, then also a_1 is b-sound.
2. Fixed a b, there exists a greatest b-compatible up-to technique ω_b, which Pous [29] calls *the companion*.

Therefore, rather than proving that a certain up-to technique a is compatible, to show the soundness of a is enough to prove that $a \sqsubseteq \omega_b$. This is extremely useful because there are many techniques which are not compatible, but still they are below the companion (and thus sound), like for instance the domain of signs from Section 6.2 or many of the so called respectful techniques [35] which are common in process calculi and GSOS specifications.

Interestingly enough, ω_b is an up-closure also when one considers as up-to techniques arbitrary monotone maps, rather than just up-closure operators (see Lemma 3.2 [29]). This fact allows us to give an alternative characterisation of ω_b as an abstract domain Ω_b which we found suggestive and useful (at least in our examples), but that can be easily derived from the results of Pous [29]. We first need the following well-known lemma from [37].

Lemma 8.1. *Let $a_1, a_2\colon C \to C$ be two up-closures. $a_1 \sqsubseteq a_2$ iff $Pre(a_2) \subseteq Pre(a_1)$.*

Theorem 8.2. *Let $b\colon C \to C$ be a Scott cocontinuous map[5] and let $\omega_b\colon C \to C$ be the closure operator associated to the sublattice Ω_b, defined as follows.*

$$\top \sqsupseteq b(\top) \sqsupseteq bb(\top) \sqsupseteq \cdots \sqsupseteq \nu b$$

Then ω_b is the greatest b-compatible map, that is (1) ω is compatible and (2) if a is compatible, then $a \sqsubseteq \omega_b$. Moreover, (3) for any up-closure a, $a \sqsubseteq \omega_b$ iff $\Omega_b \subseteq Pre(a)$.

The theorem helps in understanding the difference between being compatible and being below the companion. By point •3 of Lemma 5.1, a is compatible iff $(A =)Pre(a)$ is closed by b, that is for all $x \in Pre(a)$, $b(x) \in Pre(a)$. Being below the companion means instead that just Ω_b should be included into $Pre(a)$. This latter condition is obviously much weaker, but still is enough to entail soundness.

Corollary 8.3. *If $b^i(\top) \in Pre(a)$ for all $i \in \mathbb{N}$, then a is b-sound.*

This corollary is not particularly useful to prove soundness, since the premise is often hard to check. However, this is enough for the purposes of our paper. The condition of being below the companion could be better checked by defining the companion itself as the greatest fixed point of a certain "second order" operator and then use again coinduction. We stop here, as this goes beyond the scope of this paper, and we refer the interested reader to the work of Pous [29]. It is however important to remark here that, in Section 9.1, we will give a coinductive characterization for an analogous of the companion in the context of abstract interpretation.

Example 8.4. We conclude this section, by showing that the domain of signs s is a sound up-to technique for $(b_* \sqcap f)$. In this case, it is easy to compute Ω_b:

$$Z \sqsupseteq [0, \infty)$$

Since this is included into $Pre(s)$, which is the domain in Section 4.3, then by Corollary 8.3, s is sound. Note instead that the domain of signs $Pre(s)$ is *not* closed under $b_* \sqcap f$: this means exactly that s is not $b_* \sqcap f$-compatible.

Remark 8.5. The existence of the smallest abstract domain (or equivalently by virtue of Lemma 8.1 the greatest up-closure) that is fully complete w.r.t. $i \sqcup b^*$ is irrelevant for abstract interpretation because it is always the abstract domain containing only $\top$ that, obviously, does not contain the property f which needs to be checked. However, it makes sense to look for, amongst all the abstract domains containing f, the smallest fully complete one. The f-companion that we will introduce in the next section is the smallest abstract domain (or equivalently the largest up-closure) containing f that is fully complete w.r.t. b^* (by Proposition 5.4.3 this is also fully complete w.r.t. $i \sqcup b^*$).

[5] The assumption of Scott co-continuity can be removed to the price of a more elaborated characterization of Ω_b.

9 Local Completeness

Inspired by up-to techniques, we give a novel definition of completeness, called local completeness. This notion is strictly weaker than completeness, but still is sufficient to solve the original problem of program analysis, namely to check whether $\mu b \sqsubseteq f$ for a given property f and predicate transformer b.

Definition 9.1. Let C be a complete lattice, $b\colon C \to C$ be a monotone map and $f \in C$. We say that an up-closure $a\colon C \to C$ is *local complete*, or (b, f)*-complete*, iff (*1*) $a(f) \sqsubseteq f$ and (*2*) $\mu(ab) \sqsubseteq f$ iff $\mu b \sqsubseteq f$.

Our interest in (b, f)-completeness is justified by the following result, stating that, rather than checking $\mu b \sqsubseteq_C f$, one can safely lift b to the abstract domain $A = Pre(a)$ and check whether $\mu \overline{b}^a \sqsubseteq_A f$.

Proposition 9.2. *If a is (b, f)-complete, then $\mu \overline{b}^a \sqsubseteq_A f$ iff $\mu b \sqsubseteq_C f$.*

We named (b, f)-completeness also local completeness since, as illustrated by the following result, it is similar to completeness but localised at f.

Proposition 9.3. *Let C be a complete lattice, $b, a\colon C \to C$ be a monotone map and a an up-closure. Then:*

$$\begin{array}{c} a \text{ is } b\text{-complete} \\ \textit{iff} \\ \textit{for all } f \in Pre(a),\ a \text{ is } (b, f)\text{-complete.} \end{array}$$

Example 9.4. Consider a, b^*, i and the lattice in Section 7.1 and recall that a is not $(i \sqcup b^*)$-complete. Now for $f \in \{\top, 4, 3\}$, it is immediate to see that a is $(i \sqcup b^*, f)$-complete. Instead, for $f = 2$, it is not: $v(i \sqcup b^*) = 1$, while $va(i \sqcup b^*) = 3$.

In Section 8, we have seen that the class of sound up-to techniques is downward closed. This is not the case with the standard definition of completeness for abstract domains (Example 9.5) but it holds for local completeness (Proposition 9.6).

Example 9.5. Recall from Example 9.4 that a is not $(i \sqcup b^*)$-complete. Now take a' to be the up-closure such that $Pre(a') = \{\top, 4, 3\}$. In this case, one has that $a'(\mu(i \sqcup b^*)) = 3 = \mu a'(i \sqcup b^*)$, i.e., a' is $(i \sqcup b^*)$-complete. In this case $a \sqsubseteq a'$.

Proposition 9.6. *Let C be a complete lattice and $b, a_1, a_2\colon C \to C$ be a monotone map and two up-closures such that $a_1 \sqsubseteq a_2$. Let $f \in C$. If a_2 is (b, f)-complete, then a_1 is (b, f)-complete.*

This property makes the proof of local completeness much easier than those of completeness. Indeed, for the latter it is enough to prove full completeness, while for the former it is enough to prove to be below some fully complete domain (Theorem 9.10). This is similar to what happens with the companion for up-to techniques. However, the small asymmetry of i and f discussed in Section 6 forces us to consider a little variation of the notion of companion.

Definition 9.7. Let C be a complete lattice, $b\colon C \to C$ be a monotone map and $f \in C$. A monotone map $a\colon C \to C$ is (b, f)*-compatible*, iff (*1*) $a(f) \sqsubseteq f$ and (*2*) $ab \sqsubseteq ba$. The f*-companion* of b is

$$\omega_{b,f} = \bigsqcup \{a \mid a \text{ is } (b, f)\text{-compatible}\} \ .$$

In the above definition the least upper bound is taken in the lattice of monotone functions. However $\omega_{b,f}$ is guaranteed to be an up-closure which, additionally, is (b, f)-compatible.

Proposition 9.8. *The following holds:*

1. $\omega_{b,f} b \sqsubseteq b\omega_{b,f}$,
2. $\omega_{b,f}(f) \sqsubseteq f$,
3. $x \sqsubseteq \omega_{b,f}(x)$ *for all* $x \in C$,
4. $\omega_{b,f}(\omega_{b,f}(x)) \sqsubseteq \omega_{b,f}(x)$ *for all* $x \in C$.

Observe that (b, f)-compatibility entails $(b \sqcap f)$-compatibility by Proposition 5.4.5, but the converse does not hold in general. We need this stronger notion of compatibility because, under Assumption 6.3, $(b_* \sqcap f)$-compatibility alone does not allow to deduce $(i \sqcup b^*)$-completeness. Instead, for (b_*, f)-compatibility, this follows immediately from Theorem 6.4.

Corollary 9.9. *Let $C \underset{b^*}{\overset{b_*}{\leftrightarrows}} C$ be a pair of adjoint, $a\colon C \to C$ be an up-closure and $i, f \in C$. If a is (b_*, f)-compatible, then a is $(i \sqcup b^*)$-complete, hence also $(i \sqcup b^*, f)$-complete.*

The second part of the statement follows from Proposition 9.3. Next, we combine Proposition 9.6, Proposition 9.8 and Corollary 9.9 to obtain the main result of this section.

Theorem 9.10. *Let $C \underset{b^*}{\overset{b_*}{\leftrightarrows}} C$ be a pair of adjoint, $a\colon C \to C$ be an up-closure and $i, f \in C$. If $a \sqsubseteq \omega_{b_*, f}$, then a is $(i \sqcup b^*, f)$-complete.*

It is worth to visualise the difference between $\omega_{b_* \sqcap f}$ and $\omega_{b_*, f}$ in terms of the associated abstract domains. Under the assumptions of Theorem 8.2, $\Omega_{b_* \sqcap f}$ is the sublattice of C (consisting of a chain) given by

$$\top \sqsupseteq (b_* \sqcap f)(\top) \sqsupseteq (b_* \sqcap f)(b_* \sqcap f)(\top) \sqsupseteq \cdots \sqsupseteq \nu(b_* \sqcap f)$$

that, since $b_*(\top) = \top$ and $b_*(x \sqcap y) = b_* x \sqcap b_* y$, coincides with

$$\top \sqsupseteq f \sqsupseteq b_*(f) \sqcap f \sqsupseteq b_* b_*(f) \sqcap b_*(f) \sqcap f \sqsupseteq \cdots \sqsupseteq \nu(b_* \sqcap f).$$

Instead $\Omega_{b_*, f}$ is the smallest meet-complete sublattice of C containing

$$\top \qquad f \qquad b_*(f) \qquad b_* b_*(f) \qquad b_* b_* b_*(f) \qquad \ldots \tag{21}$$

Corollary 9.11. *If $b_*^j(f) \in Pre(a)$ for all $j \in \mathbb{N}$, then a is $(i \sqcup b^*, f)$-complete.*

Example 9.12. Recall the abstract domain of signs Sign_Z, b_* defined in (18) and $f = [0, \infty)$. We know that s is not (b_*, f)-compatible because of (11). However, since $b_*(f) = Z$, $\Omega_{b_*, f}$ is just the complete lattice $Z \sqsupseteq [0, \infty)$. Therefore s is below the f-companion.

9.1 A coinductive characterization of the f-companion

As mentioned in Section 8, the companion enjoys a coinductive characterization that is useful to prove by "second order coinduction" soundness of up-to techniques. We conclude this section by briefly showing that a similar characterization can be given for the f-companion in order to prove local completeness of abstract domains. Our argument is a tiny variation of Section 6 in [29].

Definition 9.13. Let $[C \to C]$ be the complete lattice of monotone maps on C. The function $B\colon [C \to C] \to [C \to C]$ is defined for all $a\colon C \to C$ as

$$B(a) = \bigsqcup \{c \mid cb \sqsubseteq ba, c(f) \sqsubseteq f\} \ .$$

Lemma 9.14. *B is monotone and for all functions $a, a'\colon C \to C$,*

$$a' \sqsubseteq B(a) \text{ iff } a'b \sqsubseteq ba \text{ and } a'(f) \sqsubseteq f.$$

This means that a is (b, f)-compatible iff $a \sqsubseteq B(a)$, that is a is a post-fixed point of B. By the Knaster-Tarski fixed point theorem, one has immediately the following result.

Theorem 9.15. $\omega_{b,f} = \nu B$

10 Conclusion

In this paper we studied the relationship existing in between sound up-to techniques and complete abstract domains. In general, the two concepts do not coincide (Section 7.1) but, under reasonable assumptions (Assumption 6.3), the sufficient conditions that are commonly used to prove soundness of up-to techniques –compatibility– and completeness of abstract domains –full completeness– are equivalent (Theorem 6.4). This allows to look at fully complete abstract domains as sound up-to techniques and, vice versa, to look at compatible up-to techniques as complete abstract domains. As an example of the latter, we have shown that the Hopcroft and Karp's algorithm [1, 21], which was recently observed to rely on up-to techniques [4], can also be studied from the viewpoint of complete abstract interpretation.

We hope that our observation can lead to a fruitful cross-fertilisation amongst two areas that, so far, have developed their own technologies independently. As a proof of concept for this technology transfer, we have shown that recent developments in up-to techniques [29] lead to a weaker notion of completeness, called local completeness, that is enough to ensure that if a certain property is not satisfied in the abstract domain, then it does not hold in the concrete one. Interestingly enough, local completeness can be proved by means of coinduction. As a short term application, we mention that, to prove completeness of an abstract domains for a certified abstract interpreter (see e.g. [2, 23, 36]) one could, thanks to our work, reuse one of the many available libraries for up-to techniques that have been developed in different proof assistants (see e.g., [14, 30]).

We leave as a future work the connection with domain completion techniques [13, 19, 32] which, intuitively, define strategies to enrich an abstract domain with new values as long as it is not precise enough to prove a given property. The correspondence between completeness in abstract interpretation and soundness in up-to techniques can also motivate the extension of methods for proving the absence of false alarms in abstract interpretations, such as the proof system in [17], to prove soundness of corresponding up-to techniques.

References

[1] Alfred V. Aho and John E. Hopcroft. 1974. *The Design and Analysis of Computer Algorithms* (1st ed.). Addison-Wesley Longman Publishing Co., Inc.

[2] Yves Bertot. 2009. Structural abstract interpretation: A formal study using Coq. In *Language Engineering and Rigorous Software Development*. LNCS, Vol. 5520. Springer, 153–194. https://doi.org/10.1007/978-3-642-03153-3_4

[3] Filippo Bonchi, Pierre Ganty, Roberto Giacobazzi, and Dusko Pavlovic. 2018. Sound up-to techniques and Complete abstract domains. *arXiv preprint arXiv:1804.10507* (2018).

[4] Filippo Bonchi and Damien Pous. 2013. Checking NFA equivalence with bisimulations up to congruence. In *POPL: 40th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 457–468.

[5] Didier Caucal. 1990. Graphes canoniques de graphes algébriques. *ITA* 24, 4 (1990), 339–352. http://www.numdam.org/item?id=ITA_1990__24_4_339_0

[6] Patrick Cousot. 1997. Types as abstract interpretations. In *POPL: 24th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 316–331. https://doi.org/10.1145/263699.263744

[7] Patrick Cousot. 2000. Partial Completeness of Abstract Fixpoint Checking. In *SARA: Int. Symp. on Abstraction, Reformulation, and Approximation*. Springer, 1–25. https://doi.org/10.1007/3-540-44914-0_1

[8] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL: 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 238–252.

[9] Patrick Cousot and Radhia Cousot. 1979. A constructive characterization of the lattices of all retractions, preclosure, quasi-closure and closure operators on a complete lattice. *Portug. Math.* 38, 2 (1979), 185–198.

[10] Patrick Cousot and Radhia Cousot. 1979. Systematic design of program analysis frameworks. In *POPL: 6th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 269–282.

[11] Patrick Cousot and Radhia Cousot. 1999. Refining Model Checking by Abstract Interpretation. *Automated Software Engineering* 6, 1 (1999), 69–95.

[12] Patrick Cousot, Radhia Cousot, Jérôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2005. The ASTRÉE Analyzer. In *ESOP: European Symposium on Programming (LNCS)*, Vol. 3444. Springer, 21–30.

[13] Patrick Cousot, Pierre Ganty, and Jean-François Raskin. 2007. Fixpoint-Guided Abstraction Refinements. In *SAS: 14th Int. Static Analysis Symp. (LNCS)*, Vol. 4634. Springer, 333–348. https://doi.org/10.1007/978-3-540-74061-2_21

[14] Nils Anders Danielsson. 2017. Up-to Techniques Using Sized Types. *Proc. ACM Program. Lang.* 2, POPL (2017), 43:1–43:28. https://doi.org/10.1145/3158131

[15] B. A. Davey and H. A. Priestley. 2002. *Introduction to Lattices and Order* (2 ed.). Cambridge University Press. https://doi.org/10.1017/CBO9780511809088

[16] Manuel Fähndrich and Francesco Logozzo. 2011. Static Contract Checking with Abstract Interpretation. In *Int. Conf. on Formal Verification of Object-oriented Software (FoVeOOS'10)*. Springer-Verlag, Berlin, Heidelberg, 10–30.

[17] Roberto Giacobazzi, Francesco Logozzo, and Francesco Ranzato. 2015. Analyzing Program Analyses. In *POPL: 42nd ACM SIGACT-SIGPLAN symposium on Principles of programming languages*. ACM, 261–273.

[18] Roberto Giacobazzi and Elisa Quintarelli. 2001. Incompleteness, Counterexamples, and Refinements in Abstract Model-Checking. In *SAS: 8th Int. Static Analysis Symposium (LNCS)*, Vol. 2126. Springer, 356–373.

[19] Roberto Giacobazzi, Francesco Ranzato, and Francesca Scozzari. 2000. Making abstract interpretations complete. *J. ACM* 47, 2 (2000), 361–416.

[20] John Hopcroft. 1971. *An N Log N Algorithm for Minimizing States in a Finite Automaton.* Technical Report. Stanford Univ Calif Dept of Computer Science.

[21] John E. Hopcroft and Richard M. Karp. 1971. *A Linear Algorithm for Testing Equivalence of Finite Automata.* Technical Report 114. Cornell Univ.

[22] Chung-Kil Hur, Georg Neis, Derek Dreyer, and Viktor Vafeiadis. 2013. The Power of Parameterization in Coinductive Proof. In *POPL: 40th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. ACM, 193–206.

[23] Xavier Leroy. 2014. Formal verification of a static analyzer: abstract interpretation in type theory. In *Types-The 2014 Types Meeting*.

[24] Robert Milner. 1989. *Communication and Concurrency.* Prentice Hall.

[25] Peter O'Hearn. 2015. From Categorical Logic to Facebook Engineering. In *LICS: 30th Annual ACM/IEEE Symposium on Logic in Computer Science*. IEEE, 17–20.

[26] David M.R. Park. 1969. Fixpoint induction and proofs of program properties. In *Machine Intelligence*, Vol. 5. Edinburgh Univ. Press, 59–78.

[27] Joachim Parrow and Tjark Weber. 2016. The largest respectful function. *arXiv preprint arXiv:1605.04136* (2016).

[28] Damien Pous. 2007. Complete Lattices and Up-To Techniques. In *APLAS: Asian Symposium on Programming Languages and Systems (LNCS)*, Vol. 4807. Springer, 351–366. https://doi.org/10.1007/978-3-540-76637-7_24

[29] Damien Pous. 2016. Coinduction all the way up. In *LICS: 31st Annual ACM/IEEE Symposium on Logic in Computer Science*. ACM, 307–316.

[30] Damien Pous. 2016. Coq libraries for "Coinduction all the way up". http://perso.ens-lyon.fr/damien.pous/cawu/. (2016).

[31] Damien Pous and Davide Sangiorgi. 2012. Enhancements of the bisimulation proof method. In *Advanced Topics in Bisimulation and Coinduction*. Cambridge University Press, 233–289.

[32] Francesco Ranzato, Olivia Rossi Doria, and Francesco Tapparo. 2008. A Forward-Backward Abstraction Refinement Algorithm. In *VMCAI: 9th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*. LNCS, Vol. 4905. Springer, 248–262. https://doi.org/10.1007/978-3-540-78163-9_22

[33] Francesco Ranzato and Francesco Tapparo. 2007. Generalized Strong Preservation by Abstract Interpretation. *J. Log. Comput.* 17, 1 (2007), 157–197. https://doi.org/10.1093/logcom/exl035

[34] Francesco Ranzato and Francesco Tapparo. 2007. A New Efficient Simulation Equivalence Algorithm. In *LICS: 22nd Annual IEEE Symposium on Logic in Computer Science*. IEEE, 171–180. https://doi.org/10.1109/lics.2007.8

[35] Davide Sangiorgi. 1998. On the Bisimulation Proof Method. *Mathematical Structures in Computer Science* 8 (1998), 447–479.

[36] VERASCO. 2015. A Formally-Verified Static Analyzer for C. http://compcert.inria.fr/verasco/. (2015).

[37] M. Ward. 1942. The closure operators of a lattice. *Ann. Math.* 43, 2 (1942), 191–196.