Reachability-guided Abstraction Refinement

Pierre Ganty[1][0000−0002−3625−6003], Nicolas Manini[1,2][0000−0002−7561−3763], and Francesco Ranzato[3][0000−0003−0159−0068]

[1] IMDEA Software Institute, Spain
`{nicolas.manini,pierre.ganty}@imdea.org`
[2] Universidad Politécnica de Madrid, Spain
[3] Dipartimento di Matematica, University of Padova, Italy
`francesco.ranzato@unipd.it`

Abstract. To mitigate the state explosion problem in model checking, abstraction techniques provide sound but typically incomplete approximations of a system's behaviour. While complete abstractions eliminate false alarms, they are often impractical—or even uncomputable—due to their high computational cost. We introduce semi-completeness, a relaxed notion of completeness that retains sufficient precision to capture a system's behaviour over relevant regions of the domain. Building on this, we develop abstraction refinement algorithms that compute semi-complete abstractions without incurring the cost of full completeness. Furthermore, we present an algorithm that interleaves abstraction refinement with fixed-point computations—specifically reachability analysis. This achieves semi-completeness on-the-fly, without requiring prior knowledge of the region of interest, such as the reachable states. We demonstrate the effectiveness of our approach on fragments of the μ-calculus, showing that our abstractions preserve the validity of formulae over all reachable states.

Keywords: Abstract Model Checking · Reachable States · Abstraction Refinement · Complete Abstraction · Semi-Complete Abstraction.

1 Introduction

Model checking is a fundamental technique for system verification, yet the well-known state explosion problem hampers its practical applicability to complex hardware and software systems. Abstraction techniques effectively address this issue by reducing the state space to a tractable size. However, while abstract model checking is sound-by-construction, it is typically incomplete. Ideally, abstractions should be complete to precisely capture the concrete system's behaviour; yet, such completeness is rarely achievable with standard abstract domains. In this paper, we address this limitation by investigating how to construct abstractions that, while not fully complete, preserve sufficient precision to enable effective verification.

Contributions. Section 3 introduces the notion of *semi-completeness*, a weakening of (forward) completeness for a concrete function f over a concrete domain C. The property ρ-semi f-completeness restricts the completeness of f to a semantically relevant region of C, determined by a parameter $\rho \subseteq C$ which is a lower closure operator specifying the extent to which completeness is required. To enforce ρ-semi f-completeness, we present Algorithm 1, an abstraction refinement procedure that requires as input a full specification R of the region where completeness must hold. In addition to proving its correctness, we show that the proposed algorithm performs strictly fewer refinement steps than the standard fully complete refinement procedure.

Section 4 introduces Algorithm 2, a refinement procedure that achieves semi-completeness without requiring a full specification R of the region in advance. Instead, this algorithm computes R on-the-fly; for instance, when R represents the reachable state subspace, the algorithm derives it from the initial states and the successor transformer. Notably, it achieves semi-completeness *without* constructing the fully complete refinement or generating the entire fixed-point R.

Section 5 applies these notions to μ-calculus model checking. We show that the computed semi-complete abstractions preserve the validity of formulas in the $\{\wedge, \exists\}$-free fragment over all reachable states: a reachable state s satisfies a formula φ in this abstraction if and only if it does so in the concrete model. Finally, we demonstrate how minor extensions of this algorithm cover both the universal fragment and the full μ-calculus.

Motivating Example. Figure 1 illustrates an example. The top-right inset depicts an extended finite state machine (FSM) with configurations $\langle s, x, y\rangle \in \{s_e, s_o\} \times \mathbb{Q} \times \mathbb{Z}$. This FSM induces a transition system where transitions follow the edges of the FSM; for instance, the system may move from $\langle s_e, 0, 0\rangle$ to $\langle s_o, 0, 1\rangle$ via the $y+1$ edge, or loop in $\langle s_e, 0, 0\rangle$ via the $2x$ edge. The main plot in Figure 1 displays a fragment of this system for a fixed rational number $M \in \mathbb{Q}$. The vertical axis represents values of x and the horizontal axis represents y. System configurations where $s = s_e$ are marked as squares, while those where $s = s_o$ are circles. The initial configuration $\langle s_e, 0, 0\rangle$ is located at the origin of the axes. Reachable configurations are filled in black and lie entirely along the non-negative y-semiaxis.

We partition the configuration space into four blocks:

$$E_M^{>} \triangleq \{\langle s_e, x, y\rangle \mid x > M\}, \qquad E_M^{\leq} \triangleq \{\langle s_e, x, y\rangle \mid x \leq M\},$$
$$O_0^{\geq} \triangleq \{\langle s_o, x, y\rangle \mid x \geq 0\}, \qquad O_0^{<} \triangleq \{\langle s_o, x, y\rangle \mid x < 0\}.$$

In the figure, $E_M^{\leq}$ and $E_M^{>}$ are shown as gray blocks with continuous borders, while $O_0^{<}$ and $O_0^{\geq}$ appear as light gray blocks with dashed borders. Rounded corners indicate strict inequalities (open intervals), namely, $E_M^{>}$ and $O_0^{<}$.

Because the system has infinitely many reachable states and an infinite bisimilarity quotient, standard reduction techniques are infeasible. The key insight, however, is that combining reachability analysis with bisimilarity reveals a much

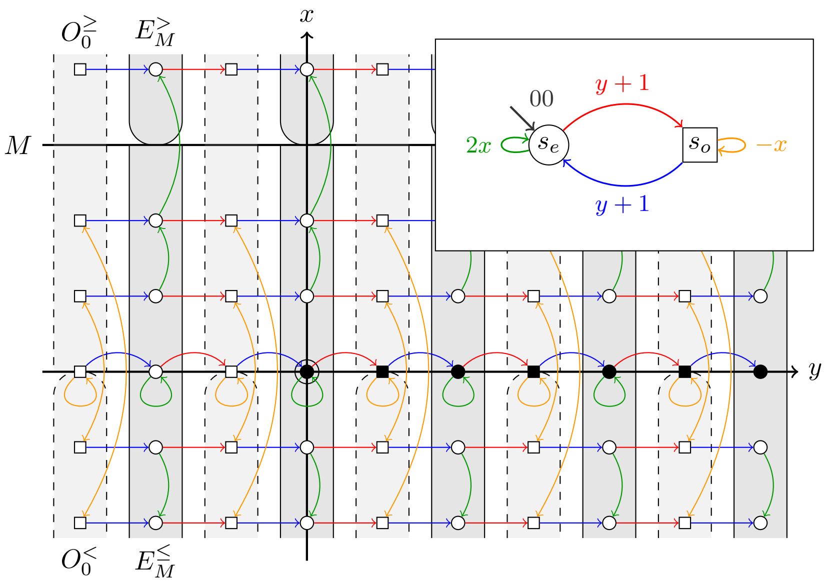

Fig. 1: The inset depicts an extended finite-state machine with variables x and y, together with its induced transition system. A partition of the state space into four blocks ($E_M^{>}$, $E_M^{\leq}$, $O_0^{\geq}$, $O_0^{<}$), parametrized by $M \in \mathbb{Q}$, is also shown: two gray blocks with solid borders and two light-gray blocks with dashed borders.

simpler structure: among the infinitely many bisimilarity blocks, only two actually contain reachable configurations. Our approach leverages this observation to compute abstractions that preserve the relevant semantics (e.g., μ-calculus properties) over reachable states, without enumerating the entire reachable set or constructing the full bisimilarity quotient.

Related Work. System abstraction is a widely adopted and effective technique for mitigating the state-explosion problem in model checking. It shifts reasoning to a more tractable domain, at the cost of some loss of precision. To be sound and complete with respect to the properties of interest, an abstraction must preserve them: a property holds on the concrete system if and only if it holds on its abstraction. Due to space constraints, we focus on the most closely related works, classifying them along two dimensions: (i) whether refinement preserves a single property or an entire specification language, and (ii) whether reachability information is taken into account.

Counterexample-Guided Abstraction Refinement (CEGAR) [4, 5] incrementally constructs abstractions that preserve a single ACTL* property, introducing just enough precision to eliminate spurious counterexamples. In contrast, our refinement is reachability-guided rather than counterexample-guided: reachability information determines where precision is required from a semantic standpoint, independently of any specific counterexample. In a similar manner, Predicate-Directed Reachability (PDR) [3, 15] takes as input a single safety property and aims to prove it; our approach, instead, is designed to model check all properties expressible in a given target logic.

Lee and Yannakakis [16] compute abstractions that preserve the full μ-calculus over reachable states, without enforcing precision on unreachable ones. Henzinger et al. [14] also study preservation results for fragments of the μ-calculus, but without exploiting reachability information.

In abstract interpretation, Cousot et al. [10] and Ranzato et al. [19] develop refinements that preserve a single safety property while accounting for reachable states. Bonchi et al. [2] address the same setting, relating abstract interpretation techniques to up-to methods. Also, Ranzato and Tapparo [20, 21] focus on preserving fragments of the specification language, but, unlike our approach, do not incorporate reachability considerations.

2 Background

Functions and Closures. We denote by id_X (or simply id when clear from the context) the identity function on a set X. Given a function $f\colon X \to X$ and $i \in \mathbb{N}$, $f^i : X \to X$ denotes the usual i-th power of f (i.e., $f^0 \triangleq id_X$ and $f^{i+1} \triangleq f f^i$). For clarity, we sometimes omit parentheses in function application, writing $f\,x$ instead of $f(x)$. Complete lattices are denoted by $\langle C, \preceq_C, \curlywedge_C, \curlyvee_C, \top_C, \bot_C\rangle$, where subscripts are omitted when clear from the context. For $S \subseteq C$, $\downharpoonleft S \triangleq \{c \in C \mid \exists s \in S.\, c \preceq s\}$ denotes the *downward closure* of S. A set S is *downward closed* iff $S = \downharpoonleft S$ holds, and we write $\downharpoonleft c$ as a shorthand for $\downharpoonleft\{c\}$. We denote monotone (i.e., order-preserving) functions as $f\colon C \xrightarrow{m} D$. A function is *additive* (resp., *continuous*) when it preserves lubs of arbitrary subsets (resp., of chains). We denote with $\mathrm{lfp}_a(f)$ (resp., $\mathrm{gfp}_a(f)$) the least (resp., greatest) fixed point of $f\colon C \to C$ that is greater (resp., smaller) than $a \in C$, whenever it exists. Moreover, $\mathrm{lfp}(f) \triangleq \mathrm{lfp}_\bot(f)$ and $\mathrm{gfp}(f) \triangleq \mathrm{gfp}_\top(f)$. Defining $x_0 \triangleq a$ and $x_{n+1} \triangleq f(x_n)$ yields the *Kleene iterates* of f from a. For complete lattices and continuous functions, if $f(a) \preceq a$ then $\mathrm{lfp}_a(f) = \curlyvee\{x_n \mid n \in \mathbb{N}\}$.

We denote by $\mathrm{uco}(C)$ the set of *upper closure operators* (ucos) over a complete lattice C, that is, functions $\mu\colon C \to C$ that are monotone, idempotent, and extensive ($x \preceq_C \mu(x)$ for all x). Dually, $\mathrm{lco}(C)$ denotes the set of *lower closure operators* (lcos) $\rho\colon C \to C$, which are monotone, idempotent, and reductive ($\rho(x) \preceq_C x$). Both ucos and lcos are determined by their sets of fixed points, which coincides with their images $\mu(C)$ and $\rho(C)$, respectively. For $\mu \in \mathrm{uco}(C)$, this means that $\mu(C) = \{c \in C \mid \mu(c) = c\}$ and that, for all $c \in C$, $\mu(c) = \curlywedge_C\{x \in \mu(C) \mid c \preceq_C x\}$. Henceforth, we shorten the notation $\mu(C)$ for the set of fixed points of a closure operator by omitting the argument and writing simply μ. Viewing closure operators either as functions or as sets, depending on the context, provides a convenient and flexible notation. For instance, given $c \in C$, writing $c \in \mu$ means $c \in \mu(C)$, which is equivalent to $\mu(c) = c$. Similarly,

$\mu \subseteq \nu$ stands for $\mu(C) \subseteq \nu(C)$, which in turn is equivalent to $\mu(c) = c$ implying $\nu(c) = c$ for all $c \in C$. We sometimes mix the two views, as in $\mu(\rho) \subseteq \rho$, which states that the fixed points of ρ, namely $\rho(C)$, include the images under μ of the fixed points of ρ. Upper and lower closures are compared using the pointwise order $\sqsubseteq$: If $\mu, \nu \in \mathrm{uco}(C)$ (or $\mu, \nu \in \mathrm{lco}(C)$), then $\mu \sqsubseteq \nu$ holds when for all $x \in C$, $\mu(x) \preceq_C \nu(x)$. For upper closures, $\mu \sqsubseteq \nu$ holds iff $\nu \subseteq \mu$, whereas for lower closures, $\mu \sqsubseteq \nu$ iff $\mu \subseteq \nu$. Accordingly, the intuition for upper closures is that if $\mu \sqsubseteq \nu$ then μ is a *refinement* of ν (or, equivalently, ν is an *abstraction* of μ), since the image of μ includes that of ν. Dually, for lower closures.

Complete Abstractions. Abstract domains are classically introduced either as Galois connections or as upper closure operators on a complete lattice C [8,9]; it is well known that the two formalisms are interchangeable [7, Chapter 11]. In this work we adopt the closure operator view, which in our setting provides significant advantages in terms of presentation. Thus, $\mu \in \mathrm{uco}(C)$ is regarded as an abstraction of C, so that $\mu(c)$ is the best abstract approximation of $c \in C$ in the abstract domain $\mu(C)$.

Given a function $f \colon C \to C$, an abstraction μ is *forward f-complete* when $f\mu = \mu f \mu$ holds; we write $\mathbb{C}^f(\mu)$ to denote this property. Intuitively, $\mathbb{C}^f(\mu)$ holds when the behaviour of f is fully preserved by the abstraction μ, since applying the concrete function f to abstract values in μ yields a result that is already an abstract value in μ. Forward complete abstractions are therefore ideal for model checking, as they introduce the least possible loss of precision. The *f-complete shell* (or simply *complete shell*) of μ [12]—the least refinement of μ that is forward f-complete, i.e., $f\text{-shell}(\mu) = \bigcap\{\nu \in \mathrm{uco}(C) \mid \mu \subseteq \nu\,, \mathbb{C}^f(\nu)\}$—plays a key role in what follows. In this sense, forward completeness characterizes maximal precision, while complete shells are the coarsest abstractions (i.e., those that abstract as much as possible) achieving such precision.

Transition Systems. Let $G = (\Sigma_G, I_G, \to_G)$ be a transition system (TS), where Σ_G is a (possibly infinite) set of states, $I_G \subseteq \Sigma_G$ is the set of initial states, and $\to_G \subseteq \Sigma_G \times \Sigma_G$ is the transition relation. We write $x \to_G y$ to denote $(x, y) \in \to_G$. The successor transformer $\mathsf{post}_G \colon \wp(\Sigma_G) \to \wp(\Sigma_G)$ is defined as usual: $\mathsf{post}_G(X) \triangleq \{y \in \Sigma_G \mid \exists x \in X.\, x \to_G y\}$. Dually, the predecessor transformer $\mathsf{pre}_G \colon \wp(\Sigma_G) \to \wp(\Sigma_G)$ is defined by $\mathsf{pre}_G(Y) \triangleq \{x \in \Sigma_G \mid \exists y \in Y.\, x \to_G y\}$. The set of *reachable states* is $\mathsf{post}^*_G(I_G) \triangleq \mathrm{lfp}(\lambda X.\, I_G \cup \mathsf{post}_G(X))$. Subscripts are omitted when no ambiguity arises.

3 Semi-Completeness

In this section we introduce *semi-completeness*, a weakening of forward completeness, together with its "operational" counterpart, *semi-stability*[4]. Recall that f-completeness requires an abstraction $\mu \in \mathrm{uco}(C)$ to capture the behaviour of

[4] The prefix "semi-" is inspired by Lee and Yannakakis [16, Definition 3.2].

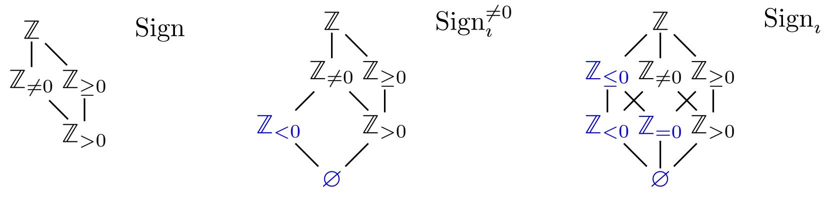
$\mathbb{Z}$
Sign
$\mathbb{Z}_{\neq 0}$ $\mathbb{Z}_{\geq 0}$
$\mathbb{Z}_{>0}$
$\mathbb{Z}$
$\text{Sign}_\iota^{\neq 0}$
$\mathbb{Z}_{\neq 0}$ $\mathbb{Z}_{\geq 0}$
$\mathbb{Z}_{<0}$ $\mathbb{Z}_{>0}$
$\varnothing$
$\mathbb{Z}$
Sign_ι
$\mathbb{Z}_{\leq 0}$ $\mathbb{Z}_{\neq 0}$ $\mathbb{Z}_{\geq 0}$
$\mathbb{Z}_{<0}$ $\mathbb{Z}_{=0}$ $\mathbb{Z}_{>0}$
$\varnothing$

Fig. 2: Three Sign abstract domains. Elements represent subsets of $\mathbb{Z}$: $\mathbb{Z}_{\geq 0}$ denotes the set of non-negative integers, $\mathbb{Z}_{\neq 0}$ the set of non-zero integers, and so on.

$f\colon C \to C$ on *all elements* of C, thereby enforcing precision across the entire domain. Semi-completeness relaxes this requirement. The key idea is that many applications demand precision only in certain subspaces in which f is applied, rather than on the whole domain C. To express this, semi-completeness is parameterised by an additional lower closure $\rho \in \text{lco}(C)$, which specifies "to what extent" forward completeness should hold. This parameter function ρ should be seen as a "filter" that removes irrelevant information. Choosing ρ to be a lower closure operator is natural, since $\rho(c)$ represents the "region of c" relevant for achieving completeness, and therefore $\rho(c) \preceq c$ is an expected requirement. Moreover, monotonicity ensures that elements containing less information cannot retain more irrelevant information than elements containing more information, while idempotency guarantees that ρ removes all irrelevant information in a single step.

We introduce the *non-spilling* property to characterize how the "filter" ρ is expected to interact with the abstraction μ.

Definition 3.1 (Non-Spilling lco). Given an abstraction $\mu \in \text{uco}(C)$, we define $\text{NS}(\mu) \triangleq \{\rho \in \text{lco}(C) \mid \mu(\rho) \subseteq \rho\}$ as the set of lcos that are *non-spilling* for μ. Equivalently, $\text{NS}(\mu) = \{\rho \in \text{lco}(C) \mid \rho\mu\rho = \mu\rho\}$.

A non-spilling lower closure ρ ensures that ρ is "compatible" with μ: applying μ to a filtered element $\rho(x)$ does not reintroduce information that ρ removes, that is, formally, $\rho(\mu(\rho(x))) = \mu(\rho(x))$ holds.

Example 3.2 ((Non-)Spilling lco). Let $\mu \in \text{uco}(\wp(\mathbb{Z}))$ be the uco induced by $\text{Sign}_\iota^{\neq 0}$ in Fig. 2 (i.e., the fixed points of μ are the subsets of $\mathbb{Z}$ included in $\text{Sign}_\iota^{\neq 0}$). Consider the lco ρ' defined by $\rho' \triangleq \lambda X.\, X \cap \{-1, 1\}$. This lco is spilling for μ: for instance, $\{1\} \in \rho'$ (i.e., $\rho'(\{1\}) = \{1\}$), while $\mu(\{1\}) = \mathbb{Z}_{>0} \notin \rho'$, since $\rho'(\mathbb{Z}_{>0}) = \{1\} \neq \mathbb{Z}_{>0}$. On the other hand, the lco ρ defined by $\rho \triangleq \lambda X.\, X \smallsetminus \{0\}$ is non-spilling for μ. In fact, for any $X \in \rho$ (i.e., any X such that $0 \notin X$), we have $X \subseteq \mathbb{Z}_{\neq 0}$, and therefore $\mu(X) \in \{\mathbb{Z}_{\neq 0}, \mathbb{Z}_{<0}, \mathbb{Z}_{>0}, \varnothing\} \subseteq \rho$. ◊

Definition 3.3 (Forward Semi-Completeness). Given a concrete function $f\colon C \to C$, $\mu \in \text{uco}(C)$, and $\rho \in \text{lco}(C)$, μ is *forward ρ-semi f-complete* (or simply *ρ-semi f-complete*), denoted $\mathbf{fsc}_f^\rho(\mu)$, when $\rho\,\mu\,\rho\,f\,\mu = \rho\,f\,\mu$ holds[5].

[5] Note that $\mathbf{fsc}_f^\rho(\mu)$ holds when for all $x \in \mu$, $\rho f(x)$ is a fixed point of $\rho\mu$.

Sub/superscripts in $\mathbf{fsc}^{\rho}_{f}(\mu)$ may be omitted when they are clear from the context. Observe that semi-completeness $\mathbf{fsc}^{\rho}_{f}(\mu)$ is implied by $\mathbb{C}^{\rho f}(\mu)$, i.e. by forward ρf-completeness $\mu\rho f\mu = \rho f\mu$. In contrast, the semi-completeness equation $\rho\mu\rho f\mu = \rho f\mu$ requires this equality only up to the information selected by the filter lco ρ. Hence, $\mathbf{fsc}^{\rho}_{f}(\mu)$ can be viewed as a weakening of $\mathbb{C}^{\rho f}(\mu)$, enforcing completeness only on the portion of the domain retained by ρ. In particular, observe that when ρ is non-spilling, $\mathbf{fsc}^{\rho}_{f}(\mu)$ and $\mathbb{C}^{\rho f}(\mu)$ coincide:

$$\forall\mu \in \mathrm{uco}(C),\ \rho \in \mathrm{NS}(\mu).\ \mathbf{fsc}^{\rho}_{f}(\mu) \Leftrightarrow \mathbb{C}^{\rho f}(\mu)\ . \tag{1}$$

Moreover, even when ρ is spilling, decoupling ρ and f is advantageous. It allows one to reason independently about f, which determines the concrete behaviour for which completeness is sought, and ρ, which specifies where this completeness must hold. This separation also supports the refinement strategies developed in Sections 3.2 and 4, where ρ is computed on-the-fly rather than assumed to be known in advance.

Example 3.4 (Semi-Completeness). Consider the sign abstractions in Fig. 2, the lcos ρ' and ρ from Example 3.2, and the inverse function $\imath : \wp(\mathbb{Z}) \to \wp(\mathbb{Z})$, i.e., $\imath(X) \triangleq \{-x \mid x \in X\}$. Let μ' be the uco induced by Sign. We first observe that the abstraction μ' is not ρ'-semi $\imath$-complete, since $\rho'\imath(\mathbb{Z}_{\geq 0}) = \{-1\}$ while $\rho'\mu'\rho'\imath(\mathbb{Z}_{\geq 0}) = \rho'\mu'(\{-1\}) = \rho'(\mathbb{Z}_{\neq 0}) = \{-1, 1\}$. The same example shows that μ' is not ρ-semi $\imath$-complete either. Let now μ be the uco induced by $\mathrm{Sign}^{\neq 0}_{\imath}$. We observe that μ is ρ'-semi $\imath$-complete. Indeed, for $X \in \{\mathbb{Z}, \mathbb{Z}_{\neq 0}\}$, we have $\rho'\imath(X) = \{-1, 1\}$ and $\rho'\mu(\{-1, 1\}) = \rho'(\mathbb{Z}_{\neq 0}) = \{-1, 1\}$. For $X \in \{\mathbb{Z}_{\geq 0}, \mathbb{Z}_{>0}\}$, we obtain $\rho'\imath(X) = \{-1\}$ with $\rho'\mu(\{-1\}) = \rho'(\mathbb{Z}_{<0}) = \{-1\}$. Finally, we have $\rho'\imath(\mathbb{Z}_{<0}) = \{1\} = \rho'\mu(\{1\})$, and $\rho'\imath(\varnothing) = \varnothing = \rho'\mu(\varnothing)$.

Observe that ρ' is a spilling lco for μ (as shown in Example 3.2), and indeed $\mathbb{C}^{\rho'\imath}(\mu)$ does not hold. For instance, $\rho'\imath(\mathbb{Z}_{<0}) = \rho'(\mathbb{Z}_{>0}) = \{1\} \notin \mu$. Consider now the non-spilling lco ρ. In this case, μ is ρ-semi $\imath$-complete. Indeed, $\rho\imath(\mathbb{Z}) = \rho\imath(\mathbb{Z}_{\neq 0}) = \mathbb{Z}_{\neq 0} = \rho\mu(\mathbb{Z}_{\neq 0})$ holds, along with $\rho\imath(\mathbb{Z}_{\geq 0}) = \rho\imath(\mathbb{Z}_{>0}) = \mathbb{Z}_{<0} = \rho\mu(\mathbb{Z}_{<0})$ and $\rho\imath(\mathbb{Z}_{<0}) = \mathbb{Z}_{>0} = \rho\mu(\mathbb{Z}_{>0})$. Finally, $\rho\imath(\varnothing) = \varnothing = \rho\mu(\varnothing)$ proves that ρ-semi $\imath$-completeness holds. By (1), this implies that $\mathbb{C}^{\rho\imath}(\mu)$ holds. ◊

As anticipated, semi-completeness is strictly weaker than forward completeness, meaning that: `(I)` $\mathbb{C}^{f}(\mu)$ is recovered as the special case of id-semi f-completeness, and `(II)` forward completeness strictly implies semi-completeness. In particular, given μ and f, the set $\{\rho \in \mathrm{lco}(C) \mid \mathbf{fsc}^{\rho}_{f}(\mu)\}$ is downward closed:

Lemma 3.5. *Let $f : C \to C$, $\mu \in \mathrm{uco}(C)$, and $\rho \in \mathrm{lco}(C)$. Then $\mathbf{fsc}^{\rho}_{f}(\mu)$ holds iff $\forall\rho' \in \mathrm{lco}(C).\, \rho' \sqsubseteq \rho \Rightarrow \mathbf{fsc}^{\rho'}_{f}(\mu)$.*

Lemma 3.5 immediately yields point `(II)`. Since $\rho \sqsubseteq id$ holds for every $\rho \in \mathrm{lco}(C)$, downward closure implies that $\mathbf{fsc}^{id}_{f}(\mu)$ (i.e., forward f-completeness) holds if and only if ρ-semi f-completeness holds for every $\rho \in \mathrm{lco}(C)$. The following example shows that semi-completeness is strictly weaker than forward completeness.

Example 3.6. Consider the domains in Fig. 2 and the lcos ρ' and ρ from Example 3.4, where we showed that $\mathrm{Sign}_\imath^{\neq 0}$ is both ρ-semi $\imath$-complete and ρ'-semi $\imath$-complete. Note that $\mathrm{Sign}_\imath$ is the $\imath$-complete shell of Sign. Moreover, $\mathrm{Sign}_\imath^{\neq 0}$ is a strict abstraction of $\mathrm{Sign}_\imath$. Therefore, forward semi-completeness may hold even when forward completeness does not. $\Diamond$

3.1 Semi-Stability

We now introduce *semi-stability*, an equivalent notion to semi-completeness that admits an "operational" formulation and underlies abstraction refinement algorithms aimed at achieving semi-completeness, such as Algorithms 1 and 2.

Definition 3.7 (Semi-Stability). Let $f\colon C \to C$, $\mu \in \mathrm{uco}(C)$, and $\rho \in \mathrm{NS}(\mu)$. Given a concrete value $x \in C$, μ is *ρ-semi f-stable on x* if

$$\forall y \in C.\, x \preceq \rho f y \Rightarrow \rho\mu x \preceq f\mu y \ .$$

A pair $\langle x, y\rangle \in C^2$ satisfying $x \preceq \rho f y$ and $\rho\mu x \not\preceq f\mu y$ is called *unstable*. Moreover, μ is ρ-semi f-stable if it is ρ-semi f-stable on every $x \in C$, denoted by $\mathbf{ssa}_f^\rho(\mu)$.

Sub/superscripts in $\mathbf{ssa}_f^\rho(\mu)$ may be omitted when clear from context. Semi-stability and semi-completeness are equivalent for monotone functions:

Lemma 3.8. *If $f\colon C \overset{m}{\to} C$, $\mu \in \mathrm{uco}(C)$, and $\rho \in \mathrm{lco}(C)$, then $\mathbf{ssa}_f^\rho(\mu) \Leftrightarrow \mathbf{fsc}_f^\rho(\mu)$.*

In particular, an unstable pair $\langle x, y\rangle$ witnesses non-semi-completeness: $\rho f\mu y \neq \rho\mu\rho f\mu y$, since $x \preceq \rho f y \Rightarrow \rho\mu x \preceq \rho\mu\rho f y \preceq \rho\mu\rho f\mu y$, and $\rho\mu\rho f\mu y = \rho f\mu y$ would imply $\rho\mu x \preceq f\mu y$, contradicting instability.

Example 3.9 (Semi-Stability). Consider the abstractions in Fig. 2, the lco ρ, and the ucos μ, μ' from Example 3.4, where μ' was shown to be not ρ-semi $\imath$-complete. By Lemma 3.8, μ' is therefore not ρ-semi $\imath$-stable: for example, the pair $\langle\{-1\}, \{0, 1\}\rangle$ is unstable, since $\{-1\} \subseteq \{-1\} = \rho\imath(\{0, 1\})$ and $\rho\mu'(\{-1\}) = \rho(\mathbb{Z}_{\neq 0}) = \mathbb{Z}_{\neq 0} \not\subseteq \mathbb{Z}_{\leq 0} = \imath\mu'(\{0, 1\})$. On the other hand, μ is ρ-semi $\imath$-stable, since $\rho\mu(\{-1\}) = \rho(\mathbb{Z}_{<0})$, so the pair $\langle\{-1\}, \{0, 1\}\rangle$ is not unstable in $\mathrm{Sign}_\imath^{\neq 0}$. $\Diamond$

3.2 Ideal Semi-Completion

We now exploit semi-stability (in particular its "operational" formulation) to design an algorithm for computing semi-stable abstractions, which, via Lemma 3.8, ensures semi-completeness. To this end, we introduce a class of lower closure operators, called *ideal closures*. Given a concrete value $c \in C$, define the function $\lfloor c \rfloor\colon C \to C$ as $\lfloor c \rfloor(x) \triangleq x \curlywedge c$. These functions are lower closures whose set of fixed points is the principal ideal $\downarrow c$ (see [9, § 6.4]). Intuitively, $\lfloor c \rfloor(x)$ returns the "portion" of x below c. In the context of $\lfloor c \rfloor$-semi f-completeness, the value c specifies the region of C on which we want f-completeness to hold, and

$\lfloor c \rfloor(x)$ represents the portion of x where completeness is enforced. Observe that $c \preceq d \;\Rightarrow\; \downarrow c \subseteq \downarrow d \;\Rightarrow\; \lfloor c \rfloor \sqsubseteq \lfloor d \rfloor$.

We now define a procedure that, given a monotone function $f : C \overset{m}{\to} C$, an input abstraction $\mu_i \in \mathrm{uco}(C)$, and a value $c \in C$ specifying the region of interest, computes a refinement μ of μ_i that is $\lfloor \mu c \rfloor$-semi f-complete[6]. Moreover, the output μ is always coarser than the complete shell $\mu^* \triangleq f\text{-shell}(\mu_i)$ and is $\lfloor \mu^* c \rfloor$-semi f-complete. This procedure, called *ideal semi-completion*, achieves these guarantees without necessarily computing the full complete shell μ^*.

Algorithm 1: Ideal Semi-Completion

```
  // Input: $f : C \overset{m}{\to} C$, $\mu_i \in \mathrm{uco}(C)$, $c \in C$.
1 $\mathrm{uco}(C) \ni \mu \coloneqq \mu_i$;
2 while $\mathrm{UP} \coloneqq \{\langle x, y\rangle \in C^2 \mid x \preceq \mu c,\ x \preceq f y,\ \mu x \not\preceq f \mu y\} \neq \varnothing$ do
      // Inv 1: $\mu \in \mathrm{uco}(C)$.
      // Inv 2: $\mu^* \sqsubseteq \mu$.
3     choose $\langle x, y\rangle \in \mathrm{UP}$;
4     $\mu \coloneqq \mu \cup \{\mu z \curlywedge f \mu y \mid z \in C,\ \mu z \preceq \mu x\}$;
5 end while
6 return $\mu$;
```

Algorithm 1 follows a “stabilization” approach: at each iteration, line 4 enforces $\lfloor \mu c \rfloor$-stability for a selected pair $\langle x, y\rangle$. The set UP of Unstable Pairs (line 2) is empty if and only if μ is $\lfloor \mu c \rfloor$-semi f-stable. Indeed, $x \preceq \mu c$ and $x \preceq f y$ hold if and only if $x \preceq \lfloor \mu c \rfloor f y$, and moreover, $x \preceq \mu c$ entails $\lfloor \mu c \rfloor \mu x = \mu x$, so that $\mu x \preceq f \mu y$ holds if and only if $\lfloor \mu c \rfloor \mu x \preceq f \mu y$. Upon termination, $\lfloor \mu c \rfloor$-semi f-stability, along with $\lfloor \mu c \rfloor \in \mathrm{NS}(\mu)$, and (1) guarantee that μ is $\lfloor \mu c \rfloor$-semi f-complete. A key aspect of the algorithm is that, while c is constant, the operator $\lfloor \mu c \rfloor$ is not: it depends on μ, which is updated at line 4. Thus, at each iteration, stabilization occurs with respect to a potentially different function $\lfloor \mu c \rfloor f$. In summary, Algorithm 1 computes $\lfloor \mu c \rfloor$-semi f-complete abstraction refinements while avoiding the full complete shell computation. This is particularly important because the resulting abstractions are precise in the domain region *below* $\mu(c)$, and the value of c can be chosen to encode the limit cases that the analysis should cover.

Theorem 3.10 (Correctness of Algorithm 1). *Upon termination, Algorithm 1 returns a refinement μ of μ_i such that both $\mathbf{fsc}_f^{\lfloor \mu^* c \rfloor}(\mu)$ and $\mathbf{fsc}_f^{\lfloor \mu c \rfloor}(\mu)$ hold, where $\mu^* \triangleq f\text{-shell}(\mu_i)$. Moreover, μ is refined at every iteration and remains coarser than μ^*.*

Notice that by instantiating Algorithm 1 with $c = \top$, we recover the standard stabilization approach, which computes the full f-complete shell.

Example 3.11 (Execution of Algorithm 1). Consider the motivating example from Section 1. Note that, for any configuration s, $s = s_e$ holds iff y is even, so we

[6] Considering $\lfloor \mu c \rfloor$ instead of $\lfloor c \rfloor$ yields abstractions coarser than f-shell(μ_i).

represent configurations as pairs $\langle x, y\rangle$. Let $\mathbb{E}$ and $\mathbb{O}$ denote the set of even and odd integers, respectively. Given a set $S \subseteq \mathbb{Z}$, define $(a, b)_S \triangleq \{\langle x, y\rangle \in \mathbb{Q} \times \mathbb{Z} \mid a < x < b,\ y \in S\}$, the set of pairs $\langle x, y\rangle$ with x ranging in the open interval (a, b), and $y \in S$. This notation extends naturally to closed or mixed intervals. Let μ_i be the abstraction induced by the initial partition $\{[0, +\infty)_{\mathbb{O}}, (-\infty, 0)_{\mathbb{O}}, (M, +\infty)_{\mathbb{E}}, (-\infty, M]_{\mathbb{E}}\}$. That is, $\mu_i(X)$ is the union of all blocks intersecting X[7]. Moreover, let $c = [0, 0]_{\mathbb{Z}}$ (i.e., c is the horizontal axis). We execute Algorithm 1 on input $\langle \mathsf{pre}, \mu_i, c\rangle$.

At the *first iteration*, we have $\mu(c) = (-\infty, M]_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}}$ and the pair $\langle c, [0, +\infty)_{\mathbb{O}}\rangle$ is unstable (i.e., it is in UP), since: $c \subseteq \mu(c)$ holds, along with $c \subseteq [0, +\infty)_{\mathbb{E}} \cup (-\infty, 0]_{\mathbb{O}} = \mathsf{pre}([0, +\infty)_{\mathbb{O}})$, and $\mu(c) \nsubseteq \mathsf{pre}\,\mu([0, +\infty)_{\mathbb{O}})$ hold (notice that the right hand side coincides with $\mathsf{pre}([0, +\infty)_{\mathbb{O}})$). Therefore, the algorithm refines μ by iterating over the abstract points $z \subseteq \mu(c)$ and adding the following new values to μ: for $z = \mu(c)$ we add $\mu(c) \cap \mathsf{pre}([0, +\infty)_{\mathbb{O}}) = [0, M]_{\mathbb{E}} \cup [0, 0]_{\mathbb{O}}$; for $z = (-\infty, M]_{\mathbb{E}}$ we add $(-\infty, M]_{\mathbb{E}} \cap \mathsf{pre}([0, +\infty)_{\mathbb{O}}) = [0, M]_{\mathbb{E}}$; for $z = [0, +\infty)_{\mathbb{O}}$ we add $[0, +\infty)_{\mathbb{O}} \cap \mathsf{pre}([0, +\infty)_{\mathbb{O}}) = [0, 0]_{\mathbb{O}}$; and for $z = \varnothing$ nothing is added to μ.

At the *second iteration*, we have $\mu(c) = [0, M]_{\mathbb{E}} \cup [0, 0]_{\mathbb{O}}$ and the pair $\langle c, [0, 0]_{\mathbb{O}}\rangle$ is unstable since: $c \subseteq \mu(c)$ holds, along with $c \subseteq c = \mathsf{pre}([0, 0]_{\mathbb{O}})$, and $\mu(c) \nsubseteq \mathsf{pre}\,\mu([0, 0]_{\mathbb{O}})$ (note that the right hand side coincides with $\mathsf{pre}([0, 0]_{\mathbb{O}})$). Therefore, the algorithm refines μ by iterating over the abstract points $z \subseteq \mu(c)$ and by adding the following new values to μ: for $z = \mu(c)$ we add $\mu(c) \cap \mathsf{pre}([0, 0]_{\mathbb{O}}) = \mu(c) \cap c = c$; for $z = [0, M]_{\mathbb{E}}$ we add $[0, M]_{\mathbb{E}} \cap c = [0, 0]_{\mathbb{E}}$; for $z = [0, 0]_{\mathbb{O}}$ we have that $[0, 0]_{\mathbb{O}} \cap c = [0, 0]_{\mathbb{O}}$ and no new element is added (same for $z = \varnothing$).

At the *third iteration*, $\mu(c) = c$ and UP $= \varnothing$ hold, since the abstraction is $\lfloor \mu c \rfloor$-semi pre-complete. In fact, $\mu = \mu_i \cup \{[0, 0]_{\mathbb{E}}, [0, 0]_{\mathbb{O}}, [0, 0]_{\mathbb{Z}}, [0, M]_{\mathbb{E}}\}$ holds. We now check that for all $x \in \mu$, $\lfloor \mu c \rfloor\, \mathsf{pre}(x) \in \mu$ holds. We first check the elements in μ_i: we have that $\lfloor \mu c \rfloor\, \mathsf{pre}((-\infty, M]_{\mathbb{E}}) = \lfloor \mu c \rfloor\, \mathsf{pre}([0, +\infty)_{\mathbb{O}}) = [0, 0]_{\mathbb{Z}} \in \mu$, and $\lfloor \mu c \rfloor\, \mathsf{pre}((M, +\infty)_{\mathbb{E}}) = \lfloor \mu c \rfloor\, \mathsf{pre}((-\infty, 0)_{\mathbb{O}}) = \varnothing \in \mu$. Moreover, for the remaining elements $x \in \mu_i$ (i.e., x is a union of blocks), we have that $\lfloor \mu c \rfloor\, \mathsf{pre}(x)$ coincides with $[0, 0]_{\mathbb{Z}}$ or $\varnothing$, depending on whether x intersects c or not. For the remaining points in μ, we have that $\lfloor \mu c \rfloor\, \mathsf{pre}([0, 0]_{\mathbb{E}}) = \lfloor \mu c \rfloor\, \mathsf{pre}([0, 0]_{\mathbb{O}}) = \lfloor \mu c \rfloor\, \mathsf{pre}([0, 0]_{\mathbb{Z}}) = \lfloor \mu c \rfloor\, \mathsf{pre}([0, M]_{\mathbb{E}}) = [0, 0]_{\mathbb{Z}} \in \mu$. We have thus shown that μ is $\lfloor \mu c \rfloor$-semi pre-complete, so that UP $= \varnothing$, i.e., Algorithm 1 terminates. $\Diamond$

To illustrate a practical application of Algorithm 1, consider a system modeling a machine whose behavior depends on temperature. Let t be a real-valued variable representing temperature in Celsius degrees. Since temperatures below absolute zero are physically impossible, we can exclude states with $t \leq 0\,\mathrm{K}$ by defining $c = \{s \in \mathit{States} \mid t(s) > 0\,\mathrm{K}\}$, where *States* denotes the state space. This idea generalizes to constraints arising from design specifications rather than physical limitations. For instance, for machines intended to operate within a bounded temperature range, we can encode $c = \{s \in \mathit{States} \mid t(s) \in [t_{\min}, t_{\max}]\}$. Similarly, for systems with temporal constraints—such as airplane components operating for a limited time during flight—one can replace temperature with time and encode boundaries analogously.

[7] See Section 5 for a formal presentation of ucos induced by partitions.

4 Interleaving Fixed Point and Semi-Completion Computations

Algorithm 1 computes semi-complete abstraction refinements μ with respect to the ideal lower closure $\lfloor \mu c \rfloor$, assuming that the concrete value c is given. We now consider a more general setting in which c is not known in advance and must be computed on-the-fly. Formally, given an initial abstraction $\mu_i \in \text{uco}(C)$, and a pair of left- and right-adjoint functions $l, r\colon C \overset{m}{\to} C$, we aim to compute a refinement of μ_i that is $\lfloor \mu^*(c^*) \rfloor$-semi r-complete, where μ^* is the forward r-complete shell of μ_i and c^* is the least fixed point of l above some $a \in C$. Crucially, we show how to solve this problem without computing either r-shell(μ_i) or $c^* = \text{lfp}_a(l)$. To motivate this general setting, consider the adjoint pair post and $\widetilde{\mathsf{pre}}$ of a transition system, where $\widetilde{\mathsf{pre}}(Y) \triangleq \{x \mid \forall y.\, x \to y \Rightarrow y \in Y\}$ is the weakest liberal precondition. Our approach then yields an abstraction μ that is $\lfloor \mu^*(\mathsf{post}^*(I)) \rfloor$-semi $\widetilde{\mathsf{pre}}$-complete. Intuitively, the lower closure $\lfloor \mu^*(\mathsf{post}^*(I)) \rfloor$ filters out unreachable states—which are irrelevant for model checking—thus allowing the abstraction μ to be effectively used in verification while remaining coarser than the entire complete shell μ^*. This application is developed in Section 5.

4.1 Interleaving Algorithm

We first recall that, given two complete lattices A and B, two functions $l\colon A \to B$ and $r\colon B \to A$ are respectively called a *left* and a *right adjoint* (or, together, an adjoint pair), if for all $a \in A$ and $b \in B$, $l(a) \preceq_B b \Leftrightarrow a \preceq_A r(b)$. We denote by $\text{Adj}(C) \triangleq \{\langle l, r \rangle \mid l, r\colon C \to C \text{ form an adjoint pair}\}$ the set of adjoint pairs over the complete lattice C.

Given $\langle l, r \rangle \in \text{Adj}(C)$, Algorithm 2 interleaves the computation of the fixed point $\text{lfp}_a(l)$ with refinements of the abstraction μ in order to enforce $\lfloor \mu(c) \rfloor$-semi r-stability.

Algorithm 2: Fixed Point-based Ideal Semi-Completion

```
   // Input: $\mu_i \in \text{uco}(C)$, $\langle l, r \rangle \in \text{Adj}(C)$, and $a \in C$.
1  $C \ni c \coloneqq \bot$;
2  $\text{uco}(C) \ni \mu \coloneqq \mu_i$;
3  repeat
       // Inv 1: $c \preceq \text{lfp}_a(l)$.
4      while $\mu(c) \neq \mu(a \curlyvee l(c))$ do  $c \coloneqq a \curlyvee l(c)$ ;
5      if $\text{UP} \coloneqq \{\langle x, y \rangle \in C^2 \mid x \preceq \mu c,\ x \preceq ry,\ \mu x \not\preceq r\mu y\} \neq \varnothing$ then
           // Inv 2: $\mu \in \text{uco}(C)$.
           // Inv 3: $\mu^* \sqsubseteq \mu$.
6          choose $\langle x, y \rangle \in \text{UP}$;
7          $\mu \coloneqq \mu \cup \{\mu z \curlywedge r\mu y \mid \mu z \preceq \mu x\}$;
8      end if
9  until $\mu(c) = \mu(a \curlyvee l(c)) \wedge \text{UP} = \varnothing$;
10 return $\langle \mu, c \rangle$;
```

In Algorithm 2 the concrete value c corresponds to the current Kleene iterate of $\lambda x.\, a \curlyvee l(x)$, while lines 5-8 follow the same stabilization strategy as Algorithm 1. In particular, the abstraction μ is refined at line 7 so as to stabilize a pair $\langle x, y\rangle$). Note that the value μc—and therefore the lco $\lfloor \mu c \rfloor$—is not constant during the repeat-until iterations of Algorithm 2: μ is refined at line 7, as in Algorithm 1, and additionally c is updated at line 4. In particular, the *concrete* fixed point iteration (line 4) updates the value of c to the next Kleene iterate of $\lambda x.\, a \curlyvee l(x)$ only when this results in a change of $\mu(c)$. Upon termination $\mathrm{UP} = \varnothing$, and therefore the abstraction μ is $\lfloor \mu c \rfloor$-semi r-complete. This ensures that backward l-completeness holds on all the previously computed values of c. Consequently, $\mu(c)$ coincides with the abstract fixed point $\mu(\mathrm{lfp}_a(l))$, and the resulting abstraction is $\lfloor \mu(\mathrm{lfp}_a(l)) \rfloor$-semi r-complete.

Theorem 4.1 (Correctness of Algorithm 2). *Upon termination, Algorithm 2 returns a pair $\langle \mu, c\rangle$ such that r-shell$(\mu_i) = \mu^* \sqsubseteq \mu \sqsubseteq \mu_i$ and $\mu(c) = \mu(\mathrm{lfp}_a(l))$. Moreover, both $\mathbf{fsc}_r^{\lfloor \mu(c) \rfloor}(\mu)$ and $\mathbf{fsc}_r^{\lfloor \mu^*\, \mathrm{lfp}_a(l) \rfloor}(\mu)$ hold, and μ is refined at every iteration.*

Example 4.2 (Execution of Algorithm 2). Consider the motivating example presented in Section 1, together with the notation introduced in Example 3.11. We execute Algorithm 2 on this TS with input $a = \{\langle 0, 0\rangle\}$, the adjoint pair $\langle \mathsf{post}, \widetilde{\mathsf{pre}}\rangle$, and the abstraction μ_i.

At the *first iteration*, the loop at line 4 first updates c to $[0,0]_{\{0\}}$, since $\mu(\varnothing) = \varnothing \neq (-\infty, M]_{\mathbb{E}} = \mu([0,0]_{\{0\}})$. It then updates c again so that $c = [0,0]_{\{0,1\}}$, because $\mu([0,0]_{\{0\}}) \neq (-\infty, M]_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}} = \mu([0,0]_{\{0,1\}})$. The loop terminates since $\mu([0,0]_{\{0,1\}}) = \mu([0,0]_{\{0,1,2\}})$, and therefore $\mu(c) = (-\infty, M]_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}}$. The pair $\langle c, \mu(c)\rangle$ is unstable (i.e., it belongs to UP), since $c \subseteq \mu(c)$ and $c \subseteq [0, \frac{M}{2}]_{\mathbb{E}} \cup (-\infty, 0]_{\mathbb{O}} = \widetilde{\mathsf{pre}}((-\infty, M]_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}}) = \widetilde{\mathsf{pre}}(\mu(c))$, while $\mu(c) \nsubseteq \widetilde{\mathsf{pre}}(\mu(c))$. The abstraction μ is therefore refined by iterating over the abstract points $z \subseteq \mu(c)$ and adding the following new values: for $z = \mu(c)$ we add $\mu(c) \cap \widetilde{\mathsf{pre}}(\mu(c)) = [0, \frac{M}{2}]_{\mathbb{E}} \cup [0,0]_{\mathbb{O}}$; for $z = (-\infty, M]_{\mathbb{E}}$ we add $(-\infty, M]_{\mathbb{E}} \cap \widetilde{\mathsf{pre}}(\mu(c)) = [0, \frac{M}{2}]_{\mathbb{E}}$; for $z = [0, +\infty)_{\mathbb{O}}$ we add $[0, +\infty)_{\mathbb{O}} \cap \widetilde{\mathsf{pre}}(\mu(c)) = [0,0]_{\mathbb{O}}$; for $z = \varnothing$ nothing is added.

At the *second iteration*, line 4 does not update c, since $\mu([0,0]_{\{0,1\}}) = [0, \frac{M}{2}]_{\mathbb{E}} \cup [0,0]_{\mathbb{O}} = \mu([0,0]_{\{0,1,2\}})$. The pair $\langle c, \mu(c)\rangle$ remains unstable, since $c \subseteq \mu(c)$ and $c \subseteq [0,0]_{\mathbb{Z}} = \widetilde{\mathsf{pre}}([0, \frac{M}{2}]_{\mathbb{E}} \cup [0,0]_{\mathbb{O}}) = \widetilde{\mathsf{pre}}(\mu(c))$, while $\mu(c) \nsubseteq \widetilde{\mathsf{pre}}(\mu(c))$. The abstraction μ is refined again by considering $z \subseteq \mu(c)$ and adding to μ the following values: for $z = \mu(c)$ we add $\mu(c) \cap \widetilde{\mathsf{pre}}(\mu(c)) = [0,0]_{\mathbb{Z}}$; for $z = [0, \frac{M}{2}]_{\mathbb{E}}$ we add $[0, \frac{M}{2}]_{\mathbb{E}} \cap \widetilde{\mathsf{pre}}(\mu(c)) = [0,0]_{\mathbb{E}}$; for $z = [0,0]_{\mathbb{O}}$ no new element is added since $[0,0]_{\mathbb{O}} \cap \widetilde{\mathsf{pre}}(\mu(c)) = [0,0]_{\mathbb{O}} \in \mu$; for $z = \varnothing$ nothing is added.

At the *third iteration*, c is not updated and $\mu(c) = [0,0]_{\mathbb{Z}}$. We show that μ is $\lfloor \mu(c) \rfloor$-semi $\widetilde{\mathsf{pre}}$-complete. At this stage, we have $\mu = \mu_i \cup \{[0,0]_{\mathbb{E}}, [0,0]_{\mathbb{O}}, [0,0]_{\mathbb{Z}}, [0, \frac{M}{2}]_{\mathbb{E}}, [0, \frac{M}{2}]_{\mathbb{E}} \cup [0,0]_{\mathbb{O}}\}$. We verify that for every $x \in \mu$, $\lfloor \mu c \rfloor\, \widetilde{\mathsf{pre}}(x) \in \mu$ holds. First consider $x \in \mu_i$: for all $x \in \{\varnothing, (-\infty, M]_{\mathbb{E}}, [0, +\infty)_{\mathbb{O}}, (-\infty, 0)_{\mathbb{O}}, (M, +\infty)_{\mathbb{E}} \cup (-\infty, M]_{\mathbb{E}}, [0, +\infty)_{\mathbb{O}} \cup (-\infty, 0)_{\mathbb{O}}\}$, we have $\widetilde{\mathsf{pre}}(x) = \varnothing$, hence $\lfloor \mu c \rfloor\, \widetilde{\mathsf{pre}}(x) = \varnothing \in \mu$. Moreover, we have $\lfloor \mu c \rfloor\, \widetilde{\mathsf{pre}}((M, +\infty)_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}}) = \lfloor \mu c \rfloor ((\frac{M}{2}, +\infty)_{\mathbb{E}}) = \varnothing \in$

μ, and $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}((M, +\infty)_{\mathbb{E}} \cup (-\infty, 0)_{\mathbb{O}}) = \lfloor \mu c \rfloor ((M, +\infty)_{\mathbb{O}}) = \varnothing \in \mu$. Furthermore, $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}([0, +\infty)_{\mathbb{O}} \cup (-\infty, M]_{\mathbb{E}}) = \lfloor \mu c \rfloor ([0, \frac{M}{2})_{\mathbb{E}} \cup (-\infty, 0]_{\mathbb{O}}) = [0, 0]_{\mathbb{Z}} \in \mu$, and $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}((-\infty, M]_{\mathbb{E}} \cup (-\infty, 0)_{\mathbb{O}}) = \lfloor \mu c \rfloor ((-\infty, 0)_{\mathbb{E}} \cup (0, M]_{\mathbb{O}}) = \varnothing \in \mu$. Similarly, $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}((M, +\infty)_{\mathbb{E}} \cup (-\infty, M]_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}}) = \lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}((-\infty, M]_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}} \cup (-\infty, 0)_{\mathbb{O}}) = \lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}((M, +\infty)_{\mathbb{E}} \cup (-\infty, M]_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}} \cup (-\infty, 0)_{\mathbb{O}}) = [0, 0]_{\mathbb{Z}} \in \mu$, and $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}((M, +\infty)_{\mathbb{E}} \cup [0, +\infty)_{\mathbb{O}} \cup (-\infty, 0)_{\mathbb{O}}) = \lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}((M, +\infty)_{\mathbb{E}} \cup (-\infty, M]_{\mathbb{E}} \cup (-\infty, 0)_{\mathbb{O}}) = \varnothing \in \mu$. This shows that $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}(x) \in \mu$ holds for every $x \in \mu_i$. Finally, for the elements $x \in \mu \setminus \mu_i$, we have $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}([0, 0]_{\mathbb{E}}) = \lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}([0, 0]_{\mathbb{O}}) = \varnothing \in \mu$, $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}([0, 0]_{\mathbb{Z}}) = [0, 0]_{\mathbb{Z}} \in \mu$, $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}([0, \frac{M}{2}]_{\mathbb{E}}) = \varnothing \in \mu$, and $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}([0, \frac{M}{2}]_{\mathbb{E}} \cup [0, 0]_{\mathbb{O}}) = \varnothing \in \mu$. Thus, $\lfloor \mu c \rfloor \, \widetilde{\mathsf{pre}}(x) \in \mu$ for every $x \in \mu$, which proves that μ is $\lfloor \mu(c) \rfloor$-semi $\widetilde{\mathsf{pre}}$-complete. Consequently, $\mathrm{UP} = \varnothing$ and Algorithm 2 terminates. $\Diamond$

5 Abstract Model Checking

We apply our results in the context of model checking. We first discuss the role of unreachable states in the μ-calculus. Then, we recall the definition of abstract model checking and show how our refinement algorithm can be used to model check formulas over reachable states.

Given a finite set of atomic propositions $p \in \mathcal{P}$ and a set of variables $x \in \mathcal{X}$, μ-calculus formulas in $\mathscr{L}_1$ are defined as:

$$\mathscr{L}_1 \ni \varphi \coloneqq p \mid \bar{p} \mid x \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists \bigcirc \varphi \mid \forall \bigcirc \varphi \mid \mu\, x \colon \varphi \mid \nu\, x \colon \varphi \ .$$

The semantics of a formula $\varphi \in \mathscr{L}_1$ is defined for a TS $G = \langle \Sigma, I, \to \rangle$, a context $\mathcal{E} \colon \mathcal{X} \to \wp(\Sigma)$, a state labeling function such that $p(\Sigma) \subseteq \Sigma$ is the set of states satisfying an atomic proposition $p \in \mathcal{P}$. The semantics $[\![\varphi]\!]^G_{\mathcal{E}}$ of φ over G and $\mathcal{E}$ is the set of states on which φ holds, defined as:

$$\begin{gathered} [\![p]\!]^G_{\mathcal{E}} \triangleq p(\Sigma) \qquad [\![\bar{p}]\!]^G_{\mathcal{E}} \triangleq \Sigma \setminus p(\Sigma) \qquad [\![x]\!]^G_{\mathcal{E}} \triangleq \mathcal{E}(x) \\ [\![\varphi_1 \left\{\begin{smallmatrix}\vee\\ \wedge\end{smallmatrix}\right\} \varphi_2]\!]^G_{\mathcal{E}} \triangleq [\![\varphi_1]\!]^G_{\mathcal{E}} \left\{\begin{smallmatrix}\cup\\ \cap\end{smallmatrix}\right\} [\![\varphi_2]\!]^G_{\mathcal{E}} \qquad [\![\left\{\begin{smallmatrix}\exists\\ \forall\end{smallmatrix}\right\} \bigcirc \varphi]\!]^G_{\mathcal{E}} \triangleq \left\{\begin{smallmatrix}\mathsf{pre}\\ \widetilde{\mathsf{pre}}\end{smallmatrix}\right\} [\![\varphi]\!]^G_{\mathcal{E}} \\ [\![\left\{\begin{smallmatrix}\mu\\ \nu\end{smallmatrix}\right\} x \colon \varphi]\!]^G_{\mathcal{E}} \triangleq \left\{\begin{smallmatrix}\mathrm{lfp}\\ \mathrm{gfp}\end{smallmatrix}\right\} \left(\lambda X.\, [\![\varphi]\!]^G_{\mathcal{E}[x \mapsto X]}\right) \ . \end{gathered}$$

When clear from the context, superscripts will be omitted, and we define $[\![\varphi]\!]^G \triangleq [\![\varphi]\!]^G_{\mathcal{E}_\perp}$ where $\mathcal{E}_\perp$ is the empty context. For closed formulas (i.e., formulas where all variables are bound by a quantifier), and for any context $\mathcal{E}$, it holds that $[\![\varphi]\!]^G = [\![\varphi]\!]^G_{\mathcal{E}}$. Any subset, i.e. fragment, of $\mathscr{L}_1$ inherits the semantics $[\![\cdot]\!]^G_{\mathcal{E}}$.

We remark that removing unreachable state from a transition system has no effect on whether a reachable state r belongs to $[\![\varphi]\!]$. To formalize this, we introduce a modified semantics $(\!|\cdot|\!)^{G,S}_{\mathcal{E}}$ which is additionally parameterized by a set of states $S \subseteq \Sigma$. The semantics $(\!|\cdot|\!)^{G,S}_{\mathcal{E}}$ is defined inductively in the same way as $[\![\cdot]\!]^G$, except in the cases of $\forall\bigcirc$ and $\exists\bigcirc$, where we set:

$$(\!|\left\{\begin{smallmatrix}\exists\\ \forall\end{smallmatrix}\right\} \bigcirc \varphi|\!)^{G,S}_{\mathcal{E}} \triangleq \lfloor S \rfloor \left(\left\{\begin{smallmatrix}\mathsf{pre}\\ \widetilde{\mathsf{pre}}\end{smallmatrix}\right\} (\!|\varphi|\!)^{G,S}_{\mathcal{E}} \right) \ . \tag{2}$$

Intuitively, the definition in (2) ensures that only states in S are considered in the next-step semantics. In particular, if $\mathsf{post}^*(I) \subseteq S$, the operator $\lfloor S \rfloor$ effectively removes unreachable states from the semantics.

Theorem 5.1 (Model Checking for Reachable States). *Let $\varphi \in \mathscr{L}_1$ be a closed formula, $S \in \wp(\Sigma)$ satisfy $\mathsf{post}^*(I) \subseteq S$, and let $r \in \mathsf{post}^*(I)$. Then, $r \in [\![\varphi]\!]^G_{\mathcal{E}} \Leftrightarrow r \in (\!|\varphi|\!)^{G,S}_{\mathcal{E}}$.*

Hence, Theorem 5.1 shows that the validity of a formula φ on a reachable state r can be checked by testing whether $r \in (\!|\varphi|\!)^{G,S}$.

5.1 Abstract Semantics of μ-Calculus

Abstract domains can be equivalently defined as upper closures or as Galois connections [9]; in this section, where $C = \wp(\Sigma)$, we adopt Galois connections $\langle \alpha, C, A, \gamma \rangle$ for clarity of presentation, where $\alpha \colon C \to A$ maps concrete to abstract values, and $\gamma \colon A \to C$ is the corresponding concretization map. Abstract semantics for temporal logics play a central role in abstract model checking. This approach is well-studied (see, e.g., [1,5,6,11,13,17,18,21,23]), and relies on defining an abstract semantic function that assigns abstract values to formulas. We follow the framework of Ranzato and Tapparo [21], where the abstract semantic function $[\![\cdot]\!]^{\sharp A}_{\mathcal{E}} : \mathscr{L}_1 \to A$ is defined inductively by replacing each concrete semantic operator with its best correct approximation in A:

$$\begin{gathered}
[\![p]\!]^{\sharp A}_{\mathcal{E}} \triangleq \alpha(p(\Sigma)) \qquad [\![\bar{p}]\!]^{\sharp A}_{\mathcal{E}} \triangleq \alpha(\Sigma \setminus p(\Sigma)) \qquad [\![x]\!]^{\sharp A}_{\mathcal{E}} \triangleq \alpha(\mathcal{E}(x)) \\
[\![\varphi_1 \left\{\begin{smallmatrix}\vee\\ \wedge\end{smallmatrix}\right\} \varphi_2]\!]^{\sharp A}_{\mathcal{E}} \triangleq \alpha(\gamma[\![\varphi_1]\!]^{\sharp A}_{\mathcal{E}} \left\{\begin{smallmatrix}\cup\\ \cap\end{smallmatrix}\right\} \gamma[\![\varphi_2]\!]^{\sharp A}_{\mathcal{E}}) \quad [\![\left\{\begin{smallmatrix}\exists\\ \forall\end{smallmatrix}\right\} \bigcirc \varphi]\!]^{\sharp A}_{\mathcal{E}} \triangleq \alpha(\left\{\begin{smallmatrix}\mathsf{pre}\\ \widetilde{\mathsf{pre}}\end{smallmatrix}\right\} \gamma[\![\varphi]\!]^{\sharp A}_{\mathcal{E}}) \qquad (3) \\
[\![\left\{\begin{smallmatrix}\mu\\ \nu\end{smallmatrix}\right\} x \colon \varphi]\!]^{\sharp A}_{\mathcal{E}} \triangleq \left\{\begin{smallmatrix}\mathrm{lfp}\\ \mathrm{gfp}\end{smallmatrix}\right\} \left(\lambda X.\, [\![\varphi]\!]^{\sharp A}_{\mathcal{E}[x \mapsto \gamma(X)]}\right) .
\end{gathered}$$

By construction, $[\![\varphi]\!]^{\sharp A}_{\mathcal{E}}$ soundly approximates the concrete semantics, i.e., $\alpha[\![\varphi]\!]^G_{\mathcal{E}} \preceq_A [\![\varphi]\!]^{\sharp A}_{\mathcal{E}}$, while the converse need not hold. An abstraction A is *strongly preserving* for a language $\mathscr{L} \subseteq \mathscr{L}_1$ if, for all $\varphi \in \mathscr{L}$ and $X \in \wp(\Sigma)$, $\alpha(X) \preceq [\![\varphi]\!]^{\sharp A}_{\mathcal{E}} \Leftrightarrow X \subseteq [\![\varphi]\!]^G_{\mathcal{E}}$ [17, 21]. Strong preservation ensures equivalence between abstract and concrete validity for all formulas of $\mathscr{L}$. It is known [20, 21] that strong preservation corresponds to forward completeness for the concrete semantic primitives defining the language.[8] In this work, we focus on a weaker notion, *reachable strong preservation* (r.s.p.), which requires strong preservation only over reachable states.

Definition 5.2 (Reachable Strong Preservation). Given a transition system G, an abstraction A, and $\mathscr{L} \subseteq \mathscr{L}_1$, an abstract semantics $\langle\!\langle \cdot \rangle\!\rangle^A \colon \mathscr{L} \to A$ (mapping closed formulas to their abstract semantics in A) satisfies the *r.s.p. property* for $\mathscr{L}$ if, for every $S \subseteq \mathsf{post}^*(I)$, $S \subseteq [\![\varphi]\!]^G \Leftrightarrow \alpha(S) \preceq \langle\!\langle \varphi \rangle\!\rangle^A$.

[8] The continuity assumptions in [21] are satisfied here, since we work with finite abstractions.

In the following, we show how our algorithms yield abstract semantics satisfying the r.s.p. property for increasingly expressive fragments of $\mathscr{L}_1$, by requiring more stringent assumptions on the abstractions: from general ucos to preorder-induced ucos and finally equivalence-induced ucos.

Challenge 1: Model Checking for Reachable States. Let us consider the disjunction-free universal fragment $\mathscr{L}_2$ of the μ-calculus:

$$\mathscr{L}_2 \ni \varphi \coloneqq p \mid \bar{p} \mid x \mid \varphi \wedge \varphi \mid \forall\bigcirc\, \varphi \mid \mu\, x\colon \varphi \mid \nu\, x\colon \varphi \ .$$

We show how Algorithm 2 can be used to produce an abstract semantics satisfying the r.s.p. property of Definition 5.2 for $\mathscr{L}_2$. To this end, given a set of states $S \subseteq \Sigma$, we consider an abstract semantic function $(\!|\cdot|\!)^{\sharp A,S} : \mathscr{L}_2 \to A$, which is the abstract counterpart of $(\!|\cdot|\!)^{G,S}$. More precisely, $(\!|\cdot|\!)^{\sharp A,S}$ is defined inductively as $[\![\cdot]\!]^{\sharp A}$ in (3), replacing every occurrence of $[\![\cdot]\!]^{\sharp A}$ with $(\!|\cdot|\!)^{\sharp A,S}$, except for the $\exists\bigcirc$ and $\forall\bigcirc$ connectives, for which we define:

$$(\!|\left\{\begin{smallmatrix}\exists\\ \forall\end{smallmatrix}\right\} \bigcirc\, \varphi|\!)^{\sharp A,S}_{\mathcal{E}} \triangleq \alpha(\lfloor S\rfloor \left\{\begin{smallmatrix}\mathsf{pre}\\ \widetilde{\mathsf{pre}}\end{smallmatrix}\right\} \gamma[\![\varphi]\!]^{\sharp A}_{\mathcal{E}}) \ . \tag{4}$$

Before proceeding, we recall some basic notions on relations. A relation $R \subseteq \Sigma \times \Sigma$ is a preorder if it is reflexive and transitive, and an equivalence if it is also symmetric. It is known [22, § 5.2.1] that any preorder R (in particular, any equivalence) over Σ induces an additive uco $\mu \in \mathrm{uco}(\langle \wp(\Sigma), \subseteq\rangle)$ defined for any $X \in \wp(\Sigma)$ by $\mu(X) \triangleq R(X)$, where $R(S) \triangleq \{s' \in \Sigma \mid \exists s \in S.\, (s, s') \in R\}$. For brevity, we write $R(x) \triangleq R(\{x\}) = \mu(\{x\})$. Any equivalence relation defines a partition (and vice versa), where the blocks of the partition and the equivalence classes coincide. We therefore write P both for a partition and its associated equivalence relation, and the map $\lambda X.\, P(X) = \cup\{P(s) \in P \mid s \in X\}$ defines an additive uco. Finally, let $\equiv_{\mathcal{P}}$ denote the equivalence induced by the atomic propositions $\mathcal{P}$, i.e., $s \equiv_{\mathcal{P}} s' \overset{\triangle}{\iff} \forall p \in \mathcal{P}.\, s \in p(\Sigma) \Leftrightarrow s' \in p(\Sigma)$, and let $P_{\mathcal{P}}$ be the corresponding partition.

Theorem 5.3 (Abstract Model Checking for Reachable States). *If Algorithm 2 terminates on input $\langle P_{\mathcal{P}}, \langle \mathsf{post}, \widetilde{\mathsf{pre}}\rangle, I\rangle$ and outputs $\langle \mu, c\rangle$, then $(\!|\cdot|\!)^{\sharp\mu,\mu(c)}$ satisfies the r.s.p. property for $\mathscr{L}_2$.*

Theorem 5.3 ensures that Algorithm 2 produces abstractions suitable for model checking every formula in $\mathscr{L}_2$ over the reachable states.

Challenge 2: Disjunctive Abstractions. Let us now consider the following universal fragment $\mathscr{L}_3$:

$$\mathscr{L}_3 \ni \varphi \coloneqq p \mid \bar{p} \mid x \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \forall\bigcirc\, \varphi \mid \mu\, x\colon \varphi \mid \nu\, x\colon \varphi \ .$$

The fragment $\mathscr{L}_3$ is more expressive than $\mathscr{L}_2$, as it also allows disjunctions. Section 5.1 showed that Algorithm 2 supports abstract model checking of $\mathscr{L}_2$ formulas over reachable states. In order to achieve the same result for $\mathscr{L}_3$, we need to ensure that the resulting abstraction is additive (i.e., complete for unions), which is necessary for preserving disjunctions. To this end, we introduce a variant of Algorithm 2 where the uco μ is replaced by a preorder, thereby enforcing additivity. This modified procedure, Algorithm 2uni, is obtained from Algorithm 2 by substituting μ with a preorder R throughout the algorithm and by replacing lines (5–8) with the following primed lines:

```
5′ if $\mathrm{UP} \coloneqq \{\langle x,y\rangle \in \Sigma^2 \mid x \in R(c), x \in r(\{y\}), R(x) \not\subseteq r(R(y))\} \neq \varnothing$ then
6′     choose $\langle x,y\rangle \in \mathrm{UP}$;
7′     $R \coloneqq R \smallsetminus \{(z,z') \in R \mid z \in R(x);\ z' \notin r(R(y))\}$;
8′ end if
```

Algorithm 2uni exploits the additivity of R and the distributivity of the lattice $\langle \wp(\Sigma), \subseteq\rangle$ to restrict the semi-completeness check at line $5'$ to images of single states, rather than all abstract elements. The refinement step at line $7'$ updates the underlying preorder[9]. Importantly, every refinement preserves reflexivity and transitivity, so that R remains a preorder throughout the execution.

Theorem 5.4. *If Algorithm 2uni terminates on input* $\langle P_{\mathcal{P}}, \langle \mathsf{post}, \widetilde{\mathsf{pre}}\rangle, I\rangle$ *and outputs* $\langle R, c\rangle$*, then* $(\!|\cdot|\!)^{\sharp R, R(c)}$ *satisfies the r.s.p. property for* $\mathscr{L}_3$*.*

Challenge 3: Partitioning Abstractions. We now show how a further modification of Algorithm 2 achieves r.s.p. for the full language $\mathscr{L}_1$. Building on Section 5.1, we extend Algorithm 2uni so that semi-completeness is enforced also for pre, and not only $\widetilde{\mathsf{pre}}$, as above. As we previously replaced arbitrary ucos with preorders in order to guarantee additivity, we now further restrict the abstractions computed by the algorithm to those induced by *partitions*, that is, by symmetric preorders. The resulting procedure, Algorithm 2bis, is obtained from Algorithm 2 by replacing μ with a partition P and by substituting lines (5–8) with the following ones:

```
5″ if $\mathrm{UP} \coloneqq \{\langle x,y\rangle \in \Sigma^2 \mid x \in P(c), x \in r(\{y\}), P(x) \not\subseteq r(P(y))\} \neq \varnothing$ then
6″     choose $\langle x,y\rangle \in \mathrm{UP}$;
7″     $P \coloneqq (P \smallsetminus \{P(x)\}) \cup \{P(x) \cap r(P(y)),\ P(x) \smallsetminus r(P(y))\}$;
8″ end if
```

Algorithm 2bis exploits additivity in the same way as Algorithm 2uni, so that completeness is checked only on the blocks of the partition P, rather than on arbitrary abstract elements (i.e., unions of blocks). It enforces semi-completeness simultaneously for pre and $\widetilde{\mathsf{pre}}$, thus supporting the full $\mathscr{L}_1$ language.

Theorem 5.5. *If Algorithm 2bis terminates on input* $\langle P_{\mathcal{P}}, \langle \mathsf{post}, \widetilde{\mathsf{pre}}\rangle, I\rangle$ *and outputs* $\langle P, c\rangle$*, then* $(\!|\cdot|\!)^{\sharp P, P(c)}$ *satisfies the r.s.p. property for* $\mathscr{L}_1$*.*

[9] Each refinement might remove infinitely many pairs from R; however, with suitable symbolic representations of R, each update can be implemented in finitely many steps.

Let us observe that Algorithm 2bis solves the problem addressed by Lee and Yannakakis [16]. However, our framework is more general and extends their approach in three directions: (i) it supports arbitrary abstractions, not only partitioning ones (cf. Challenge 1); (ii) completeness can be enforced for any concrete function, not only for $\widetilde{\mathsf{pre}}$; and (iii) the notion of reachable state space is generalized to any semantically relevant region, specified by the lco ρ.

The three constructions presented in Challenges 1-3 show how increasingly expressive fragments of the μ-calculus can be supported by progressively stronger classes of abstractions. Starting from arbitrary ucos, Algorithm 2 yields abstractions ensuring the r.s.p. property for the disjunction-free fragment $\mathscr{L}_2$. By restricting abstractions to preorder-induced ucos, Algorithm 2uni guarantees additivity, thereby enabling reachable-state model checking for the fragment $\mathscr{L}_3$ with disjunctions. Finally, by further restricting abstractions to partitions, Algorithm 2bis enforces semi-completeness for both $\widetilde{\mathsf{pre}}$ and pre, which suffices to obtain the r.s.p. property for the full μ-calculus $\mathscr{L}_1$.

6 Conclusion and Future Work

We addressed limitations of classical abstract model checking, where strong preservation typically requires globally complete abstractions. By introducing semi-completeness, we provide a principled relaxation of forward completeness that enables more practical refinements. Parameterized by lower closure operators, semi-completeness enforces completeness only over semantically relevant regions rather than globally. We presented two refinement procedures: one enforcing semi-completeness for a given region, and another interleaving refinement with the region's fixed point computation. Both avoid constructing fully complete abstractions and are especially effective when verifying reachable states. Applied to μ-calculus model checking, our framework yields reachability-aware abstract semantics for relevant fragments, reducing verification effort.

Several directions remain for future work. First, we plan to further investigate the algorithmic aspects of our approach, including complexity bounds, suitable symbolic representations, and the role of non-determinism in the refinement process. Second, integrating the proposed interleaving procedure into existing model checking frameworks would enable empirical evaluation and the development of practical optimizations. Finally, we aim to extend our framework to probabilistic and real-time systems, thereby broadening its applicability, and to further explore connections with related areas such as deductive verification.

Acknowledgements. Francesco Ranzato was partially supported by the *Italian MUR* under the PRIN 2022 PNRR project no. P2022HXNSC; by a *WhatsApp Research Award*; and by an *Amazon Research Award* for "AWS Automated Reasoning". This publication is also supported by grant PID2022-138072OB-I00, funded by MCIN/AEI/10.13039/501100011033/ FEDER, UE; by grant PIPF-2022/COM-24370, funded by the Madrid Regional Government; and by grant CEX2024-001471-M, funded by MICIU/AEI/10.13039/501100011033.

References

1. Banda, G., Gallagher, J.P.: Constraint-based abstract semantics for temporal logic: A direct approach to design and implementation. In: Proceedings 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning, LPAR 2010. Lecture Notes in Computer Science, vol. 6355, pp. 27–45. Springer (2010). https://doi.org/10.1007/978-3-642-17511-4_3
2. Bonchi, F., Ganty, P., Giacobazzi, R., Pavlovic, D.: Sound Up-to Techniques and Complete Abstract Domains. In: Proceedings 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018. pp. 175–184. ACM, New York, NY, USA (2018). https://doi.org/10.1145/3209108.3209169
3. Bradley, A.R.: SAT-based model checking without unrolling. In: Proceedings 12th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2011. Lecture Notes in Computer Science, vol. 6538, pp. 70–87. Springer (2011). https://doi.org/10.1007/978-3-642-18275-4_7
4. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proceedings 12th International Conference on Computer Aided Verification, CAV 2000. Lecture Notes in Computer Science, vol. 1855, pp. 154–169. Springer (2000). https://doi.org/10.1007/10722167_15
5. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement for symbolic model checking. J. ACM **50**(5), 752–794 (2003). https://doi.org/10.1145/876638.876643
6. Clarke, E.M., Grumberg, O., Long, D.E.: Model checking and abstraction. ACM Trans. Program. Lang. Syst. **16**(5), 1512–1542 (1994). https://doi.org/10.1145/186025.186051
7. Cousot, P.: Principles of Abstract Interpretation. MIT Press, Cambridge (2021)
8. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: Proceedings ACM POPL 1977. pp. 238–252. ACM (1977). https://doi.org/10.1145/512950.512973
9. Cousot, P., Cousot, R.: Systematic design of program analysis frameworks. In: Proceedings ACM POPL 1979. pp. 269–282. ACM Press (1979). https://doi.org/10.1145/567752.567778
10. Cousot, P., Ganty, P., Raskin, J.: Fixpoint-guided abstraction refinements. In: Proceedings 14th International Static Analysis Symposium, SAS 2007. Lecture Notes in Computer Science, vol. 4634, pp. 333–348. Springer (2007). https://doi.org/10.1007/978-3-540-74061-2_21
11. Dams, D., Grumberg, O.: Abstraction and abstraction refinement. In: Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R. (eds.) Handbook of Model Checking, pp. 385–419. Springer (2018). https://doi.org/10.1007/978-3-319-10575-8_13
12. Filé, G., Giacobazzi, R., Ranzato, F.: A unifying view of abstract domain design. ACM Comput. Surv. **28**(2), 333–336 (1996). https://doi.org/10.1145/234528.234742
13. Giacobazzi, R., Ranzato, F.: Incompleteness of states w.r.t. traces in model checking. Inf. Comput. **204**(3), 376–407 (2006). https://doi.org/10.1016/J.IC.2006.01.001
14. Henzinger, T.A., Majumdar, R., Raskin, J.: A classification of symbolic transition systems. ACM Trans. Comput. Log. **6**(1), 1–32 (2005). https://doi.org/10.1145/1042038.1042039

15. Hoder, K., Bjørner, N.S.: Generalized property directed reachability. In: Proceedings 15th International Conference on Theory and Applications of Satisfiability Testing, SAT 2012. Lecture Notes in Computer Science, vol. 7317, pp. 157–171. Springer (2012). https://doi.org/10.1007/978-3-642-31612-8_13
16. Lee, D., Yannakakis, M.: Online Minimization of Transition Systems. In: Proc. of the 24th Annual ACM Symposium on Theory of Computing, STOC '92. pp. 264–274. ACM (1992). https://doi.org/10.1145/129712.129738
17. Loiseaux, C., Graf, S., Sifakis, J., Bouajjani, A., Bensalem, S., Probst, D.: Property preserving abstractions for the verification of concurrent systems. Formal Methods in System Design **6**(1), 11–44 (1995). https://doi.org/10.1007/bf01384313
18. Massé, D.: Semantics for abstract interpretation-based static analyzes of temporal properties. In: Proc. 9th International Static Analysis Symposium, SAS 2002. Lecture Notes in Computer Science, vol. 2477, pp. 428–443. Springer (2002). https://doi.org/10.1007/3-540-45789-5_30
19. Ranzato, F., Rossi-Doria, O., Tapparo, F.: A forward-backward abstraction refinement algorithm. In: Proceedings 9th International Conference on Verification, Model Checking, and Abstract Interpretation, VMCAI 2008. Lecture Notes in Computer Science, vol. 4905, pp. 248–262. Springer (2008). https://doi.org/10.1007/978-3-540-78163-9_22
20. Ranzato, F., Tapparo, F.: Making abstract model checking strongly preserving. In: Proceedings 9th International Symposium on Static Analysis, SAS 2002. Lecture Notes in Computer Science, vol. 2477, pp. 411–427. Springer (2002). https://doi.org/10.1007/3-540-45789-5_29
21. Ranzato, F., Tapparo, F.: Generalized strong preservation by abstract interpretation. Journal of Logic and Computation **17**(1), 157–197 (2007). https://doi.org/10.1093/LOGCOM/EXL035
22. Ranzato, F., Tapparo, F.: Generalizing the Paige–Tarjan algorithm by abstract interpretation. Information and Computation **206**(5), 620–651 (2008). https://doi.org/10.1016/J.IC.2008.01.001
23. Shoham, S., Grumberg, O.: Monotonic abstraction-refinement for CTL. In: Proceedings 10th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2004. Lecture Notes in Computer Science, vol. 2988, pp. 546–560. Springer (2004). https://doi.org/10.1007/978-3-540-24730-2_40