Metric Spaces and Termination Analyses

Aziem Chawdhary[1,2] and Hongseok Yang[2]

[1] Durham University
[2] Queen Mary University of London

Abstract. We present a framework for defining abstract interpreters for liveness properties, in particular program termination. The framework makes use of the theory of metric spaces to define a concrete semantics, relates this semantics with the usual order-theoretic semantics of abstract interpretation, and identifies a set of conditions for determining when an abstract interpreter is sound for analysing liveness properties. Our soundness proof of the framework is based on a novel relationship between unique fixpoints in metric semantics and post-fixpoints computed by abstract interpreters. We illustrate the power of the framework by providing an instance that can automatically prove the termination of programs with general (not necessarily tail) recursion.

1 Introduction

Recently, there has been great interest in the automatic verification of program termination. Quite a few techniques for automatically verifying termination or general liveness properties of imperative programs have been proposed [1, 2, 4–8, 16–18], some of which have led to successful tools, such as TERMINATOR [7].

In this paper, we step back from all these technological advances, and reexamine a theoretical foundation of automatic techniques for verifying termination or liveness properties of programs. Most of the proposed techniques are based on abstracting programs (in addition to clever results on well-founded relations such as [3, 19]), but these abstraction methods are justified by rather ad-hoc arguments [4]. This is in contrast with the soundness of abstraction for safety properties, which follows a standard framework of abstract interpretation [10, 11]. Our aim is to develop a theory that provides a similar systematic answer for when an abstraction is sound for proving liveness properties. By doing so, we want to relieve the burden of inventing a new way of proving soundness from designers of liveness analysis.

Our main result is a new framework for developing sound precise abstract interpreters for liveness properties of programs with general recursion. Technically, the key feature of our framework is to use a concrete semantics based on metric space [13, 14, 20] and to spell out a condition under which this concrete metric-space semantics can be related to a usual order-theoretic semantics of abstract interpretation. We illustrate the power of the framework by providing an instance that can automatically prove the termination of recursive procedures.

Our framework uses a metric-space semantics, because such a semantics justifies a novel strategy for computing approximate fixpoints during abstract interpretation for *liveness*. Imagine that we want to develop a sound termination analysis. Our analysis needs to overapproximate the set of all computation traces of a given program and to check whether the overapproximation does not include an infinite trace. In the standard order-theoretic setting, the set of computation traces of a program is defined in terms of the greatest fixpoint of some function F [9], but overapproximating the greatest fixpoint of F precisely wrt. termination is difficult. For instance, a post-fixpoint x of F (i.e., $F(x) \sqsubseteq x$), which is normally computed by an abstract interpreter for safety, does not overapproximate the greatest fixpoint in general. Hence, fixpoint-computation strategies from safety analyses cannot be used for termination analysis without changes. Alternatively, one might consider the following sequence converging to the greatest fixpoint of F (under the assumption of the continuity of F):

$$\top \sqsupseteq F(\top) \sqsupseteq F^2(\top) \sqsupseteq F^3(\top) \sqsupseteq \dots$$

and want to compute an overapproximating sequence $\{x_n\}$ such that $F^n(\top) \sqsubseteq x_n$ for all n, and $x_m = x_{m+1}$ for some m. In this case, a fixpoint-computation strategy finds this x_m, and returns it as a result. The problem here is that the strategy is very imprecise; it cannot prove termination of most nontrivial programs (especially those whose time complexity is not constant).

The metric-space semantics of our framework resolves this overapproximation issue. It defines the set of computation traces of a program in terms of a *unique fixpoint* of a function G, and then it guarantees that this unique fixpoint can be overapproximated by a post-fixpoint of G, as long as the post-fixpoint lives in a restricted semantic universe, such as the one with the *closed* sets of traces.[3] Thus, when developing a sound termination analysis in our framework, one can re-use fixpoint-computation strategies from existing safety analyses (which compute post-fixpoints), after adjusting the strategies so that computed post-fixpoints live in the restricted universe.

Using a metric-space semantics has another benefit that our framework can hide call stacks, which appear in a small-step operational semantics of recursive procedures. Hence, a user of the framework does not need to worry about abstracting call stacks [15], and can focus on the problem of proving a desired liveness property.

Related Work Among the automatic techniques for proving program termination cited already, we discuss two techniques further [4, 8]. The first is our previous work [4], where we proved the soundness of a termination analysis, by directly relating greatest fixpoints in the concrete trace semantics with post-fixpoints computed by the termination analysis. Our proof relied on the fact that the language contained only tail recursions so that greatest fixpoints could be

[3] A trace set is closed iff all Cauchy sequences in the set have limits in the set. We will explain it further in the main part of the paper.

$$e \ ::= \ x \mid r \mid e+e \mid r \times e \qquad b \ ::= \ e=e \mid e \neq e \mid e \leq e \mid e < e \mid b \wedge b \mid b \vee b \mid \neg b$$
$$c \ ::= \ x := e \mid c; c \mid \texttt{if}\, b\, c\, c \mid f() \mid \texttt{fix}\, f.\, c$$

Fig. 1. Programming Language with General Recursion

rephrased in terms of least fixpoints and infinite iterations. This rewriting is not applicable if a programming language includes non-tail recursions. In contrast, the framework of this paper can handle programs with general recursion.

The second technique is a termination analysis for recursive procedures in [8]. This technique works by replacing each recursive function call by a non-deterministic choice between entering a procedure body (in the case that the procedure does not terminate) or the application of a summary of the procedure (in the case that the procedure does terminate). The instance of our framework in this paper can be seen as a modified version of this technique where program transformations are done on the fly and termination proofs and procedure summarizations are done at the same time.

Recently Cousot et al. [12] defined bi-inductive domains to account for both infinite and finite program properties. They combine a domain for finite behaviours with another for infinite behaviours, and produce a new domain whose order is defined using the orders from the two underlying domains. A least fixpoint on this new domain can overapproximate the union of the least fixpoint in the finite domain and the greatest fixpoint in the infinite domain. However, the semantic functions may not be monotone with respect to the order of the new domain, and so cannot be computed by the usual fixpoint iteration. This limitation means that we once again have to reason about least and greatest fixpoints, a situation that we avoid in this paper by using metric spaces.

2 Programming Language

Let PName be the set of procedures names, ranged over by f, g, and let Var be a finite set of program variables x, y that contain rational numbers in $\mathbb{Q}$. We consider a simple imperative language with parameterless procedures f, g and rational variables x, y. The grammar of the language is given in Fig. 1, where we use r to denote a rational constant.

Most commands in our language are standard. The only unusual case is the definition of recursive procedure $\texttt{fix}\, f.\, c$. It defines a recursive procedure f whose body is c, and then it immediately calls the defined procedure. Note that while loops can be expressed in this language using recursion. We write $\Gamma \vdash c$ for a finite subset Γ of PName, where Γ includes all the free function names in c.

3 Framework

In this section we describe our framework for developing a sound abstract interpreter for liveness properties. Throughout the paper, we will use $\mathbb{N}$ for the set of *positive* integers.

3.1 Review on Metric Spaces

We start with a brief review on metric spaces. For further information on metric semantics, we refer the reader to the standard book and survey on this topic [13, 20].

A **metric space** is a non-empty set X with a function $d_X : X \times X \to [0, \infty)$, called metric, that satisfies the three conditions below:

1. Identity of indiscernible: $\forall x, y \in X.\ d_X(x, y) = 0 \iff x = y$.
2. Symmetry: $\forall x, y \in X.\ d_X(x, y) = d_X(y, x)$.
3. Triangular inequality: $\forall x, y, z \in X.\ d_X(x, z) \leq d_X(x, y) + d_X(y, z)$.

Consider a sequence $\{x_n\}_{n\in\mathbb{N}}$ in a metric space (X, d_X). The sequence $\{x_n\}_{n\in\mathbb{N}}$ is **Cauchy** iff for all real numbers $\epsilon > 0$, there exists some $N \in \mathbb{N}$ such that $\forall m, n \geq N.\ d_X(x_m, x_n) \leq \epsilon$. The sequence $\{x_n\}_{n\in\mathbb{N}}$ **converges to** x **in** X iff for all real numbers $\epsilon > 0$, there exists an $N \in \mathbb{N}$ such that $\forall m \geq N.\ d_X(x_m, x) \leq \epsilon$.

A metric space X is **complete** iff every Cauchy sequence converges to some element in X. In this paper, we will consider only complete metric spaces.

Let (X, d_X) and (Y, d_Y) be metric spaces and let α be a positive real number. A function $F : X \to Y$ is **non-expansive** iff for all $x, x' \in X$, we have that $d_Y(F(x), F(x')) \leq d_X(x, x')$. It is α**-contractive** iff $d_Y(F(x), F(x')) \leq \alpha \times d_X(x, x')$ holds for all $x, x' \in X$. Intuitively, the non-expansiveness means that F does not increase the distance between elements, and the contractiveness says that F actually decreases the distance.

In this paper, we use the well-known Banach's unique fixpoint theorem:

Theorem 1 (Banach's Unique Fixpoint Theorem). *Let (X, d_X) be a metric space. If X is complete and a function $F : X \to X$ is α-contractive for some $0 \leq \alpha < 1$, the function F has the unique fixpoint. Furthermore, this unique fixpoint can be obtained as follows: first pick an arbitrary x_1 in X, then construct the sequence $\{x_n\}_{n\in\mathbb{N}}$ with $x_{n+1} = F(x_n)$ and finally take the limit of this sequence.*[4]

We will denote the unique fixpoint of F by $\mathsf{ufix}(F)$.

3.2 Concrete Metric-Space Semantics

Our framework consists of two parts. The first part is a concrete semantics based on metric spaces. It is parameterized by the data below, which should be provided by a user of the framework:

[4] This limit always exists, because the constructed sequence is Cauchy.

1. A pre-ordered complete metric space $(\mathcal{D}, d, \sqsubseteq, \top)$ with the biggest element $\top$. We require that for all Cauchy sequences $\{x_n\}_{n\in\mathbb{N}}$ in $\mathcal{D}$ and all $x \in \mathcal{D}$,
$$(\forall n \in \mathbb{N}.\ x_n \sqsubseteq x) \implies \lim_{n\to\infty} x_n \sqsubseteq x. \tag{1}$$
Elements of $\mathcal{D}$ can be understood as semantic counterparts of syntactic commands; our concrete semantics interprets a command c as an element in $\mathcal{D}$.
2. Monotone non-expansive functions seq, $\mathsf{asgn}_{x,e}$ and if_b for all assignments $x{:=}e$ and all boolean conditions b:
$$\mathsf{seq} : \mathcal{D} \times \mathcal{D} \to \mathcal{D}, \qquad \mathsf{asgn}_{x,e} : \mathcal{D}, \qquad \mathsf{if}_b : \mathcal{D} \times \mathcal{D} \to \mathcal{D}.$$
These functions define the meaning of the sequencing, assignment and conditional statements in our language.
3. A function $\mathsf{proc} : \mathsf{PName} \to \mathcal{D} \to \mathcal{D}$ for modelling the execution of procedures. We write proc_f instead of $\mathsf{proc}(f)$, and require that $\mathsf{proc}_f(-)$ be a monotone $\frac{1}{2}$-contractive function for all $f \in \mathsf{PName}$. Intuitively, an input x to $\mathsf{proc}_f(-)$ denotes all the possible computations by the body of the procedure f, and $\mathsf{proc}_f(x)$ extends each of these computations with steps taken immediately before or after running the procedure body during the call of f.
4. A subset $\textsc{LivProperty}$ of $\mathcal{D}$ that is downward closed with respect to $\sqsubseteq$:
$$x \sqsubseteq y \ \wedge\ y \in \textsc{LivProperty} \implies x \in \textsc{LivProperty}.$$
This subset consists of elements in $\mathcal{D}$ (which are semantic counterparts of commands) satisfying a desired liveness property, such as termination.

Note that the semantic domain $\mathcal{D}$ here has both pre-order and metric-space structures and that the semantic operators respect both structures by being monotone and non-expansive. These two structures are related by the requirement (1) on $\sqsubseteq$ and Cauchy sequences. One important consequence of the relationship is the lemma below, and it will play a crucial role for the soundness of our framework:

Lemma 1. *For all $\frac{1}{2}$-contractive monotone functions $F : \mathcal{D} \to \mathcal{D}$, a post-fixpoint of F overapproximates the unique fixpoint of F. That is, if x satisfies $F(x) \sqsubseteq x$, we have that $\mathsf{ufix}\,F \sqsubseteq x$, where $\mathsf{ufix}\,F$ is the unique fixpoint of F.*

Proof. Let x be a post-fixpoint of F. By the Banach fixpoint theorem, we know that the unique fixpoint $\mathsf{ufix}\,F$ of F exists and is also the limit of the following Cauchy sequence:
$$x,\ F(x),\ F^2(x),\ F^3(x),\ \ldots$$
Since x is a post-fixpoint of F (i.e., $F(x) \sqsubseteq x$) and F is monotone,
$$x \ \sqsupseteq\ F(x)\ \sqsupseteq\ F^2(x)\ \sqsupseteq\ F^3(x)\ \sqsupseteq\ F^4(x)\ \ldots$$
That is, $F^n(x) \sqsubseteq x$ for all n. Thus, the limit $\mathsf{ufix}\,F$ of $\{F^n(x)\}_{n\in\mathbb{N}}$ also satisfies $\mathsf{ufix}\ F \sqsubseteq x$ by the requirement (1) of our framework. We have just proved the lemma. □

$$[\![\Gamma \vdash c]\!] \;:\; [\![\Gamma]\!] \to \mathcal{D}$$

$$\begin{array}{rl}
[\![\Gamma \vdash f()]\!]\eta = \eta(f) & [\![\Gamma \vdash c_1; c_2]\!]\eta = \mathsf{seq}([\![\Gamma \vdash c_1]\!]\eta, [\![\Gamma \vdash c_2]\!]\eta) \\
[\![\Gamma \vdash x{:=}e]\!]\eta = \mathsf{asgn}_{x,e} & [\![\Gamma \vdash \mathtt{if}\, b\, c_1\, c_2]\!]\eta = \mathsf{if}_b([\![\Gamma \vdash c_1]\!]\eta, [\![\Gamma \vdash c_2]\!]\eta) \\
[\![\Gamma \vdash \mathtt{fix}\, f.c]\!]\eta = \mathsf{ufix}\, F & \text{(where } F(x) = \mathsf{proc}_f([\![\Gamma, f \vdash c]\!]\eta[f \mapsto x]))
\end{array}$$

Fig. 2. Concrete Semantics defined by the Framework

The domain $\mathcal{D}$ and the operators above give rise to a metric-space semantics of programs. Let $[\![\Gamma]\!]$ be the domain for procedure environments (i.e., $\Pi_{f \in \Gamma}\mathcal{D}$), pre-ordered pointwise and given the product metric, where the distance between η and η' in $\Pi_{f \in \Gamma}\mathcal{D}$ is given by $\max_{f \in \Gamma} d(\eta(f), \eta'(f))$. The semantics interprets $\Gamma \vdash c$ as a non-expansive map from $[\![\Gamma]\!]$ to $\mathcal{D}$, and it is given in Fig. 2.

Note that the semantics defines $\mathtt{fix}\, f.c$ as the unique fixpoint of a function F modelling the meaning of the procedure body c. To ensure the existence of the fixpoint here, the semantics maintains that all commands denote only non-expansive functions. Then, it defines the function F in terms of non-expansive $[\![\Gamma, f \vdash c]\!]$ and 1/2-contractive proc_f, and ensures that F is 1/2-contractive. Hence, by the Banach fixpoint theorem, F has the unique fixpoint.

Lemma 2. *For all commands $\Gamma \vdash c$, $[\![\Gamma \vdash c]\!]$ is a well-defined non-expansive function from $[\![\Gamma]\!]$ to $\mathcal{D}$. Furthermore, $[\![\Gamma \vdash c]\!]$ is monotone.*

The use of metric spaces means that in order to design an instance of our generic framework one now needs to prove certain properties of the concrete semantics. Firstly, one has to prove that the semantic domain $\mathcal{D}$ for the meaning of commands is a complete metric space, in addition to having a pre-order structure. Secondly, one needs to show that all the semantic operators are non-expansive.

These new proof obligations often make it impossible to re-use an existing concrete semantics. For instance, a naive trace semantics, such as the one in [4], uses the powerset of traces as a semantic universe for commands, but this powerset cannot be used in our framework. This is because it does not form a complete metric space, when it is given a natural notion of distance function. In order to use the framework in this paper, one has to modify the powerset of traces, so that it has a good metric-theoretic structure, as will be done in Sec. 4.

However, these obligations come with a reward—the soundness of an order-theoretic abstract semantics, which is to be presented next.

3.3 Abstract Semantics

The second part of our framework is the abstract semantics. For a function $f : X^n \to X$ and a subset X_0 of X, we say that f can be restricted to X_0 if for all $\boldsymbol{x} \in X_0^n$, we have that $f(\boldsymbol{x}) \in X_0$. Using this terminology, we describe the parameters of our abstract semantics:

1. A set $\mathcal{A}$ with a partition $\mathcal{A}_p \uplus \mathcal{A}_t = \mathcal{A}$. The elements of $\mathcal{A}$ provide abstract meanings of commands. We call elements in $\mathcal{A}_t$ *total* and those in $\mathcal{A}_p$ *partial*. The set $\mathcal{A}$ should come with the additional data below.
   (a) Distinguished elements $\bot$ and $\top$ in $\mathcal{A}$ such that $\top \in \mathcal{A}_t$.
   (b) An algorithm $\mathsf{checktot}$ that answers the membership to $\mathcal{A}_t$ soundly but not necessarily in a complete way. That is, $\mathsf{checktot}(A) = \mathsf{true}$ means that $A \in \mathcal{A}_t$, but $\mathsf{checktot}(A) \neq \mathsf{true}$ does not mean that $A \notin \mathcal{A}_t$.
   (c) A concretization function $\gamma : \mathcal{A}_t \to \mathcal{D}$, such that $\gamma(\top) = \top$. Note that the domain of γ is $\mathcal{A}_t$, not $\mathcal{A}$.
2. Functions $\mathsf{seq}^\sharp$, $\mathsf{asgn}^\sharp_{x,e}$ and $\mathsf{if}^\sharp_b$ for all assignments $x{:=}e$ and booleans b:

$$\mathsf{seq}^\sharp : \mathcal{A} \times \mathcal{A} \to \mathcal{A}, \qquad \mathsf{asgn}^\sharp_{x,e} : \mathcal{A}, \qquad \mathsf{if}^\sharp_b : \mathcal{A} \times \mathcal{A} \to \mathcal{A}.$$

   These functions give the abstract meaning of the sequencing, assignment and conditional statements in our language. We require that these functions can be restricted to $\mathcal{A}_t$, and that they overapproximate their concrete counterparts:

$$\begin{aligned} \forall A_0, A_1 \in \mathcal{A}_t.\ \mathsf{seq}(\gamma(A_0), \gamma(A_1)) &\sqsubseteq \gamma(\mathsf{seq}^\sharp(A_0, A_1)) \\ \wedge \quad \mathsf{asgn}_{x,e} &\sqsubseteq \gamma(\mathsf{asgn}^\sharp_{x,e}) \\ \wedge \quad \mathsf{if}_b(\gamma(A_0), \gamma(A_1)) &\sqsubseteq \gamma(\mathsf{if}^\sharp_b(A_0, A_1)). \end{aligned}$$

   Note that this soundness condition is only relevant for total elements in $\mathcal{A}_t$.
3. A function $\mathsf{proc}^\sharp : \mathsf{PName} \to \mathcal{A} \to \mathcal{A}$ for modelling the execution of procedures. For all $f \in \mathsf{PName}$, we require that $\mathsf{proc}^\sharp{}_f$ can be restricted to $\mathcal{A}_t$, and that it should overapproximate proc_f:

$$\forall f \in \mathsf{PName}.\quad \forall A \in \mathcal{A}_t.\quad \mathsf{proc}_f(\gamma(A)) \sqsubseteq \gamma(\mathsf{proc}^\sharp{}_f(A)).$$

4. A predicate $\textsc{satisfyLiv}^\sharp$ on $\mathcal{A}_t$ such that

$$\forall A \in \mathcal{A}_t.\quad \textsc{satisfyLiv}^\sharp(A) = \mathsf{true} \implies \gamma(A) \in \textsc{LivProperty}.$$

   Intuitively, $\textsc{satisfyLiv}^\sharp$ identifies abstract elements denoting commands with a desired liveness property.
5. A widening operator $\triangledown : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ [10]. This operator needs to satisfy three conditions. Firstly, it can be restricted to a map from $\mathcal{A}_t$. Secondly, it overapproximates an upper bound of its right argument: $\gamma(A_2) \sqsubseteq \gamma(A_1 \triangledown A_2)$ for all $A_1, A_2 \in \mathcal{A}_t$. Finally, it turns any sequences in $\mathcal{A}$ into one with a stable element. That is, for all $\{A_n\}_{n\in\mathbb{N}}$ in $\mathcal{A}$, the widened sequence $\{A'_n\}_{n\in\mathbb{N}}$ with $A'_1 = A_1$ and $A'_{n+1} = A_n \triangledown A_{n+1}$ contains an index m with $A'_m = A'_{m+1}$.

Note that among the abstract elements in $\mathcal{A}$, only total ones in $\mathcal{A}_t$ have meanings in the concrete domain $\mathcal{D}$ via γ. That is, elements in $\mathcal{A}_p$ need not be concretizable in $\mathcal{D}$. The absence of the concretization relationship between $\mathcal{A}_p$ and $\mathcal{D}$ is intended, because it allows an analysis designer to use a flexible fixpoint strategy during abstract interpretation. Concretely, even though an abstract interpreter aims to compute a value in $\mathcal{D}$ (more precisely, $\{\gamma(A) \mid A \in \mathcal{A}_t\}$)

$$[\![\Gamma \vdash c]\!]^\sharp \;:\; [\![\Gamma]\!]^\sharp \to \mathcal{A}$$

$$\begin{array}{rlrl}
[\![\Gamma \vdash f()]\!]^\sharp \eta^\sharp & = \eta^\sharp(f) & [\![\Gamma \vdash c_1; c_2]\!]^\sharp \eta^\sharp & = \mathsf{seq}^\sharp([\![\Gamma \vdash c_1]\!]^\sharp \eta^\sharp,\ [\![\Gamma \vdash c_2]\!]^\sharp \eta^\sharp) \\
[\![\Gamma \vdash x{:=}e]\!]^\sharp \eta^\sharp & = \mathsf{asgn}^\sharp_{x,e} & [\![\Gamma \vdash \mathtt{if}\, b\, c_1\, c_2]\!]^\sharp \eta^\sharp & = \mathsf{if}^\sharp_b([\![\Gamma \vdash c_2]\!]^\sharp \eta^\sharp,\ [\![\Gamma \vdash c_2]\!]^\sharp \eta^\sharp) \\
[\![\Gamma \vdash \mathtt{fix}\, f.c]\!]^\sharp \eta^\sharp & = \lceil \mathsf{widenfix}\ F \rceil & \multicolumn{2}{l}{(\text{where } F(A) = \mathsf{proc}^\sharp_f([\![\Gamma, f \vdash c]\!]^\sharp \eta^\sharp[f \mapsto A]))}
\end{array}$$

Fig. 3. Abstract Semantics defined by the Framework

at the end of a fixpoint computation, it can temporarily step outside of $\mathcal{D}$ and use elements in $\mathcal{A}_p$ during the computation, as long as its final result is an element in $\mathcal{D}$. We found this flexibility very useful for achieving high precision in our framework; in order to have a complete metric-space structure, a concrete domain $\mathcal{D}$ often does not include certain semantic elements, such as the empty set, that could serve as the meaning of intermediate results of a precise fixpoint-computation strategy of an abstract interpreter.

The parameters given above are enough to induce an abstract semantics of programs, but to do so, we need to define two operators using the parameters. The first operator is the ceiling $\lceil - \rceil$, which replaces partial elements by $\top$:

$$\lceil A \rceil \;=\; \textbf{if}\ (\mathsf{checktot}(A) = \mathsf{true})\ \textbf{then}\ A\ \textbf{else}\ \top.$$

The second is the widened fixpoint operator $\mathsf{widenfix}$. Given a function $F : \mathcal{A} \to \mathcal{A}$, the operator constructs the sequence $\{A_n\}_{n \in \mathbb{N}}$ with $A_1 = \bot$ and $A_{n+1} = A_n \triangledown F(A_n)$. Then, it returns the first A_m with $A_m = A_{m+1}$. The condition on $\triangledown$ ensures that such A_m exists.

Let $[\![\Gamma]\!]^\sharp$ be the abstract domain for procedure environments (i.e., $[\![\Gamma]\!]^\sharp = \Pi_{f \in \Gamma} \mathcal{A}$). The abstract semantics interprets programs $\Gamma \vdash c$ as functions from $[\![\Gamma]\!]^\sharp$ to $\mathcal{A}$. The defining clauses in the semantics are given in Fig. 3.

The semantics in Fig. 3 are mostly standard, but the abstract semantics of $\mathsf{fix}\, f.c$ deserves attention. After computing a widened fixpoint, $[\![\Gamma \vdash \mathsf{fix}\, f.c]\!]^\sharp$ checks whether the fixpoint is a total element. If not, $[\![\Gamma \vdash \mathsf{fix}\, f.c]\!]^\sharp$ approximates the fixpoint by $\top$, which should be total by the requirement of the framework. This additional step and the requirements of our framework ensure one important property of the semantics:

Lemma 3. *For all $\Gamma \vdash c$ and $\eta^\sharp \in [\![\Gamma]\!]^\sharp$, if $\eta^\sharp(f) \in \mathcal{A}_t$ for every $f \in \Gamma$, we have that $[\![\Gamma \vdash c]\!]^\sharp \eta^\sharp \in \mathcal{A}_t$.*

Intuitively, the lemma says that $[\![\Gamma \vdash c]\!]^\sharp$ can be restricted to total elements. Using this lemma, we express the soundness of the abstract semantics:

$$\forall \eta^\sharp \in [\![\Gamma]\!]^\sharp.\ \ (\forall f \in \Gamma.\ \ \eta^\sharp(f) \in \mathcal{A}_t) \implies [\![\Gamma \vdash c]\!]\gamma(\eta^\sharp) \sqsubseteq \gamma([\![\Gamma \vdash c]\!]^\sharp \eta^\sharp). \qquad (2)$$

In $\gamma(\eta^\sharp)$ above, we use the componentwise extension of γ to procedure environments. Note that although γ is not defined on partial elements, the soundness claim above is well-formed, because Lemma 3 ensures that $[\![\Gamma \vdash c]\!]^\sharp \eta^\sharp$ is total. We prove the soundness in the next theorem:

Theorem 2. *The abstract semantics is sound. That is,* (2) *holds for all* $\mathit{\Gamma} \vdash c$.

Proof (Sketch). Our proof is by induction on the structure of c. Here we focus on the most interesting case that $c \equiv \mathtt{fix}\, f.c_1$, where we can see the interaction between the metric structure and the pre-order structure of $\mathcal{D}$. Let $F\colon \mathcal{D} \to \mathcal{D}$ and $G\colon \mathcal{A} \to \mathcal{A}$ be functions defined by

$$F(x) \;=\; \mathsf{proc}_f([\![\mathit{\Gamma}, f \vdash c_1]\!]\gamma(\eta^\sharp)[f \mapsto x]), \;\; G(A) \;=\; \mathsf{proc}^\sharp{}_f([\![\mathit{\Gamma}, f \vdash c_1]\!]^\sharp\eta^\sharp[f \mapsto A]).$$

We need to prove that

$$(\mathsf{ufix}\, F) \;\sqsubseteq\; \gamma(\lceil \mathsf{widenfix}\, G \rceil). \tag{3}$$

If $\mathsf{checktot}(\mathsf{widenfix}\;\; G) \neq \mathsf{true}$, then $\gamma(\lceil \mathsf{widenfix}\;\; G\rceil) = \gamma(\top) = \top$. Thus, (3) holds. Suppose that $\mathsf{checktot}(\mathsf{widenfix}\, G) = \mathsf{true}$, which implies that $\mathsf{widenfix}\, G \in \mathcal{A}_t$. In this case, it is sufficient to prove that $\gamma(\mathsf{widenfix}\; G)$ is a post-fixpoint of F. Because then, the inequality (3) follows from Lemma 1. By the definition of $\mathsf{widenfix}$, $(\mathsf{widenfix}\, G) \;=\; (\mathsf{widenfix}\, G) \mathbin{\triangledown} G(\mathsf{widenfix}\, G)$. Because of the condition on $\triangledown$, this implies that

$$\gamma(G(\mathsf{widenfix}\, G)) \;\sqsubseteq\; \gamma(\mathsf{widenfix}\, G). \tag{4}$$

The LHS of (4) is greater than or equal to $F(\gamma(\mathsf{widenfix}\, G))$ as shown below:

$$\begin{aligned} \gamma(G(\mathsf{widenfix}\, G)) \;&=\; \gamma\big(\mathsf{proc}^\sharp{}_f\; [\![\mathit{\Gamma}, f \vdash c_1]\!]^\sharp\eta^\sharp[f \mapsto (\mathsf{widenfix}\, G)]\big) \\ &\sqsupseteq\; \mathsf{proc}_f\big(\gamma\big([\![\mathit{\Gamma}, f \vdash c_1]\!]^\sharp\eta^\sharp[f \mapsto (\mathsf{widenfix}\, G)]\big)\big) \\ &\sqsupseteq\; \mathsf{proc}_f\big([\![\mathit{\Gamma}, f \vdash c_1]\!]\gamma(\eta^\sharp)[f \mapsto \gamma(\mathsf{widenfix}\, G)]\big) \\ &=\; F(\gamma(\mathsf{widenfix}\, G)). \end{aligned} \tag{5}$$

The first inequality holds because $\mathsf{proc}^\sharp$ overapproximates proc. The second follows from the induction hypothesis and the monotonicity of proc_f. The inequalities in (4) and (5) imply the desired $F(\gamma(\mathsf{widenfix}\, G)) \sqsubseteq \gamma(\mathsf{widenfix}\, G)$. □

3.4 Generic Analysis

Let $\eta^\sharp_*$ be the unique abstract environment for the empty context $\mathit{\Gamma} = \emptyset$. Our generic analysis takes a command c with no free procedures, and computes the function: $\textsc{LivAnalysis}(c) \;=\; \textsc{satisfyLiv}^\sharp([\![c]\!]^\sharp\eta^\sharp_*)$. The result is a boolean value, indicating whether c satisfies a liveness property specified by $\textsc{LivProperty}$.

Theorem 3. *Let* η_* *be the unique concrete environment for the empty context* $\mathit{\Gamma} = \emptyset$*. Then, for all commands* c *with no free procedures, if* $\textsc{LivAnalysis}(c) = \mathsf{true}$*, we have that* $[\![c]\!]\eta_* \in \textsc{LivProperty}$*.*

4 Instance of the Framework

In this section, we instantiate the framework and define a sound abstract interpreter for proving the termination of programs with general recursion.

4.1 Concrete Semantics

Our instance of concrete semantics of the framework interprets commands as sets of traces satisfying certain healthiness conditions. The notion of traces here is slightly unusual, because the traces are sequences of *tagged states* and they need to meet our well-formedness conditions. In this section, we will explain the meanings of tagged states and traces, and provide parameters necessary for instantiating a concrete semantics from the framework.

Tagged States, Pre-traces and Traces We start with the definition of traces. A **state** is a map from program variables to rational numbers, and a **tagged state** is a pair of state and tag:

$$\mathsf{Tag} = \{none\} \cup \mathsf{PName} \times \{call, ret\}, \quad \mathsf{State} = \mathsf{Var} \to \mathbb{Q}, \quad \mathsf{tState} = \mathsf{State} \times \mathsf{Tag}.$$

The tag of a tagged state indicates whether the state is the initial or the final state of a procedure call, or just a normal one not related to a call. The $(f, call)$ and (f, ret) tags mean that the state is, respectively, the initial and the final state of the call $f()$, and the *none* tag indicates that the state is a normal state, i.e., it is neither the initial nor the final state of a procedure call. We use symbol σ to denote elements in tState, and use s to denote elements in State.

A **pre-trace** τ is a nonempty finite or infinite sequence of tagged states, such that τ starts with a *none*-tagged state and if it is finite, it ends with a *none*-tagged state.

$$\mathsf{nState} = \mathsf{State} \times \{none\}, \quad \mathsf{preTrace} = \mathsf{nState}(\mathsf{tState}^*)\mathsf{nState} \cup \mathsf{nState}(\mathsf{tState}^\infty),$$

where tState^∞ means the set of (countably) infinite sequences of tagged states.

A trace τ is a pre-trace that satisfies well-formedness conditions. To define these conditions, we consider the sets $\mathcal{W}, \mathcal{O}$ of sequences of tagged states that are the least fixpoints of the below equations:

$$\begin{aligned} \mathcal{W} &= \mathsf{nState}^* \cup \mathcal{W}\mathcal{W} \cup \Big(\textstyle\bigcup_{f \in \mathsf{PName}, s, s_1 \in \mathsf{State}} \{(s, (f, call))\}\, \mathcal{W}\, \{(s_1, (f, ret))\}\Big), \\ \mathcal{O} &= \mathcal{W} \cup \mathcal{O}\mathcal{O} \cup \Big(\textstyle\bigcup_{f \in \mathsf{PName}, s \in \mathsf{State}} \{(s, (f, call))\}\mathcal{O}\Big). \end{aligned}$$

Intuitively, $\mathcal{W}$ describes sequences where every procedure call has a matching return and calls and returns are well-bracketed. The other set $\mathcal{O}$ defines a bigger set; in each trace in $\mathcal{O}$, some procedure calls might not have matching returns, but calls and returns should be well-bracketed.

Definition 1. *A pre-trace τ is a* **trace** *iff τ is finite and belongs to $\mathcal{W}$, or τ is infinite and all of its finite prefixes are in $\mathcal{O}$. We write Trace for the set of traces.*

For $\tau \in \mathsf{Trace}$ and $n \in \mathbb{N} \cup \{\infty\}$, the projection $\tau[n]$ is the n-prefix of τ; in case that $|\tau| < n$, $\tau[n] = \tau$.[5] Using this projection, we define the distance function on traces as follows:

$$d(\tau, \tau') = 2^{-\max\{n \mid \tau[n] = \tau'[n]\}} \qquad \text{(where we regard } 2^{-\infty} = 0\text{)}.$$

[5] $\tau[n]$ does not necessarily belong to Trace or even to $\mathsf{preTrace}$, but this will not cause problems for our results.

$$\begin{aligned}
\mathsf{seq}(T,T') &= \{\tau\sigma\tau' \mid (\tau\sigma \in T \cap \mathsf{tState}^{+}) \wedge (\sigma\tau' \in T')\} \cup (T \cap \mathsf{tState}^{\infty})\\
\mathsf{asgn}_{x,e} &= \{\sigma\sigma' \mid \sigma,\sigma' \in \mathsf{nState} \wedge \mathsf{first}(\sigma') = \mathsf{first}(\sigma)[x \mapsto [\![e]\!]\mathsf{first}(\sigma)]\}\\
\mathsf{if}_b(T_0,T_1) &= \{\sigma\tau \mid (\sigma\tau \in T_0 \wedge [\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}) \vee (\sigma\tau \in T_1 \wedge [\![b]\!](\mathsf{first}(\sigma)) = \mathsf{false})\}\\
\mathsf{proc}_f(T) &= \{\sigma\sigma^{(f,call)}\tau \mid \sigma\tau \in (T \cap \mathsf{tState}^{\infty})\} \cup \{\sigma\sigma^{(f,call)}\sigma^{(f,ret)}\sigma \mid \sigma \in T\}\\
&\quad \cup \{\sigma\sigma^{(f,call)}\tau\sigma_1^{(f,ret)}\sigma_1 \mid \sigma\tau\sigma_1 \in (T \cap \mathsf{tState}^{+})\}
\end{aligned}$$

Here $\mathsf{first}(\sigma)$ is the first component of the tagged state σ, and $\sigma^{(f,call)}$ and $\sigma^{(f,ret)}$ are, respectively, $(\mathsf{first}(\sigma),(f,\mathit{call}))$ and $(\mathsf{first}(\sigma),(f,\mathit{ret}))$. And $[\![b]\!]$ and $[\![e]\!]$ are the standard interpretation of booleans and expressions as functions from (untagged) states to $\{\mathsf{true},\mathsf{false}\}$ and $\mathbb{Q}$.

Fig. 4. Semantic Operators for the Instance Concrete Semantics

Lemma 4. (Trace, d) *is a complete metric space.*

Full Closed Sets of Well-formed Traces A subset $T_0 \subseteq \mathsf{Trace}$ of traces is **closed** if for all Cauchy sequences of traces in T_0, their limits belong to T_0 as well. A trace set $T_0 \subseteq \mathsf{Trace}$ is **full** if for every *none*-tagged state $\sigma \in \mathsf{nState}$, there is a trace $\tau \in T_0$ starting with σ.

The semantic domain $(\mathcal{D}, d)$ for interpreting commands in our concrete semantics is the set $\mathcal{P}_{fcl}(\mathsf{Trace})$ of *full closed* sets of traces:

$$\mathcal{D} \;=\; \mathcal{P}_{fcl}(\mathsf{Trace}), \qquad d^{\dagger}(T,T') \;=\; 2^{-\max\{n \mid T[n]=T'[n]\}}.$$

Here $T[n]$ is the result of taking the prefix of every trace in T (i.e., $T = \{\tau[n] \mid \tau \in T\}$). The closedness ensures that the $d^{\dagger}$ just defined satisfies the axioms for being a complete metric space. Also, the condition about being full allows us to meet the non-expansiveness requirement for seq in our framework.

Our domain $\mathcal{D}$ is ordered by the subset relation $\subseteq$. With respect to this $\subseteq$ order, $\mathcal{D}$ has the top element, which is the set Trace of all traces.

Lemma 5. $(\mathcal{D}, d^{\dagger})$ *is a complete metric space. Furthermore, the requirement* (1) *of our framework in Sec. 3 holds for* $\subseteq$ *and this metric space.*

Semantic Operators So far we have defined the semantic domain for commands, the first required parameter of the framework. The next four parameters are operators working on this domain, and we describe them in Fig. 4. In the figure, the sequencing operator seq concatenates traces from T and T', while treating infinite traces from T specially. And the operator proc_f duplicates initial and final states, and tags the duplicated states with information about procedure call and return.

Lemma 6. *All the operators are well-defined, and satisfy the monotonicity and non-expansiveness or* $\frac{1}{2}$*-contractiveness requirements of our framework.*

$$E \;::=\; r \mid x \mid {`x} \mid x' \mid E+E \mid r\times E \qquad P \;::=\; E=E \mid E\neq E \mid E<E \mid E\leq E$$
$$\varphi \;::=\; P \mid \mathsf{true} \mid \varphi\wedge\varphi \mid \mathsf{false} \mid \varphi\vee\varphi \mid \exists x'.\,\varphi$$

Fig. 5. Syntax for Linear Constraints

Liveness Property The only remaining parameter is LivProperty, which describes a desired liveness property on trace sets. Here we use a property such that if we restrict our attention to $T = [\![c]\!]\eta$ of some command c with no free procedure names, the membership of T to this property implies that T consists of finite traces only.

We say that a trace τ includes an infinite subsequence of open calls iff there exists $\{\tau_n\sigma_n\}_{n\in\mathbb{N}}$ such that

1. $\tau \;=\; \tau_1\sigma_1\tau_2\sigma_2\tau_3\sigma_3\ldots\tau_n\sigma_n\ldots$,
2. for all $i\in\mathbb{N}$, there exists some $f\in\mathsf{PName}$ such that $\mathsf{second}(\sigma_i) = (f, \mathit{call})$,
3. for all $i\in\mathbb{N}$, the corresponding return for σ_i does not appear in τ after σ_i, i.e., the return does not occur in the sequence $\tau_{i+1}\sigma_{i+1}\tau_{i+2}\sigma_{i+2}\ldots$.

We specify a desired liveness property of (semantic) commands, using the following subset LivProperty of $\mathcal{D}$: T is in LivProperty iff no traces in T include an infinite subsequence of open calls.

4.2 Abstract Semantics with Linear Ranking Relations

Our abstract semantics uses formulas φ for linear constraints. The syntax of these formulas is given in Fig. 5. Note that a formula φ can use three kinds of variables: normal program variables x; pre-primed ones ${`x}$ for denoting the value of $x$ before running a program; primed ones $x'$ that can be existentially quantified. We assume that the set $\mathsf{Var}$ of normal variables and the set ${`\mathsf{Var}}$ of pre-primed variables are finite and that there is an one-to-one correspondence between Var and ${`\mathsf{Var}}$, which maps $x$ to ${`x}$.

Let Form be the set of formulas φ that do not contain free primed variables. Each $\varphi\in\mathsf{Form}$ defines a relation from (untagged) states ${`s}$ with pre-primed variables (i.e., ${`s}\in{`\mathsf{Var}}\to\mathbb{Q}$) to (untagged) states s with normal variables:

$$({`s}, s) \;\models\; \varphi,$$

where $\models$ is the standard satisfaction relation from the first-order logic. Let TForm be a subset of Form consisting of *total* formulas in the sense below:

$$\mathsf{TForm} \;=\; \{\varphi\in\mathsf{Form} \mid \forall{`s}\in({`\mathsf{Var}}\to\mathbb{Q}).\,\exists s\in(\mathsf{Var}\to\mathbb{Q}).\,({`s},s)\models\varphi\}.$$

The abstract semantics in this section assumes a sound but possibly incomplete theorem prover that can answer queries of the two kinds: $\varphi\vdash\psi$ and $\vdash\forall{`X}.\,\exists X.\,\varphi$. Here ${`X}$ and X are the sets of free pre-primed variables and normal variables in φ. Note that by asking the query of the second kind, we can use a prover to check, soundly, whether a formula φ belongs to TForm.

Using what we have defined or assumed so far, we define an abstract domain $\mathcal{A}$ and its subset $\mathcal{A}_t$ of total abstract elements as follows:

$$\mathcal{A} = \mathsf{Form} \times \mathsf{Form} \times \mathsf{Form}, \quad \mathcal{A}_t = \mathsf{TForm} \times \mathsf{Form} \times \mathsf{Form}, \quad \mathcal{A}_p = \mathcal{A} - \mathcal{A}_t.$$

The element $(\mathsf{false}, \mathsf{false}, \mathsf{false})$ in $\mathcal{A}$ serves the role of $\bot$, and $(\mathsf{true}, \mathsf{true}, \mathsf{true})$ the role of $\top$. The algorithm for soundly checking the totality of abstract elements is defined using the assumed prover:

$$\mathsf{checktot}(A) \;=\; \textbf{if} \;\; (\vdash \forall `X.\exists X.\, A_1) \;\; \textbf{then} \;\; \mathsf{true} \;\; \textbf{else} \;\; \mathsf{unknown}$$

where A_i is the i-th component of A and $`X$ and X are the sets of free pre-primed and free normal variables in A_1.

Next, we define the concretization map γ, which will provide the intuitive meaning of abstract elements in $\mathcal{A}$. To do this, we need to introduce some additional notations. Firstly, for a (untagged) state s, we write $`s$ for the state obtained from s by renaming normal variables by corresponding pre-primed ones. Secondly, we write $\sigma \in \tau$ to mean that σ is a tagged state appearing in τ, and $\mathsf{iscall}(\sigma)$ to mean that the tag for σ is a procedure call:

$$\mathsf{iscall}(\sigma) \iff \exists f \in \mathsf{PName}.\, (\mathsf{second}(\sigma) = (f, \mathit{call})).$$

Finally, for all tagged states $\sigma_1, \sigma_2 \in \tau$, we say that σ_1 is an open call with respect to σ_2 in τ, denoted $\mathsf{open}(\sigma_1, \sigma_2, \tau)$, if both σ_1 and σ_2 are tagged with procedure calls, σ_1 appears strictly before σ_2 in τ, but the corresponding return for σ_1 does not appear before σ_2. The concretization is defined as follows:

$$\begin{aligned}\gamma(A) = \{\tau \in \mathsf{Trace} \mid\; & (\tau \in \mathsf{tState}^{+} \implies (`\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\mathsf{last}(\tau))) \models A_1) \;\wedge \\ & (\forall \sigma \in \tau.\, \mathsf{iscall}(\sigma) \implies (`\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\sigma)) \models A_2) \;\wedge \\ & (\forall \sigma_1, \sigma_2.\, \mathsf{open}(\sigma_1, \sigma_2, \tau) \implies (`\mathsf{first}(\sigma_1), \mathsf{first}(\sigma_2)) \models A_3)\}.\end{aligned}$$

Here A_i is the i-th component of A. According to this concretization, A_1 relates the initial and final states of a trace τ, and A_2 and A_3 describe the relationship between certain intermediate states in τ; A_2 relates the initial state and a call state in τ, and A_3 relates states at two open calls in τ. Tracking the relationship between intermediate states is crucial for the precision of our analysis. If the abstract domain included only the first component (as in our previous work [4]), the concretizations of its elements would contain traces violating LivProperty, or they would not belong to $\mathcal{D}$.

Lemma 7. *For every $A \in \mathcal{A}_t$, the set $\gamma(A)$ is in $\mathcal{D}$, i.e., it is full and closed.*

Abstract Operators For $\varphi, \psi \in \mathsf{Form}$, let $\varphi; \psi$ be their relational composition defined by

$$\varphi; \psi \;\equiv\; \exists Y'.(\varphi[Y'/X] \wedge \psi[Y'/`X]).$$

$$\begin{aligned}
\mathsf{seq}^\sharp(A, A') &= (\ A_1; A'_1,\ \ A_2 \vee (A_1; A'_2),\ \ A_3 \vee A'_3\ ) \\
\mathsf{asgn}^\sharp_{x,e} &= (\ eq_{\mathsf{Var}-\{x\}} \wedge (e[`x/x] = x),\ \mathsf{false},\ \mathsf{false}\ ) \\
\mathsf{if}^\sharp_b(A, A') &= \textbf{let}\ b_1 = \mathsf{preprime}(b)\ \text{ and }\ b_2 = \mathsf{preprime}(\mathsf{neg}(b)) \\
&\quad\ \textbf{in}\ (\ (b_1 \wedge A_1) \vee (b_2 \wedge A'_1),\ \ (b_1 \wedge A_2) \vee (b_2 \wedge A'_2),\ \ A_3 \vee A'_3\ ) \\
\mathsf{proc}^\sharp{}_f(A) &= (\ A_1,\ eq_{\mathsf{Var}} \vee A_2,\ A_2 \vee A_3\ )
\end{aligned}$$

Here $\mathsf{preprime}(b)$ renames all the variables with the corresponding pre-primed variables, and $\mathsf{neg}(b)$ is the negation of b where $\neg$ is removed by being pushed all the way down to atomic predicates using logical equivalences. For instance, $\mathsf{neg}(x{=}y{\vee}z{<}3)$ is $x{\neq}y{\wedge}3{\leq}z$.

Fig. 6. Semantic Operators for the Instance Abstract Semantics

Here X and $`X$ respectively contain normal variables in φ and pre-primed variables in ψ, Y' is the set of fresh primed variables, and the cardinalities of these three sets are the same so that the substitution in $\varphi;\psi$ is well-defined. Also, for a set X of normal variables, define the formula eq_X to be the equality on the variables in X and the corresponding pre-primed ones: $eq_X \ \equiv\ \bigwedge_{x\in X}(`x = x)$.

Using these notations, we present abstract operators in Fig. 6. Note that the abstract sequencing $\mathsf{seq}^\sharp(A, A')$ is not simply the relational composition of formulas; it also describes relationships between intermediate states of a trace. For instance, the second component $A_2 \vee (A_1; A'_2)$ relates the initial state of a trace with states at procedure call in the trace. The first disjunct A_2 considers the case that a call state is from the first argument A of the sequencing, and the second $A_1; A'_2$ is for the other case that a call is from the second argument A'.

Lemma 8. *The operators in Fig. 6 meet all the requirements of our framework.*

Widening Operator Our widening operator is parameterized by three elements. The first is a positive integer k, which bounds the number of outermost disjuncts in formulas appearing in the results of widening. We will write ∇_k to make this parameterization explicit. The second is a function lower that overapproximates a formula φ in Form by the conjunction of lower bounds on some pre-primed variables (i.e., the conjunction of formulas of the form $r \leq `x$ for some *pre-primed* variable $`x$ and rational number r):

$$\mathsf{lower}(\varphi) \quad = \quad (r_1 \leq `x_1\ \wedge\ r_2 \leq `x_2\ \wedge\ \ldots r_n \leq `x_n)$$

such that φ entails $\mathsf{lower}(\varphi)$ semantically. The third is the dual of the second function. It is a function upper that overapproximates a formula φ in Form by the conjunction of formulas of the form $`x \leq r$.

The widening operator uses three subroutines. The first is toDNF that transforms a formula $\varphi \in \mathsf{Form}$ to a disjunctive normal form, where all existential quantifications are placed right before each conjunct. The second is the function $\mathsf{bound}_k : \mathsf{Form} \to \mathsf{Form}$ for bounding the number of outermost disjuncts to k:

$$\mathsf{bound}_k(\varphi) \ = \ \textbf{if}\ \ (\text{at most } k \text{ outermost disjuncts are in } \varphi)\ \ \textbf{then}\ \ \varphi\ \ \textbf{else}\ \ \mathsf{true}$$

The third is an algorithm RFS that synthesizes a linear ranking function from φ. such as RANKFINDER in [18]. Semantically, unless RFS returns fail, it computes an overapproximation of a disjunction-free formula $\varphi \in \mathsf{Form}$, and the overapproximation expresses a linear ranking relation, such as $10 < {`x} \wedge x \leq {`x}{-}1$ for the ranking function x.

Using these parameters and subroutines, we can now define the widening operator:

$$
\begin{aligned}
A \mathbin{\triangledown_k} A' = \textbf{let } (\textstyle\bigvee_{j \in J_i} \kappa^i_j) &= \mathsf{toDNF}(A'_i) \qquad (i = 1, 2, 3 \text{ here and below}) \\
\chi^i_j &= \textstyle\bigwedge\{ {`x} = x \mid x \in \mathsf{Var} \ \text{ and } \ \kappa^i_j \vdash {`x} = x \} \\
\xi^i_j &= \textbf{if } \big(\ \mathsf{RFS}(\kappa^i_j) = \zeta^i_j \ \text{ for some formula } \zeta^i_j\ \big) \\
&\quad\ \ \textbf{then } \big(\ \zeta^i_j \wedge \mathsf{lower}(\kappa^i_j) \wedge \mathsf{upper}(\kappa^i_j) \wedge \chi^i_j\ \big) \\
&\quad\ \ \textbf{else } \big(\ \mathsf{lower}(\kappa^i_j) \wedge \mathsf{upper}(\kappa^i_j) \wedge \chi^i_j\ \big) \\
\delta_i &= \mathsf{bound}_k(A_i \vee \textstyle\bigvee_{j \in J_i} \{ \xi^i_j \mid \kappa^i_j \not\vdash A_i \}) \\
\textbf{in } (\delta_1, \delta_2, \delta_3).
\end{aligned}
$$

Lemma 9. *The operator $\triangledown_k : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ is a widening operator.*

Abstract Liveness Predicate The abstract semantics uses the following predicate $\textsc{satisfyLiv}^\sharp$ on $\mathcal{A}_t$ and checks whether an analysis result implies the desired liveness property:

$$
\begin{aligned}
\textsc{satisfyLiv}^\sharp(A) \ = \ & \textbf{let } (\textstyle\bigvee_{i \in I} \delta_i) = \mathsf{toDNF}(A_3) \\
& \textbf{in } \big(\textbf{if } (\mathsf{RFS}(\delta_i) \neq \mathsf{fail} \text{ for all } i \in I) \textbf{ then } \mathsf{true} \textbf{ else } \mathsf{false}\big).
\end{aligned}
$$

The predicate $\textsc{satisfyLiv}^\sharp$ first transforms A_3 to a disjunctive normal form. Then, it checks whether each disjunct δ_i is well-founded using the function RFS. Hence, if the predicate returns true, it means that A_3 is disjunctively well-founded. The below lemma is an easy consequence of the disjunctively well-foundedness of A_3, the result of Podelski and Rybalchenko [19] and the definition of γ.

Lemma 10. *For all $A \in \mathcal{A}_t$, if $\textsc{satisfyLiv}^\sharp(A) = \mathsf{true}$, we have that $\gamma(A) \in$ LIVPROPERTY.*

5 Conclusion

In this paper, we have presented a framework for designing a sound abstract interpreter for liveness properties. The framework incorporates the theory of metric spaces in the concrete semantics. By doing so, it justifies a new strategy for approximating fixpoints for an abstract interpreter for liveness, and relieves the burden of abstracting low-level details from an analysis designer. We hope that our results help the program analysis community to exploit metric space semantics and other unexplored areas of the semantics research for developing effective program analysis algorithms.

Acknowledgments We would like to thank Martin Escardo for suggesting to look at metric-space semantics, Xavier Rival for explaining the relationship with Cousot's work, and Andrey Rybalchenko for encouraging us to think about the cons and pros of a high-level compositional approach on program analysis, compared to a low-level one based on transition systems. Peter O'Hearn has been always supportive. and provided helpful suggestions. Chawdhary was supported by EPSRC grant EP/G042322/1 and Microsoft Research PhD Scholarship, and Yang by EPSRC grant EP/E053041/1.

References

1. I. Balaban, A. Pnueli, and L. Zuck. Ranking abstraction as companion to predicate abstraction. In *FORTE'05*, 2005.
2. A. Bradley, Z. Manna, and H. Sipma. Termination of polynomial programs. In *VMCAI'05*, 2005.
3. A. R. Bradley, Z. Manna, and H. B. Sipma. The polyranking principle. In *ICALP'05*, 2005.
4. A. Chawdhary, B. Cook, S. Gulwani, M. Sagiv, and H. Yang. Ranking abstractions. In *ESOP'08*, 2008.
5. M. Codish and C. Taboch. A semantic basis for the termination analysis of logic programs. *JLP*, 41(1):103–123, 1999.
6. B. Cook, A. Gotsman, A. Podelski, A. Rybalchenko, and M. Y. Vardi. Proving that programs eventually do something good. In *POPL'07*, 2007.
7. B. Cook, A. Podelski, and A. Rybalchenko. Termination proofs for systems code. In *PLDI'06*, 2006.
8. B. Cook, A. Podelski, and A. Rybalchenko. Summarization for termination: no return! *Formal Methods in System Design*, 35(3):369–387, 2009.
9. P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *Theor. Comput. Sci.*, 277(1-2):47–103, 2002.
10. P. Cousot and R. Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM Symposium on Principles of Programming Languages*, pages 238–252, January 1977.
11. P. Cousot and R. Cousot. Abstract interpretation frameworks. *Journal of Logic and Computation*, 2(4):511–547, 1992.
12. P. Cousot and R. Cousot. Bi-inductive structural semantics. *Information and Computation*, 207(2):258–283, 2009.
13. J. de Bakker and E. de Vink. *Control flow semantics*. MIT Press, Cambridge, MA, USA, 1996.
14. M. Escardó. A metric model of PCF. *unpublished research note*, 1998.
15. B. Jeannet and W. Serwe. Abstracting call-stacks for interprocedural verification of imperative programs. In *Int. Conf. on Algebraic Methodology and Software Technology, AMAST'04*, volume 3116 of *LNCS*, July 2004.
16. D. Kroening, N. Sharygina, A. Tsitovich, and C. Wintersteiger. Termination analysis with compositional transition invariants. In *CAV'10*, 2010. To appear.
17. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. *SIGPLAN Not.*, 36(3):81–92, 2001.
18. A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. In *VMCAI'04*, 2004.

19. A. Podelski and A. Rybalchenko. Transition invariants. In *LICS'04*, 2004.
20. F. van Breugel. An introduction to metric semantics: operational and denotational models for programming and specification languages. *Theoretical Computer Science*, 258(1-2):1 – 98, 2001.

A Missing Proofs in the Framework Section

In this appendix, we give the proofs missing in Sec. 3.

A.1 Missing Proof in the Concrete Semantics Section

Lemma 2. For all commands $\mathit{\Gamma} \vdash c$, $[\![\mathit{\Gamma} \vdash c]\!]$ is a well-defined non-expansive function from $[\![\mathit{\Gamma}]\!]$ to $\mathcal{D}$. Furthermore, $[\![\mathit{\Gamma} \vdash c]\!]$ is monotone.

Proof. We first prove that $[\![\mathit{\Gamma} \vdash c]\!]$ is a well-defined non-expansive function from $[\![\mathit{\Gamma}]\!]$ to $\mathcal{D}$. The proof is by induction on the structure of c. The cases of function call and assignment follow from the fact that both projection functions and constant functions are well-defined and non-expansive. The induction goes through for the cases of the sequential composition and the if statement, because of the induction hypothesis and the non-expansiveness requirements on seq and if_b.

The remaining case is the recursion: $\mathit{\Gamma} \vdash \mathtt{fix}\, f.c$. Pick environments η, η' and let F, G be functions defined by

$$F(k) \;=\; \mathsf{proc}_f([\![\mathit{\Gamma}, f \vdash c]\!]\eta[f \mapsto k]), \quad G(k) \;=\; \mathsf{proc}_f([\![\mathit{\Gamma}, f \vdash c]\!]\eta'[f \mapsto k]).$$

Firstly, we prove the well-definedness. For this, it is sufficient to prove that F is $\frac{1}{2}$-contractive, so that we can apply the Banach fixpoint theorem, which implies that the unique fixpoint of F exists. We can use this unique fixpoint to interpret recursion. To prove the contractiveness of F, consider k, k'. By induction hypothesis, $[\![\mathit{\Gamma}, f \vdash c]\!]$ is a well-defined non-expansive map. Thus:

$$d(k, k') \quad \geq \quad d\big([\![\mathit{\Gamma}, f \vdash c]\!]\eta[f \mapsto k],\; [\![\mathit{\Gamma}, f \vdash c]\!]\eta[f \mapsto k']\big). \tag{6}$$

In our framework we have required that $\mathsf{proc}_f(-)$ be $\frac{1}{2}$-contractive. Hence:

$$\begin{aligned} & d([\![\mathit{\Gamma}, f \vdash c]\!]\eta[f \mapsto k],\; [\![\mathit{\Gamma}, f \vdash c]\!]\eta[f \mapsto k']) \\ & \geq\; \tfrac{1}{2} \times d(\mathsf{proc}_f([\![\mathit{\Gamma}, f \vdash c]\!]\eta[f \mapsto k]),\; \mathsf{proc}_f([\![\mathit{\Gamma}, f \vdash c]\!]\eta[f \mapsto k'])) \\ & \geq\; \tfrac{1}{2} \times d(F(k), F(k')). \end{aligned} \tag{7}$$

Putting the conclusions of (6) and (7), we get that $d(k, k') \geq (\frac{1}{2} \times d(F(k), F(k')))$, the $\frac{1}{2}$-contractiveness of F.

Secondly, we prove that $[\![\mathit{\Gamma} \vdash \mathtt{fix}\, f.c]\!]$ defines a non-expansive function. Since η, η' are chosen arbitrary, it is sufficient to show that

$$d(\eta,\, \eta') \;\;\geq\;\; d(\mathsf{ufix}\, F,\, \mathsf{ufix}\, G).$$

Pick k from $\mathcal{D}$. By the Banach fixpoint theorem, both $\{F^n(k)\}_{n\in\omega}$ and $\{G^n(k)\}_{n\in\omega}$ are Cauchy sequences with limits $\mathsf{ufix}\, F$ and $\mathsf{ufix}\, G$, respectively.

We claim that the n-th elements of these two sequences are close to each other:

$$\forall n \in \omega. \quad d(F^n(k), G^n(k)) \;\leq\; d(\eta,\, \eta'). \tag{8}$$

This claim can be proved by induction on n. When $n = 0$, the LHS of the inequality is zero, so the claim holds. Suppose that $n > 0$ and also that the claim holds for all $m < n$. By the induction hypothesis,

$$d(F^{n-1}(k), G^{n-1}(k)) \;\leq\; d(\eta,\, \eta').$$

Then,

$$d\big(\,\eta[f \mapsto F^{n-1}(k)],\;\; \eta'[f \mapsto G^{n-1}(k)]\,\big) \;\;\leq\;\; d(\eta,\, \eta').$$

Now, the non-expansiveness of $[\![\Gamma, f \vdash c]\!]$ and proc_f implies that

$$d\big(\,F(F^{n-1}(k)),\; G(G^{n-1}(k))\,\big) \;\;\leq\;\; d\big(\,\eta[f \mapsto F^{n-1}(k)],\;\; \eta'[f \mapsto G^{n-1}(k)]\,\big).$$

Combining the two inequalities above gives the claim (8) for n.

Using (8), we can complete the proof of non-expansiveness. Pick $\epsilon > 0$. Then, there exists N such that for all $n > N$,

$$d(\mathsf{ufix}\, F,\; F^n(k)) \leq \epsilon/2 \quad \text{and} \quad d(G^n(k),\; \mathsf{ufix}\, G) \leq \epsilon/2.$$

By the triangular inequality, we have that

$$\begin{aligned} d(\mathsf{ufix}\, F,\; \mathsf{ufix}\, G) \;&\leq\; d(\mathsf{ufix}\, F,\; F^n(k)) + d(F^n(k), G^n(k)) + d(G^n(k),\; \mathsf{ufix}\, G) \\ &\leq\; \epsilon/2 + d(\eta,\, \eta') + \epsilon/2 \\ &=\; d(\eta,\, \eta') + \epsilon. \end{aligned}$$

Thus, $d(\mathsf{ufix}\, F,\; \mathsf{ufix}\, G) \leq d(\eta,\, \eta') + \epsilon$. Since this holds for all $\epsilon > 0$, we have the required

$$d(\mathsf{ufix}\, F,\; \mathsf{ufix}\, G) \;\;\leq\;\; d(\eta,\, \eta').$$

Next, we move on to the second part of the lemma: for all commands $\Gamma \vdash c$, their meanings $[\![\Gamma \vdash c]\!]$ are monotone functions.

The proof is again by induction on the structure of c. The monotonicity is immediate in the cases of function call and assignment. For the cases of the sequential composition and the if statement, it follows from the induction hypothesis and the monotonicity of seq and if_b. Now, it remains to show the monotonicity for the recursion case:

$$\Gamma \vdash \mathtt{fix}\, f.c$$

Consider η, η' such that $\eta \sqsubseteq \eta'$. Let F, G be functions on $\mathcal{D}$ given by

$$F(k) \;=\; \mathsf{proc}_f([\![\Gamma, f \vdash c]\!]\eta[f \mapsto k]), \qquad G(k) \;=\; \mathsf{proc}_f([\![\Gamma, f \vdash c]\!]\eta'[f \mapsto k]).$$

Define x and y to be the unique fixpoints of F and G respectively. By the induction hypothesis and the monotonicity of proc_f, we have that

$$k \sqsubseteq k' \implies \eta[f \mapsto k] \sqsubseteq \eta'[f \mapsto k'] \implies F(k) \sqsubseteq G(k'). \tag{9}$$

We need to prove that $x \sqsubseteq y$. By the Banach fixpoint theorem, x is the limit of the below Cauchy sequence:

$$y,\; F(y),\; F^2(y),\; F^3(y),\; \ldots.$$

By the requirement (1) of our framework, it suffices to show that

$$F^k(y) \ \sqsubseteq \ y.$$

We do this by induction on k. When $k = 0$, $F^k(y) = y$ so the inequality above holds. Suppose that $k > 0$. By the induction hypothesis on k, we have that $F^{k-1}(y) \sqsubseteq y$. Thus, by (9), this implies the required inequality:

$$F^k(y) \ = \ F(F^{k-1}(y)) \ \sqsubseteq \ G(y) \ = \ y$$

where the last equality uses the fact that y is the fixpoint of G. □

A.2 Missing Proofs in the Abstract Semantics Section

Lemma 3. For all $\Gamma \vdash c$ and $\eta^\sharp \in [\![\Gamma]\!]^\sharp$, if $\eta^\sharp(f) \in \mathcal{A}_t$ for every $f \in \Gamma$, we have that $[\![\Gamma \vdash c]\!]^\sharp \eta^\sharp \in \mathcal{A}_t$.

Proof. We prove the lemma by induction on the structure of c. Suppose we have a procedure environment $\eta^\sharp$ that map procedure names to elements in $\mathcal{A}_t$. We will consider each case of c separately and prove that $[\![\Gamma \vdash c]\!]^\sharp \eta^\sharp \in \mathcal{A}_t$.

- Case $c \equiv f()$. By assumption, $\eta^\sharp(f) \in \mathcal{A}_t$. Thus, $[\![c]\!]^\sharp \eta^\sharp \in \mathcal{A}_t$.
- Case $c \equiv x{:=}e$. Our framework requires that $\mathsf{asgn}_{x,e}$ be in $\mathcal{A}_t$. The lemma follows from this requirement.
- Case $c \equiv c_1; c_2$. By the induction hypothesis, both $[\![c_1]\!]^\sharp \eta^\sharp$ and $[\![c_2]\!]^\sharp \eta^\sharp$ is in $\mathcal{A}_t$. Furthermore, our framework requires that $\mathsf{seq}^\sharp$ map pairs of total elements to total elements. Hence,

$$[\![c]\!]^\sharp \eta^\sharp \ = \ \mathsf{seq}^\sharp([\![c_1]\!]^\sharp \eta^\sharp,\ [\![c_2]\!]^\sharp \eta^\sharp) \ \in \ \mathcal{A}_t.$$

- Case $c \equiv \texttt{if}\ b\ c_1\ c_2$. This case is similar to the previous one. The desired conclusion follows from the induction hypothesis and the requirement on $\mathsf{if}_b^\sharp$ with respect to total elements.
- Case $c \equiv \texttt{fix}\, f.c_1$. In this case, $[\![c]\!]^\sharp \eta^\sharp$ is always total, even when some component of $\eta^\sharp$ is not total. This is because of the $\lceil - \rceil$ operator in the semantics of $[\![\texttt{fix}\, f.c_1]\!]^\sharp$, whose range contains only total elements.

□

Theorem 2. The abstract semantics is sound. That is, (2) in Sec. 3.3 holds for all $\Gamma \vdash c$.

Proof. Our proof is by induction on the structure of c.

- Case $c \equiv f()$. $[\![f()]\!]\gamma(\eta^\sharp) \ = \ \gamma(\eta^\sharp)(f) \ = \ \gamma(\eta^\sharp(f)) \ = \ \gamma([\![f()]\!]^\sharp \eta^\sharp)$.
- Case $c \equiv x{:=}e$. This case follows from the requirement of our framework that $\mathsf{asgn}^\sharp_{x,e}$ should overapproximate $\mathsf{asgn}_{x,e}$.

– Case $c \equiv c_1; c_2$. This case follows from three ingredients – the induction hypothesis, the monotonicity of seq and the requirement that $\mathsf{seq}^\sharp$ should overapproximate seq. The below derivation shows how these ingredients give the desired conclusion.

$$\begin{aligned}
[\![c_1; c_2]\!]\gamma(\eta^\sharp) &= \mathsf{seq}([\![c_1]\!]\gamma(\eta^\sharp),\ [\![c_2]\!]\gamma(\eta^\sharp)) \\
&\sqsubseteq \mathsf{seq}(\gamma([\![c_1]\!]^\sharp\eta^\sharp),\ \gamma([\![c_2]\!]^\sharp\eta^\sharp)) \quad \text{(by ind. hypo and mono. of } \mathsf{seq}) \\
&\sqsubseteq \gamma(\mathsf{seq}^\sharp([\![c_1]\!]^\sharp\eta^\sharp,\ [\![c_2]\!]^\sharp\eta^\sharp)) \quad \text{(since } \mathsf{seq}^\sharp \text{ overapproximates } \mathsf{seq}) \\
&= \gamma([\![c_1; c_2]\!]^\sharp\eta^\sharp).
\end{aligned}$$

– Case $c \equiv \mathtt{if}\, b\, c_1\, c_2$. This case is very similar to the above. It follows from the induction hypothesis, the monotonicity of if_b, and the overapproximation property of $\mathsf{if}^\sharp_b$, as shown below.

$$\begin{aligned}
[\![\mathtt{if}\ b\ c_1\ c_2]\!]\gamma(\eta^\sharp) &= \mathsf{if}_b([\![c_1]\!]\gamma(\eta^\sharp),\ [\![c_2]\!]\gamma(\eta^\sharp)) \\
&\sqsubseteq \mathsf{if}_b(\gamma([\![c_1]\!]^\sharp\eta^\sharp),\ \gamma([\![c_2]\!]^\sharp\eta^\sharp)) \ \text{(by ind. hypo and mono. of } \mathsf{if}_b) \\
&\sqsubseteq \gamma(\mathsf{if}^\sharp_b([\![c_1]\!]^\sharp\eta^\sharp,\ [\![c_2]\!]^\sharp\eta^\sharp)) \quad \text{(since } \mathsf{if}^\sharp_b \text{ overapproximates } \mathsf{if}_b) \\
&= \gamma([\![\mathtt{if}\ b\ c_1\ c_2]\!]^\sharp\eta^\sharp).
\end{aligned}$$

– Case $c \equiv \mathtt{fix}\, f.c_1$. Let

$$\begin{aligned}
F(x) &= \mathsf{proc}_f([\![\Gamma, f \vdash c_1]\!]\gamma(\eta^\sharp)[f \mapsto x]), \\
G(A) &= \mathsf{proc}^\sharp{}_f([\![\Gamma, f \vdash c_1]\!]^\sharp\eta^\sharp[f \mapsto A]).
\end{aligned}$$

We need to prove that

$$(\mathsf{ufix}\ F) \ \sqsubseteq\ \gamma(\lceil \mathsf{widenfix}\ G \rceil). \tag{10}$$

If $\mathsf{checktot}(\mathsf{widenfix}\ G) \neq \mathsf{true}$, then $\gamma(\lceil \mathsf{widenfix}\ G \rceil) = \gamma(\top) = \top$. Thus, (10) holds. Suppose that $\mathsf{checktot}(\mathsf{widenfix}\ G) = \mathsf{true}$, which implies that $\mathsf{widenfix}\ G \in \mathcal{A}_t$. In this case, it is sufficient to prove that $\gamma(\mathsf{widenfix}\ G)$ is a post-fixpoint of F. Because then, the inequality (10) follows from Lemma 1. By the definition of $\mathsf{widenfix}$,

$$(\mathsf{widenfix}\ G) \ = \ (\mathsf{widenfix}\ G) \triangledown G(\mathsf{widenfix}\ G).$$

Because of the condition on $\triangledown$, this implies that

$$\gamma(G(\mathsf{widenfix}\ G)) \ \sqsubseteq\ \gamma(\mathsf{widenfix}\ G).$$

The LHS of this inequality is greater than or equal to $F(\gamma(\mathsf{widenfix}\ G))$ as shown below:

$$\begin{aligned}
\gamma(G(\mathsf{widenfix}\ G)) &= \gamma\big(\mathsf{proc}^\sharp{}_f\ [\![\Gamma, f \vdash c_1]\!]^\sharp\eta^\sharp[f \mapsto (\mathsf{widenfix}\ G)]\big) \\
&\sqsupseteq \mathsf{proc}_f\big(\gamma\big([\![\Gamma, f \vdash c_1]\!]^\sharp\eta^\sharp[f \mapsto (\mathsf{widenfix}\ G)]\big)\big) \\
&\sqsupseteq \mathsf{proc}_f\big([\![\Gamma, f \vdash c_1]\!]\gamma(\eta^\sharp)[f \mapsto \gamma(\mathsf{widenfix}\ G)]\big) \\
&= F(\gamma(\mathsf{widenfix}\ G)).
\end{aligned}$$

The first inequality holds because $\mathsf{proc}^\sharp$ overapproximates proc. The second inequality follows from the induction hypothesis and the monotonicity of proc_f. This shows $F(\gamma(\mathsf{widenfix}\ G)) \sqsubseteq \gamma(\mathsf{widenfix}\ G)$, as desired.

□

A.3 Missing Proof in the Generic Analysis Section

Theorem 3. Let η_* be the unique concrete environment for the empty context $\Gamma = \emptyset$. Then, for all commands c with no free procedures, if $\textsc{LivAnalysis}(c) = \mathsf{true}$, we have that $[\![c]\!]\eta_* \in \textsc{LivProperty}$.

Proof. Consider a command c that do not contain free procedure names. Suppose that

$$\textsc{LivAnalysis}(c) = \mathsf{true}.$$

Then, by Theorem 2,

$$[\![c]\!]\eta_* \;=\; [\![c]\!]\gamma(\eta_*^\sharp) \;\sqsubseteq\; \gamma([\![c]\!]^\sharp\eta_*^\sharp). \tag{11}$$

In the first equality, we use the fact that $\gamma(\eta_*^\sharp) = \eta_*$. Furthermore, $\textsc{satisfyLiv}^\sharp$ is a sound checker for the membership of $\textsc{LivProperty}$, and $\textsc{LivAnalysis}(c) = \mathsf{true}$. Hence, we also have that

$$\gamma([\![c]\!]^\sharp\eta_*^\sharp) \;\in\; \textsc{LivProperty}. \tag{12}$$

From (11), (12) and the downward closure of $\textsc{LivProperty}$, it follows that $[\![c]\!]\eta_*$ is in $\textsc{LivProperty}$, as desired. $\square$

B Missing Proofs in the Instance Section

B.1 Missing Proofs in the Concrete Semantics Section

In order to prove Lemma 4, we prove properties of the set $\mathsf{preTrace}$ with the metric d in Sec. 4.1.

Lemma 11. $(\mathsf{preTrace}, d)$ *is a metric space.*

Proof. The symmetry of d is immediate from the definition. Also,

$$d(\tau, \tau') = 0 \iff (\max\{n \mid \tau[n] = \tau'[n]\}) = \infty \iff \tau = \tau'.$$

Thus, to show that d is a metric on $\mathsf{preTrace}$, it remains to prove the triangular inequality. Consider τ, τ', τ'' in $\mathsf{preTrace}$. We will prove a stronger-than-required property that

$$d(\tau, \tau') \;\leq\; \max\bigl(d(\tau, \tau''), d(\tau'', \tau')\bigr).$$

Equivalently,

$$\max\{n \mid \tau[n] = \tau'[n]\} \;\;\geq\;\; \min\bigl(\max\{n \mid \tau[n] = \tau''[n]\},\; \max\{n \mid \tau''[n] = \tau'[n]\}\bigr).$$

Let m be the minimum on the RHS of the above inequality. Then, $\tau[m] = \tau''[m]$ and $\tau''[m] = \tau'[m]$. Thus, $\tau[m] = \tau'[m]$. This means that the LHS of the above inequality should be greater than or equal to m. $\square$

Next, we will prove that the metric space $(\mathsf{preTrace}, d)$ is complete. As a preparation for this proof, we notice that our definition of the distance d allows simpler characterization of Cauchy sequence. Recall that $\mathbb{N}$ is the set of positive integers.

Lemma 12. *A sequence* $\{\tau_i\}_{i\in\mathbb{N}}$ *in* $\mathsf{preTrace}$ *is Cauchy if and only if*

$$\forall m \in \mathbb{N}.\quad \exists n \in \mathbb{N}.\quad \forall n' \geq n.\quad \tau_{n'}[m] = \tau_n[m].$$

Proof. Recall that by the definition of Cauchy sequence a sequence $\{\tau_i\}_{i\in\mathbb{N}}$ in $\mathsf{preTrace}$ is Cauchy if and only if

$$\forall \epsilon > 0.\quad \exists n \in \mathbb{N}.\quad \forall n' \geq n.\quad d(\tau_{n'}, \tau_n) \ \leq\ \epsilon. \tag{13}$$

Restricting ϵ in (13) to those of the form 2^{-m} for some $m \in \mathbb{N}$ preserves the meaning. Thus, (13) is equivalent to

$$\forall m \in \mathbb{N}.\quad \exists n \in \mathbb{N}.\quad \forall n' \geq n.\quad d(\tau_{n'}, \tau_n) \ \leq\ 2^{-m}. \tag{14}$$

But by the definition of the distance d, we have that

$$\begin{aligned} d(\tau_{n'}, \tau_n) \ \leq\ 2^{-m} \quad &\iff \quad \max(\{m' \mid \tau_{n'}[m'] = \tau_n[m']\}) \ \geq\ m \\ &\iff \quad \tau_{n'}[m] = \tau_n[m]. \end{aligned}$$

Thus, (14) is equivalent to

$$\forall m \in \mathbb{N}.\quad \exists n \in \mathbb{N}.\quad \forall n' \geq n.\quad \tau_{n'}[m] = \tau_n[m].$$

This gives the claimed equivalence in this lemma. □

Lemma 13. $(\mathsf{preTrace}, d)$ *is complete.*

Proof. Consider a Cauchy sequence $\{\tau_n\}_{n\in\mathbb{N}}$. By the definition of Cauchy sequence, we have that

$$\forall m \in \mathbb{N}.\ \exists n_m \in \mathbb{N}.\ \forall n \geq n_m.\ \tau_{n_m}[m] = \tau_n[m]. \tag{15}$$

For each m, define

$$m^* = \max(\{n_{m'} \mid m' \leq m\} \cup \{m\})$$

where $n_{m'}$ is the index in (15). Using this notation, we define a sequence τ_∞ as follows:

$$\mathsf{proj}(\tau_\infty, m) \ =\ \begin{cases} \mathsf{proj}(\tau_{m^*}, m) & \text{if } \mathsf{proj}(\tau_{m^*}, m) \text{ is defined} \\ \text{undefined} & \text{otherwise} \end{cases}$$

where $\mathsf{proj}(\tau_\infty, m)$ means the m-th element of τ_∞. Note that since every τ_n has length at least 1 and it starts with a state in nState, $\mathsf{proj}(\tau_\infty, 1)$ is defined and it is a state in nState. To show that τ_∞ is the limit, it is sufficient to prove that

$$\forall m \in \mathbb{N}.\ \exists n_m \in \mathbb{N}.\ \forall n \geq n_m.\ \tau_n[m] = \tau_\infty[m]. \tag{16}$$

Before proving this, we note that it implies that τ_∞ is a pre-trace, i.e., $\tau_\infty \in \mathsf{preTrace}$. If τ_∞ is infinite, τ_∞ belongs to $\mathsf{nState}(\mathsf{tState}^\infty) \subseteq \mathsf{preTrace}$, because τ_∞ starts with a state in nState. If τ_∞ is finite, (16) implies that $\tau_\infty = \tau_n$ for some $n \in \mathbb{N}$, so $\tau_\infty \in \mathsf{preTrace}$.

Now, let's go back to our task of proving (16). Pick m. We claim that m^* is the witness n_m of the existential quantification in (16). To prove our claim, consider $n \geq m^*$. We need to show that

$$\forall k \in \mathbb{N}.\ 1 \leq k \leq m \implies \mathsf{proj}(\tau_n, k) = \mathsf{proj}(\tau_\infty, k) \tag{17}$$

where the equality should be interpreted as both undefined or both defined and equal. (In the rest of the proof, we use the same interpretation of equality.) By the definition of m^*, if $1 \leq k \leq m$, then $k^* \leq m^*$, so $k^* \leq n$. This implies that

$$\mathsf{proj}(\tau_{k^*}, k) \ = \ \mathsf{proj}(\tau_n, k).$$

But by definition, $\mathsf{proj}(\tau_\infty, k) = \mathsf{proj}(\tau_{k^*}, k)$. From this, the desired (17) follows. □

Lemma 4. (Trace, d) is a complete metric space.

Proof. Notice that (Trace, d) is a metric space, because Trace is a subset of $\mathsf{preTrace}$, it inherits the metric from $\mathsf{preTrace}$ and $\mathsf{preTrace}$ is a metric space.

It remains to prove the completeness of (Trace, d). Consider a Cauchy sequence $\{\tau_n\}_{n\in\mathbb{N}}$ in Trace. Let τ_∞ be the limit of this sequence in $\mathsf{preTrace}$, which exists because of Lemma 13. It remains to prove that τ_∞ belongs to Trace. By the definition of metric d and Lemma 12, we have that

$$\forall m \in \mathbb{N}.\ \exists n_m \in \mathbb{N}.\ \forall n \geq n_m.\ \ \tau_\infty[m] = \tau_n[m]. \tag{18}$$

Thus, if τ_∞ is finite, it has to be the same as some τ_n. So, it has to be in Trace, as desired. Otherwise, τ_∞ is infinite. In this case, (18) implies that all prefixes of τ_∞ are also prefixes of some traces. But, prefixes of traces always belong to $\mathcal{O}$, by the definition of traces. Thus, all prefixes of τ_∞ are in $\mathcal{O}$. This implies that τ_∞ is a trace. □

We move on to the proof of Lemma 4:

Lemma 4. $(\mathcal{D}, d^\dagger)$ is a complete metric space. Furthermore, the requirement (1) of our framework in Sec. 3 holds for $\subseteq$ and this metric space.

This lemma claims several properties of $(\mathcal{D}, d^\dagger)$. We prove them separately in the following series of lemmas. Our starting point is a slightly simpler characterization of Cauchy sequences in $(\mathcal{D}, d)$, which we will use in the following proofs:

Lemma 14. *A sequence $\{T_n\}_{n\in\mathbb{N}}$ in $\mathcal{D}$ is Cauchy if and only if*

$$\forall m \in \mathbb{N}.\ \ \exists n \in \mathbb{N}.\ \ \forall n' \geq n.\ \ T_{n'}[m] = T_n[m].$$

Proof. The proof is almost identical to that of Lemma 12, except that we replace τ_n, $\tau_{n'}$ and their m prefix projections by T_n, $T_{n'}$ and the m prefix projections of T_n and $T_{n'}$. □

Lemma 15. *$(\mathcal{D}, d^\dagger)$ is a metric space.*

Proof. The symmetry of $d^\dagger$ is immediate from the definition. Next, we show that

$$d^\dagger(T, T') = 0 \iff T = T'.$$

By the definition of $d^\dagger$, $d^\dagger(T, T) = 0$ for all $T \in \mathcal{D}$. The right-to-left direction of the equivalence follows from this. For the other direction, suppose that $d^\dagger(T, T') = 0$. Pick $\tau \in T$. Then, for all $n \in \mathbb{N}$,

$$\tau[n] \;\in\; T[n] \;=\; T'[n].$$

This implies that

$$\forall n \in \mathbb{N}.\ \exists \tau'_n \in T'.\ (\tau'_n)[n] = \tau[n].$$

Thus, $\{\tau'_n\}_{n\in\mathbb{N}}$ is a Cauchy sequence with τ as its limit. Since T' is closed and the sequence $\{\tau'_n\}_{n\in\mathbb{N}}$ is in T', the limit τ should be in T' as well. We have just shown that $T \subseteq T'$. The other inclusion can be proved similarly.

Finally, we prove that d satisfies the triangular inequality. In fact, we prove a stronger property that for all $T, T', T'' \in \mathcal{D}$,

$$d^\dagger(T, T') \;\le\; \max(d^\dagger(T, T''),\ d^\dagger(T'', T')),$$

which is equivalent to

$$\max\{n \mid T[n] = T'[n]\} \;\ge\; \min\bigl(\max\{n \mid T[n] = T''[n]\},\ \max\{n \mid T''[n] = T'[n]\}\bigr).$$

Let m be the value of the RHS of the above inequality. Then, $T[m] = T''[m]$ and $T''[m] = T'[m]$. Thus, $T[m] = T'[m]$. This means that the LHS of the above inequality should be at least m. □

Lemma 16. *For all Cauchy sequences $\{T_n\}_{n\in\mathbb{N}}$ in $\mathcal{D}$ and indices $m, k \in \mathbb{N}$, if*

$$\forall k' \ge k.\ \ T_k[m] = T_{k'}[m],$$

then for all $\tau \in T_k$, there exists a Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$ in Trace *such that*

1. *$\tau[m] = \tau_i[m]$ for all $i \in \mathbb{N}$, and*
2. *the sequence is taken from an infinite subsequence of $\{T_n\}_{n\in\mathbb{N}}$, i.e.,*

$$\exists \{k_i\}_{i\in\mathbb{N}}.\ \ (\forall i \in \mathbb{N}.\ \tau_i \in T_{k_i}) \ \wedge\ (\forall i, j \in \mathbb{N}.\ i < j \implies k_i < k_j).$$

Proof. Let $\{T_n\}_{n\in\mathbb{N}}$ and m, k be the ones satisfying the conditions of the lemma. Pick τ from T_k. Using these data, we will construct two desired sequences—the Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$ and the sequence $\{k_i\}_{i\in\mathbb{N}}$ of indices. Note that since $\{T_n\}_{n\in\mathbb{N}}$ is Cauchy,

$$\forall o \in \mathbb{N}.\ \ \exists n_o \in \mathbb{N}.\ \ \forall n \ge n_o.\ \ T_{n_o}[o] = T_n[o].$$

Using n_o's, we define the desired sequence of indices by

$$k_1 \;=\; n_m \quad \text{and} \quad k_{i+1} \;=\; \max(n_{(m+i)},\; k_i + 1\,).$$

Note that this sequence is strictly increasing. It remains to construct the other sequence $\{\tau_i\}_{i\in\mathbb{N}}$ of traces. We do this inductively. By the assumption on T_k and the choice of n_m, we must have that

$$T_k[m] \;=\; T_{\max(k,n_m)}[m] \;=\; T_{(n_m)}[m] \;=\; T_{(k_1)}[m].$$

Hence,

$$\tau[m] \;=\; \tau'[m] \quad \text{for some } \tau' \in T_{(k_1)}.$$

We define the first element of the sequence by

$$\tau_1 \;=\; \tau'.$$

For the rest, we assume that τ_i is chosen from $T_{(k_i)}$, and we inductively pick trace τ_{i+1} from $T_{(k_{i+1})}$ as follows. Since $k_{i+1} > k_i \geq n_{(m+i-1)}$, we have that

$$T_{(k_{i+1})}[m+i-1] \;=\; T_{n_{(m+i-1)}}[m+i-1] \;=\; T_{(k_i)}[m+i-1].$$

Furthermore, τ_i is in $T_{(k_i)}$. Thus, there must exist $\tau'' \in T_{(k_{i+1})}$ such that

$$\tau_i[m+i-1] \;=\; \tau''[m+i-1].$$

We define τ_{i+1} to be this τ''.

By construction, it is immediate that $\{\tau_i\}_{i\in\mathbb{N}}$ is from the infinite subsequence $\{T_{(k_i)}\}_{i\in\mathbb{N}}$ of $\{T_n\}_{n\in\mathbb{N}}$. Furthermore, $\{\tau_i\}_{i\in\mathbb{N}}$ is Cauchy, because

$$\forall n \in \mathbb{N}.\;\; \forall i \geq n.\;\; \tau_i[m+i-1] = \tau_{i+1}[m+i-1],$$

and so,

$$\forall n \in \mathbb{N}.\;\; \forall i \geq n.\;\; \tau_n[n] = \tau_i[n].$$

(Remember here that $m \in \mathbb{N}$ and so $m \geq 1$.) Finally, by construction, $\tau_1[m] = \tau_i[m]$ for all $i \in \mathbb{N}$. But $\tau_1[m] = \tau'[m] = \tau[m]$. Thus, $\tau_i[m] = \tau[m]$ for all $i \in \mathbb{N}$, as desired. □

Proposition 1. *$(\mathcal{D}, d^\dagger)$ is complete.*

Proof. Consider a Cauchy sequence $\{T_n\}_{n\in\mathbb{N}}$ in $\mathcal{D}$. Define T_∞ as follows:

$$\begin{aligned} T_\infty \;=\; \{\, \lim_{i\to\infty} \tau_i \;\mid\; & \{\tau_i\}_{i\in\mathbb{N}} \text{ is Cauchy } \wedge \\ & \exists\{k_i\}_{i\in\mathbb{N}}.\, (\forall i \in \mathbb{N}.\, \tau_i \in T_{k_i}) \;\wedge\; (\forall i, j \in \mathbb{N}.\, i < j \implies k_i < k_j)\,\}. \end{aligned}$$

Firstly, we show that T_∞ is closed. Consider a Cauchy sequence $\{\alpha_n\}_{n\in\mathbb{N}}$ in T_∞. Let α_∞ be the limit of this sequence. To prove the closedness, we need to show that α_∞ belongs to T_∞. Equivalently, we need to find a Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$ such that

1. the limit of the sequence is α_∞, and
2. the sequence is taken from an infinite subsequence of $\{T_n\}_{n\in\mathbb{N}}$, i.e., it satisfies that

$$\exists\{k_i \in \mathbb{N}\}_{i\in\mathbb{N}}.\ (\forall i \in \mathbb{N}.\ \tau_i \in T_{k_i})\ \wedge\ (\forall i, j \in \mathbb{N}.\ i < j \implies k_i < k_j).$$

Since $\{\alpha_n\}_{n\in\mathbb{N}}$ converges to α_∞, we have that

$$\forall m \in \mathbb{N}.\ \exists n_m \in \mathbb{N}.\ \forall n' \geq n_m.\ \alpha_{n'}[m] = \alpha_\infty[m].$$

For each $m \in \mathbb{N}$, we let

$$m^* \ = \ \max(\{n_{m'} \mid m' \leq m\} \cup \{m\}).$$

The maximum is used to ensure that $-^*$ is monotone with respect to $\leq$. Since α_n is in T_∞, the definition of T_∞ implies the existence of a Cauchy sequence $\{\tau_i^n\}_{i\in\mathbb{N}}$ such that the limit of the sequence is α_n and the sequence satisfies that

$$\exists\{k_i^n \in \mathbb{N}\}_{i\in\mathbb{N}}.\ (\forall i \in \mathbb{N}.\ \tau_i^n \in T_{(k_i^n)})\ \wedge\ (\forall i, j \in \mathbb{N}.\ i < j \implies k_i^n < k_j^n).$$

Thus,

$$\forall n \in \mathbb{N}.\ \forall m \in \mathbb{N}.\ \exists i_{n,m} \in \mathbb{N}.\ \forall i' \geq i_{n,m}.\ \tau_{i'}^n[m] = \alpha_n[m].$$

For each $m \in \mathbb{N}$, define

$$m^\dagger \ = \ i_{m^*,m}$$

Also, construct an increasing sequence $\{j_i\}_{i\in\mathbb{N}}$ of natural numbers by

$$j_1 \ = \ 1^\dagger \quad \text{and} \quad j_{i+1} \ = \ \min\{\, j' \mid k_{(j')}^{(i+1)^*} > k_{(j_i)}^{(i^*)} \ \wedge\ j' \geq (i+1)^\dagger \,\}.$$

Using these $-^*$ and $\{j_i\}_{i\in\mathbb{N}}$, we construct the required $\{\tau_i\}_{i\in\mathbb{N}}$ as follows:

$$\tau_i \ = \ \tau_{(j_i)}^{(i^*)}$$

Then, for all $m \in \mathbb{N}$ and all $i \geq m$,

$$\tau_i[i] \ = \ \tau_{(j_i)}^{(i^*)}[i] \ = \ \alpha_{(i^*)}[i] \ = \ \alpha_\infty[i].$$

The first equality is just the unrolling of the definition of τ_i. The second equality holds, because $j_i \geq i^\dagger$ and so $\tau_{(j_i)}^{(i^*)}[i] = \alpha_{(i^*)}[i]$. The third equality follows from the definition of i^*. We have just shown that $\tau_i[i] = \alpha_\infty[i]$, and since $i \geq m$, this implies

$$\tau_i[m] = \alpha_\infty[m].$$

Thus, $\{\tau_i\}_{i\in\mathbb{N}}$ is a Cauchy sequence that converges to α_∞. Furthermore, this sequence is taken from an infinite subsequence of $\{T_n\}_{n\in\mathbb{N}}$. Concretely, the indices of this infinite subsequence are

$$k_{(j_i)}^{(i^*)} \quad \text{for all } i \in \mathbb{N}.$$

By the choice of j_i, the index sequence is strictly increasing: if $i < l$, then $k^{(i^*)}_{(j_i)} < k^{(l^*)}_{(j_l)}$.

Secondly, we prove that T_∞ is full. Note that this implies that $T_\infty \in \mathcal{D}$. Choose a *none*-tagged state $\sigma \in \mathsf{nState}$. It is sufficient to construct a Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$ such that

1. for all $i \in \mathbb{N}$, $\tau_i[1]$ is the singleton trace σ, and
2. there exists $\{k_i\}_{i\in\mathbb{N}}$ satisfying that

$$\forall i, j \in \mathbb{N}.\ \ \tau_i \in T_{(k_i)}\ \ \wedge\ \ (i < j \implies k_i < k_j).$$

We will construct the desired sequence $\{\tau_i\}_{i\in\mathbb{N}}$ using Lemma 16. Note that since $\{T_n\}_{n\in\mathbb{N}}$ is Cauchy,

$$\exists n_1 \in \mathbb{N}.\ \ \forall n \geq n_1.\ \ T_n[1] = T_{(n_1)}[1].$$

Since all T_n's are full, there must be a trace τ in $T_{(n_1)}$ whose starting state is σ. Now, Lemma 16 implies the existence of a Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$ such that

1. $\tau_i[1] = \tau[1]$ for all $i \in \mathbb{N}$, and
2. there exists $\{k_i\}_{i\in\mathbb{N}}$ satisfying

$$\forall i, j \in \mathbb{N}.\ \ \tau_i \in T_{(k_i)}\ \wedge\ (i < j \implies k_i < k_j).$$

But, $\tau_i[1] = \tau[1]$ means that $\tau_i[1]$ is the singleton trace σ. Thus, $\{\tau_i\}_{i\in\mathbb{N}}$ is the sequence that we are looking for.

Finally, we prove that T is the limit of $\{T_n\}_{n\in\mathbb{N}}$. Pick $m \in \mathbb{N}$. We need to find $n_m \in \mathbb{N}$ such that

$$\forall n \geq n_m.\ \ T_n[m]\ =\ T_\infty[m].$$

Since $\{T_n\}_{n\in\mathbb{N}}$ is Cauchy, there exists $k \geq 1$ such that

$$\forall n \geq k.\ \ T_n[m]\ =\ T_k[m].$$

We claim that k is the desired n_m. Let n be an index such that $n \geq k$. To show the inclusion

$$T_n[m]\ \supseteq\ T_\infty[m],$$

pick τ from $T_\infty[m]$. This means that $\tau = \tau'[m]$ for some $\tau' \in T_\infty$. Then, by the definition of T_∞, there must be a Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$, that is taken from an infinite subsequence $\{T_{(k_i)}\}_{i\in\mathbb{N}}$, and that converges to τ'. Thus,

$$\exists j \in \mathbb{N}.\ \ \tau_j \in T_{(k_j)}\ \wedge\ \tau_j[m] = \tau'[m]\ \wedge\ (k_j \geq n).$$

Since $T_n[m] = T_k[m] = T_{(k_j)}[m]$, there exists $\tau'' \in T_n$ such that

$$\tau''[m]\ \ =\ \ \tau_j[m]\ \ =\ \ \tau'[m]\ \ =\ \ \tau.$$

Thus, $\tau \in T_n[m]$. It remains to show the other inclusion

$$T_n[m]\ \subseteq\ T_\infty[m].$$

Pick τ from $T_n[m]$. This means that $\tau = \tau'[m]$ for some $\tau' \in T_n$. By Lemma 16, there exists a Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$ such that

1. $\tau_i[m] = \tau'[m]$ for all $i \in \mathbb{N}$, and
2. the sequence is taken from an infinite subsequence of $\{T_n\}_{n\in\mathbb{N}}$.

By definition, the limit τ_∞ of $\{\tau_i\}_{i\in\mathbb{N}}$ should belong to T_∞. Furthermore, since the first m-prefixes of τ_i's equal $\tau'[m]$, we should also have that $\tau_\infty[m] = \tau'[m]$. Since $\tau'[m] = \tau$, it follows that $\tau \in T_\infty[m]$, as desired. □

Lemma 17. *The condition* (1) *of our framework on the distance and the preorder in Sec. 3 holds for* $(\mathcal{D}, d^\dagger, \subseteq, \mathsf{Trace})$.

Proof. Let $T \in \mathcal{D}$ and consider a Cauchy sequence $\{T_n\}_{n\in\mathbb{N}}$ in $\mathcal{D}$ such that $T_n \subseteq T$ for all n. Also, let T_∞ be the limit of this sequence. We need to prove that $T_\infty \subseteq T$. Pick τ from T_∞. Since T_∞ is the limit of $\{T_n\}_{n\in\mathbb{N}}$, we have that

$$\forall m \in \mathbb{N}.\quad \exists n \in \mathbb{N}.\quad T_n[m] \;=\; T_\infty[m].$$

Hence, for all $m \in \mathbb{N}$, there exist n_m and $\tau_{(n_m)} \in T_{(n_m)}$ such that

$$\tau_{(n_m)}[m] \;=\; \tau[m].$$

Because $T_{(n_m)}$ is a subset of T, $\tau_{(n_m)}$ belongs to T as well. Furthermore, $\{\tau_{(n_i)}\}_{i\in\mathbb{N}}$ is a Cauchy sequence converging to τ. This is because for all $m \in \mathbb{N}$ and $k \geq m$, $\tau_{(n_k)}[k] = \tau[k]$, so

$$\tau_{(n_k)}[m] \;=\; \tau[m].$$

Now, the closedness of T implies that the limit τ of the Cauchy sequence $\{\tau_{(n_i)}\}_{i\in\mathbb{N}}$ in T should belong to T as well. Since τ is chosen arbitrary, this membership of τ to T means that $T_\infty \subseteq T$, as desired. □

The remaining lemma to prove in Sec. 4.1 is Lemma 6:

Lemma 6. All the operators are well-defined, and satisfy the monotonicity and non-expansiveness or $\frac{1}{2}$-contractiveness requirements of our framework.

This lemma claims several facts on the semantic operators. We prove each one separately.

Lemma 18. $\mathsf{seq}(T, T')$ *consists of traces.*

Proof. Pick τ from $\mathsf{seq}(T, T')$. It is immediate from the definition of seq that τ is a pre-trace. We will show that τ satisfies the additional condition for traces as well. If τ belongs to $T \cap \mathsf{tState}^\infty$, it should be in T as well. This implies that τ is a trace. Suppose that $\tau \notin (T \cap \mathsf{tState}^\infty)$. Then, there exist τ_0, σ_0, τ_1 such that

$$(\tau_0\sigma_0 \in (T \cap \mathsf{tState}^+)) \;\;\wedge\;\; (\sigma_0\tau_1 \in T') \;\;\wedge\;\; \tau = \tau_0\sigma_0\tau_1.$$

Since $\sigma_0\tau_1$ is a pre-trace, σ_0 should be tagged with *none*. Furthermore, since $\tau_0\sigma_0$ is a finite trace, it has to be in $\mathcal{W}$. We now do the case analysis depending on whether $\sigma_0\tau_1$ is finite. If $\sigma_0\tau_1$ is finite, $\sigma_0\tau_1$ has to be in $\mathcal{W}$. Hence, $\tau = \tau_0\sigma_0\tau_1$ is a finite sequence belonging to $\mathcal{W}$. From this, it follows that τ is a trace. If $\sigma_0\tau_1$ is infinite, all prefixes of $\sigma_0\tau_1$ belong to $\mathcal{O}$. Thus, all prefixes of $\tau = \tau_0\sigma_0\tau_1$ also belong to $\mathcal{O}$. This implies that τ is a trace. □

Lemma 19. *For all $T, T' \in \mathcal{D}$, $\mathsf{seq}(T, T')$ is in $\mathcal{D}$, i.e., it is closed and full.*

Proof. Let T, T' be trace sets in $\mathcal{D}$. Firstly, we prove that $\mathsf{seq}(T, T')$ is full. Pick a *none*-tagged state $\sigma \in \mathsf{nState}$. Since T is full, there is a trace τ in T that starts with σ. If τ is infinite, it also belongs to $\mathsf{seq}(T, T')$, so we have just found a trace in $\mathsf{seq}(T, T')$ starting with σ. If τ is finite, it must be of the form $\tau_0\sigma_0$ for some *none*-tagged state $\sigma_0 \in \mathsf{nState}$. This is because τ is a finite pre-trace, so it should start and end with *none*-tagged states. But, T' is full. Hence, $\sigma_0\tau' \in T'$ for some sequence τ' of tagged states. Now, the definition of $\mathsf{seq}(T, T')$ implies that

$$\tau_0\sigma_0\tau' \;\in\; \mathsf{seq}(T, T').$$

Since $\tau_0\sigma_0\tau'$ starts with σ, it is the trace that we are looking for.

Secondly, we show that $\mathsf{seq}(T, T')$ is closed. Consider a Cauchy sequence $\{\tau_n\}_{n\in\mathbb{N}}$ in $\mathsf{seq}(T, T')$. Let τ_∞ be the limit of this sequence. We need to show that $\tau_\infty \in \mathsf{seq}(T, T')$. There are two cases to consider.

The first case is that there is an infinite subsequence $\{\tau_{(n_i)}\}_{i\in\mathbb{N}}$ of $\{\tau_n\}_{n\in\mathbb{N}}$ such that

$$\forall i \in \mathbb{N}.\;\; \tau_{(n_i)} \;\in\; (T \cap \mathsf{tState}^\infty).$$

Since the original sequence $\{\tau_n\}_{n\in\mathbb{N}}$ is Cauchy, the subsequence $\{\tau_{(n_i)}\}_{i\in\mathbb{N}}$ is Cauchy as well. Furthermore, the two sequences have the same limit τ_∞. This limit has to be an infinite sequence, because every member of $\{\tau_{(n_i)}\}_{i\in\mathbb{N}}$ is infinite. It also belongs to T, since T is closed. Hence,

$$\tau_\infty \;\in\; (T \cap \mathsf{tState}^\infty) \;\subseteq\; \mathsf{seq}(T, T').$$

The second case is that all elements of $\{\tau_n\}_{n\in\mathbb{N}}$ except finitely many are from

$$\{\, \tau\sigma\tau' \mid (\tau\sigma \in T \cap \mathsf{tState}^+) \;\wedge\; (\sigma\tau' \in T') \,\}.$$

This means that there is some $n_0 \in \mathbb{N}$ such that

$$\forall n \geq n_0.\;\; \exists \tau_n^0, \sigma_n, \tau_n^1.\;\; (\tau_n = \tau_n^0\sigma_n\tau_n^1) \;\wedge\; (\tau_n^0\sigma_n \in T \cap \mathsf{tState}^+) \;\wedge\; (\sigma_n\tau_n^1 \in T').$$

We sub-divide this case based on whether there is some $u \in \mathbb{N}$ with

$$\forall n \geq n_0.\;\; |\tau_n^0\sigma_n| \;\leq\; u. \tag{19}$$

Suppose that there exists such an upper bound u. Since $\{\tau_n\}_{n\in\mathbb{N}}$ is Cauchy, this implies that there is some $n_1 \geq n_0$ such that

$$\forall n \geq n_1.\;\; (\tau_n^0\sigma_n = \tau_{n_1}^0\sigma_{n_1}).$$

In this sub-case, $\{\sigma_n\tau_n^1\}_{n\geq n_1}$ is also Cauchy, its limit τ'_∞ starts with σ_n, and it satisfies the below relationship with the limit τ_∞ of $\{\tau_n\}_{n\in\mathbb{N}}$:

$$\tau_{n_1}^0\tau'_\infty \;\;=\;\; \tau_\infty.$$

Note that the sequence $\{\sigma_n\tau^1_n\}_{n\geq n_1}$ is in T', which is a closed set. Thus, τ'_∞ is in T'. Because τ'_∞ starts with σ_{n_1} and $\tau_{n_1}\sigma_{n_1}$ is in T, we have that

$$\tau_\infty \;=\; \tau_{n_1}\tau'_\infty \;\in\; \mathsf{seq}(T,T').$$

The other sub-case is that there does not exist u satisfying (19). In this sub-case, $\{\tau^0_n\sigma_n\}_{n\geq n_0}$ becomes a Cauchy sequence in T with τ_∞ as its limit. Since T is closed, τ_∞ is in T. Also, $|\tau^0_n\sigma_n|$ goes to the infinity as n increases, so τ_∞ belongs to $T\cap \mathsf{tState}^\infty$. Hence, τ_∞ is in $\mathsf{seq}(T,T')$, as desired. $\square$

Lemma 20. *The function* seq *is non-expansive.*

Proof. By the definition of the distance $d^\dagger$ on $\mathcal{D}$, proving the non-expansive of seq is equivalent to showing that for all T_0,T'_0,T_1,T'_1 in $\mathcal{D}$ and all $m\in\mathbb{N}$,

$$(T_0[m] \;=\; T_1[m] \;\wedge\; T'_0[m] \;=\; T'_1[m]) \;\Longrightarrow\; (\mathsf{seq}(T_0,T'_0)[m] \;=\; \mathsf{seq}(T_1,T'_1)[m]).$$

Let T_0,T'_0,T_1,T'_1,m be the data in the above equivalent statement, and assume the condition of the implication. We need to show that $\mathsf{seq}(T_0,T'_0)[m] = \mathsf{seq}(T_1,T'_1)[m]$. We will prove that $\mathsf{seq}(T_0,T'_0)[m] \subseteq \mathsf{seq}(T_1,T'_1)[m]$. The other subset inclusion can be proved similarly. Pick $\tau\in\mathsf{seq}(T_0,T'_0)[m]$. This means that

$$\exists\tau'\in\mathsf{seq}(T_0,T'_0).\;\; \tau'[m] \;=\; \tau.$$

Since $\tau'\in\mathsf{seq}(T_0,T'_0)$, we have

$$(\tau'\in T_0\cap\mathsf{tState}^\infty) \;\vee\; (\exists\tau_0,\sigma,\tau'_0.\;\; \tau'=\tau_0\sigma\tau'_0 \;\wedge\; \tau_0\sigma\in T_0 \;\wedge\; \sigma\tau'_0\in T'_0). \quad (20)$$

Suppose that the first disjunct holds. Since $T_0[m]=T_1[m]$, there is $\tau''\in T_1$ such that

$$|\tau''|\geq m \quad\wedge\quad \tau''[m] \;=\; \tau'[m] \;=\; \tau.$$

If τ'' is infinite, it is also in $\mathsf{seq}(T_1,T'_1)$. So, $\tau=\tau''[m]\in\mathsf{seq}(T_1,T'_1)[m]$ as desired. Consider the other case that τ'' is finite. In this case, we note two facts. Firstly, since τ'' is a trace, it should end with a *none*-tagged state, say, $\sigma\in\mathsf{nState}$. Secondly, since T_1 is full, there is $\sigma\tau'''\in T'_1$ for some sequence τ''' of tagged states. Hence, $\tau''\tau'''$ is in $\mathsf{seq}(T_1,T'_1)$. But $|\tau''|\geq m$, which means that

$$(\tau''\tau''')[m] \;=\; \tau''[m] \;=\; \tau.$$

So, τ is in $\mathsf{seq}(T_1,T'_1)$.

Now, suppose that the second disjunct of (20) holds. Let τ_0,σ,τ'_0 be the witnesses of the existential quantification in (20). Since $m\geq 1$, $T_0[m]=T_1[m]$ and $T'_0[m]=T'_1[m]$,

$$\exists\tau_1,\tau'_1.\;\; \tau_1\in T_1 \;\wedge\; (\sigma\tau'_1)\in T'_1 \;\wedge\; (\tau_0\sigma)[m]=\tau_1[m] \;\wedge\; (\sigma\tau'_0)[m]=(\sigma\tau'_1)[m].$$

Let m_0 be $|\tau_0\sigma|$. If $m_0\geq m$, we can ignore τ'_1, and complete the proof similarly as in the previous case, just doing the case-analysis on whether τ_1 is infinite or not. Suppose that $m_0<m$. In this case,

$$\tau_1 \;=\; \tau_0\sigma \quad\wedge\quad \tau'_0[m-m_0] \;=\; \tau'_1[m-m_0].$$

Thus,

$$\tau \;=\; \tau'[m] \;=\; (\tau_0\sigma)(\tau_0'[m-m_0]) \;=\; (\tau_1)(\tau_1'[m-m_0]) \;=\; (\tau_1\tau_1')[m].$$

Since $\tau_1\tau_1' \in \mathsf{seq}(T, T')$, this means that $\tau \in \mathsf{seq}(T, T')[m]$. □

Lemma 21. *For all $T \in \mathcal{D}$, if all traces in T have length at least 2 (i.e., $\forall \tau \in T.\, |\tau| \geq 2$), the specialization $\mathsf{seq}(T, -)$ by T is 1/2-contractive on $\mathcal{D}$, i.e.,*

$$\forall T_1, T_2 \in \mathcal{D}.\;\; d(\mathsf{seq}(T, T_1),\, \mathsf{seq}(T, T_2)) \;\;\leq\;\; (1/2 \times d(T_1, T_2)).$$

Proof. Let T be a trace set satisfying the condition in the lemma. Pick T_1, T_2 from $\mathcal{D}$. We need to prove that:

$$d^{\dagger}(\mathsf{seq}(T, T_1), \mathsf{seq}(T, T_2)) \;\leq\; (\frac{1}{2} \times d^{\dagger}(T_1, T_2)). \tag{21}$$

Let A and B be trace sets defined by

$$\begin{aligned} A \;&=\; \{\, \tau\sigma\tau' \mid (\tau\sigma \in T \cap \mathsf{tState}^{+}) \;\wedge\; (\sigma\tau' \in T_1)\,\}, \\ B \;&=\; \{\tau\sigma\tau' \mid (\tau\sigma \in T \cap \mathsf{tState}^{+}) \;\wedge\; (\sigma\tau' \in T_2)\,\}. \end{aligned}$$

Then, by the definition of seq,

$$\mathsf{seq}(T, T_1) \;=\; A \cup (T \cap \mathsf{tState}^{\infty}) \quad\text{ and }\quad \mathsf{seq}(T, T_2) \;=\; B \cup (T \cap \mathsf{tState}^{\infty}).$$

Thus, to prove (21), it is sufficient to show that for all $m \in \mathbb{N}$,

$$T_1[m] = T_2[m] \implies \big((A \cup (T \cap \mathsf{tState}^{\infty}))[m+1] = (B \cup (T \cap \mathsf{tState}^{\infty}))[m+1]\big). \tag{22}$$

Suppose that $T_1[m] = T_2[m]$. We will show that $A[m+1] = B[m+1]$. From this, the equality in the conclusion of the implication (22) follows, because the "$-[m+1]$" operator distributes over $\cup$.

Here we will show only one inclusion $A[m+1] \subseteq B[m+1]$; the other inclusion can be shown similarly. Suppose that we have a trace $\tau \in A[m+1]$. This means that there is a trace $\tau' \in A$ such that

$$\tau \;\;=\;\; (\tau'[m+1]).$$

By the definition of A,

$$\exists \tau_0, \sigma_0, \tau_1.\;\; (\tau' = \tau_0\sigma_0\tau_1) \;\wedge\; (\tau_0\sigma_0 \in T \cap \mathsf{tState}^{+}) \;\wedge\; (\sigma_0\tau_1 \in T_1). \tag{23}$$

Since $T_1[m] = T_2[m]$ by assumption (and $m \geq 1$), we also have that

$$\exists \tau_2.\;\; (\sigma_0\tau_2 \;\in\; T_2) \;\wedge\; (\sigma_0\tau_2[m] \;=\; \sigma_0\tau_1[m]). \tag{24}$$

Note that $\tau_0\sigma_0\tau_2 \in B$ by the definition of B, (23) and (24). Since $\tau = (\tau'[m+1])$ and $\tau' = \tau_0\sigma_0\tau_1$, we can show the desired $\tau \in B[m+1]$, if we prove that

$$(\tau_0\sigma_0\tau_1)[m+1] \;\;=\;\; (\tau_0\sigma_0\tau_2)[m+1]. \tag{25}$$

Now, recall that T contains traces of length at least 2, so $|\tau_0\sigma_0| \;\;\geq\;\; 2$. This means that the second conjunct of (24) implies (25). □

Lemma 22. *The operator* seq *is* $\subseteq$*-monotone*

Proof. In the definition of $\mathsf{seq}(T, T')$, the argument trace sets T and T' appear only in positive positions, i.e., they do not occur under the left of implication or under negation. The monotonicity follows from this. □

Lemma 23. *For all* $T_0, T_1 \in \mathcal{D}$*, the trace set* $\mathsf{if}_b(T_0, T_1)$ *is in* $\mathcal{D}$*, i.e., it is closed and full.*

Proof. Let $T = \mathsf{if}_b(T_0, T_1)$. We consider the closedness property of T first. Consider a Cauchy sequence $\{\tau_i\}_{i\in\mathbb{N}}$ in T. By the definition of Cauchy sequence, there exists an index n in $\mathbb{N}$ such that

$$\forall i \geq n. \quad \tau_i[1] = \tau_n[1].$$

Let σ be $\tau_n[1]$. If $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$, all the elements of $\{\tau_i\}_{i\in\mathbb{N}}$ except the first $n-1$ belong to T_0 and have the same σ as their starting state. This implies that the limit τ_∞ of the sequence belongs to T_0 and it has σ as its starting state. But $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$ by assumption. Thus, τ_∞ is also in T. The other case that $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{false}$ is similar.

Next, we prove that T is full. Pick a *none*-tagged state $\sigma \in \mathsf{nState}$. We consider the case that $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{false}$ only, because the other case $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$ can be proved similarly. Since T_1 is full, there is a trace of the form $\sigma\tau \in T_1$. Since $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{false}$, the trace $\sigma\tau$ is also included in T. We have just found a trace in T that starts with σ. □

Lemma 24. *The function* if_b *is non-expansive.*

Proof. By the definition of distance $d^\dagger$ on $\mathcal{D}$, proving the non-expansiveness is equivalent to showing that for all (T_0, T_1) and (T_0', T_1') in $\mathcal{D} \times \mathcal{D}$ and all $m \in \mathbb{N}$,

$$(T_0[m] \;=\; T_0'[m] \;\wedge\; T_1[m] \;=\; T_1'[m]) \implies (\mathsf{if}_b(T_0, T_1)[m] \;=\; \mathsf{if}_b(T_0', T_1')[m]).$$

Pick $\sigma\tau$ from $\mathsf{if}_b(T_0, T_1)[m]$. Since $m \geq 1$, this means that there exists $\sigma\tau'$ in $\mathsf{if}_b(T_0, T_1)$ such that

$$\sigma\tau \;=\; (\sigma\tau')[m].$$

If $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$, the trace $\sigma\tau'$ is from T_0. Since $T_0[m] = T_0'[m]$ (and $m \geq 1$), there is $\sigma\tau'' \in T_0'$ such that

$$(\sigma\tau'')[m] \;=\; (\sigma\tau')[m] \;=\; \sigma\tau.$$

Furthermore, since $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$, this trace $\sigma\tau''$ should be in $\mathsf{if}_b(T_0', T_1')$ as well. Putting all these together, we can conclude that

$$\sigma\tau \;=\; (\sigma\tau'')[m] \;\in\; \mathsf{if}_b(T_0'.T_1').$$

The case that $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{false}$ can be proved similarly. Hence,

$$\mathsf{if}_b(T_0, T_1)[m] \;\subseteq\; \mathsf{if}_b(T_0', T_1')[m]$$

Using a similar argument, we can prove the other inclusion. □

Lemma 25. *The operator* if_b *is monotone with respect to the subset order* $\subseteq$.

Proof. In the definition of $\mathsf{if}_b(T_0, T_1)$, the argument trace sets T_0 and T_1 are used only positively. From this, the monotonicity of the lemma follows. $\square$

Lemma 26. $\mathsf{proc}_f(T)$ *consists of traces only.*

Proof. Note that the definition of $\mathsf{proc}_f(T)$ is given by the union of three sets. The first two sets there consist of finite pre-traces, because all traces in T start and end with *none*-tagged states and so, σ, σ_1 in the description of the first two sets have *none* as their tags. Furthermore, they are subsets of $\mathcal{W}$, because all finite traces in T belong to $\mathcal{W}$. Hence, the sets contain traces only. For the third set in the definition of $\mathsf{proc}_f(T)$, we note that all pre-traces there are infinite, because again traces in T start with *none*-tagged states and so σ in the definition of the third set should be tagged with *none*. Furthermore, all prefixes of infinite traces in T belong to $\mathcal{O}$, so that all prefixes of traces in the third set should be in $\mathcal{O}$ as well. From these observation, it follows that $\mathsf{proc}_f(T)$ consists of traces. $\square$

Lemma 27. *For all procedures* $f \in \mathsf{PName}$ *and* $T \in \mathcal{D}$, $\mathsf{proc}_f(T)$ *belongs to* $\mathcal{D}$. *That is, it contains traces only, and it is full and closed. Furthermore, for all* $f \in \mathsf{PName}$, *function* $\mathsf{proc}_f(-)$ *is* $1/2$-*contractive and monotone.*

Proof. Pick $f \in \mathsf{PName}$ and $T \in \mathcal{D}$. Firstly, we prove that $\mathsf{proc}_f(T)$ is full and closed. Let

$$\begin{aligned}\mathsf{prolog}_f &= \{\, \sigma\sigma^{(f,call)}\sigma \mid \sigma \in \mathsf{nState} \,\},\\ \mathsf{epilog}_f &= \{\, \sigma\sigma^{(f,ret)}\sigma \mid \sigma \in \mathsf{nState} \,\}.\end{aligned}$$

Note that these sets consist of pre-traces, not traces. But,

$$\mathsf{proc}_f(T) = \mathsf{seq}(\mathsf{seq}(\mathsf{prolog}_f, T), \mathsf{epilog}_f)$$

when we use the definition of seq for sets of pre-traces as well. Furthermore, the proof of Lemma 19 does not rely on the fact that its arguments contain traces only, so it also works when we change the lemma such that the arguments of seq are full closed sets of *pre-traces*. Hence, to show that $\mathsf{proc}_f(T)$ is full and closed, it is sufficient to prove that prolog_f and epilog_f are full and closed. Note that all sequences in prolog_f or epilog_f have length 3. So, every Cauchy sequence in the sets converges to the n-th element in the sequence for some $n \in \mathbb{N}$, which means that prolog_f and epilog_f are closed. The remaining condition that prolog_f and epilog_f are full is an immediate consequence of their definitions.

Secondly, we prove that $\mathsf{proc}_f(-)$ is $\frac{1}{2}$-contractive and monotone. Recall that $\mathsf{proc}_f(T)$ is $\mathsf{seq}(\mathsf{seq}(\mathsf{prolog}_f, T), \mathsf{epilog}_f)$ for all T. We again rely on the observation that the proofs of Lemmas 20 and 21 can be generalized to full closed sets of pre-traces. Both proofs are independent of the fact that the arguments of seq consist of traces. They work equally well, when we change the lemmas such that seq takes full closed sets of *pre-traces* as its parameters. Hence, the generalization of Lemma 20 and 21 implies the $1/2$-contractiveness of $\mathsf{proc}_f(-)$, because every pre-trace in prolog_f has length greater than 2. The remaining monotonicity condition is immediate from the definition of $\mathsf{proc}_f(-)$. $\square$

Lemma 28. *For all assignments $x{:=}e$, $\mathsf{asgn}_{x,e}$ is in $\mathcal{D}$.*

Proof. From the definitions of $\mathsf{asgn}_{x,e}$, it is immediate that $\mathsf{asgn}_{x,e}$ consists of traces and that they are full. For the closedness, we note that $\mathsf{asgn}_{x,e}$ contains traces of size 2. Thus, every Cauchy sequence $\{\tau_n\}_{n\in\mathbb{N}}$ in this set should contain some τ_i that is the limit of the sequence. From this property of Cauchy sequence, the closedness follows. □

Lemma 29. LivProperty *is downward closed wrt. the $\subseteq$ order.*

Proof. This follows from the fact that $T \in$ LivProperty is defined in terms of a universal requirement on traces in T. □

B.2 Missing Proofs in the Abstract Semantics Section

Lemma 7. For every $A \in \mathcal{A}_t$, the set $\gamma(A)$ is in $\mathcal{D}$, i.e., it is full and closed.

Proof. Consider $A = (\varphi_1, \varphi_2, \varphi_3)$ in $\mathcal{A}_t$. Let T be $\gamma(\varphi_1, \varphi_2, \varphi_3)$. Firstly, we prove that T is full. Since $(\varphi_1, \varphi_2, \varphi_3) \in \mathcal{A}_t$, its first component φ_1 should be total:

$$\forall \text{`}s \in (\text{`}\mathsf{Var} \to \mathbb{Q}).\ \exists s \in (\mathsf{Var} \to \mathbb{Q}).\ \ (\text{`}s, s)\ \models\ \varphi_1.$$

This implies that for every *none*-tagged state σ_0, there exists a *none*-tagged state σ_1 such that

$$(\text{`}\mathsf{first}(\sigma_0), \mathsf{first}(\sigma_1))\ \models\ \varphi_1.$$

Furthermore, when $\sigma_0\sigma_1$ is viewed as a trace, it does not contain any procedure calls, so it satisfies the requirements imposed by φ_2 and φ_3. Hence, $\sigma_0\sigma_1$ is in T. Note that σ_0 is chosen arbitrarily. Hence, we have just shown that T is full.

Next, we show that T is closed. Let $\{\tau_n\}_{n\in\mathbb{N}}$ be a Cauchy sequence in T, and let τ_∞ be the limit of the sequence. We will prove that τ_∞ is in T. If τ_∞ is finite, it has to be the same as some τ_n in the sequence. Thus, τ_∞ is in T. Suppose that τ_∞ is infinite. We have to prove that τ_∞ satisfies the two requirements imposed by φ_2 and φ_3. To discharge the requirement from φ_2, pick $\sigma \in \tau_\infty$ such that the tag of σ is a procedure call. Let m be the position of σ in the trace τ_∞. Then, since τ_∞ is the limit of $\{\tau_n\}_{n\in\mathbb{N}}$, there exists τ_n such that $\tau_n[m] = \tau_\infty[m]$. But, τ_n is in T, and so,

$$(\text{`}\mathsf{first}(\mathsf{first}(\tau_n)), \mathsf{first}(\sigma))\ \models\ \varphi_2.$$

The LHS of $\models$ is the same as $(\text{`}\mathsf{first}(\mathsf{first}(\tau_\infty)), \mathsf{first}(\sigma))$. Hence, the requirement from φ_2 holds for τ_∞. Now, it remains to prove that τ_∞ satisfies the requirement from φ_3. Pick σ_1, σ_2 such that $\mathsf{open}(\sigma_1, \sigma_2, \tau_\infty)$. Let m be the position of σ_2 in the trace τ_∞. Again, we use the fact that τ_∞ is the limit of $\{\tau_n\}_{n\in\mathbb{N}}$, so there exists τ_n such that $\tau_n[m] = \tau_\infty[m]$, which implies that $\mathsf{open}(\sigma_1, \sigma_2, \tau_n)$. Thus,

$$(\text{`}\mathsf{first}(\sigma_1), \mathsf{first}(\sigma_2))\ \models\ \varphi_3.$$

This completes the proof that τ_∞ satisfies the condition for φ_3. □

Next, we prove Lemma 8:

Lemma 8. The operators in Fig. 6 meet all the requirements of our framework.

This lemma claims several properties of abstract operators. We prove them separately.

Lemma 30. *If both A and A' are in $\mathcal{A}_t$, their sequential composition* $\mathsf{seq}^\sharp(A, A')$ *is in $\mathcal{A}_t$ as well.*

Proof. Suppose that $A = (\varphi_1, \varphi_2, \varphi_3)$ and $A' = (\psi_1, \psi_2, \psi_3)$ are in $\mathcal{A}_t$. This means that both φ_1 and ψ_1 belong to TForm. Then, $\varphi_1; \psi_1$ is also in TForm, because the $-;-$ operator for formulas means the composition of state relations and the composition of two total relations is total. Hence,

$$\mathsf{seq}^\sharp\big((\varphi_1, \varphi_2, \varphi_3),\ (\psi_1, \psi_2, \psi_3)\big) \ = \ \big(\varphi_1; \psi_1,\ \ \varphi_2 \vee (\varphi_1; \psi_2),\ \ \varphi_3 \vee \psi_3\big)$$

belongs to $\mathcal{A}_t$ as well. □

Lemma 31. *For all A and A' in $\mathcal{A}_t$, we have that*

$$\mathsf{seq}(\gamma(A), \gamma(A')) \ \subseteq \ \gamma(\mathsf{seq}^\sharp(A, A')).$$

Proof. Let $(\varphi_1, \varphi_2, \varphi_3) = A$ and $(\psi_1, \psi_2, \psi_3) = B$. Also, let $T_0 = \gamma(\varphi_1, \varphi_2, \varphi_3)$ and $T_1 = \gamma(\psi_1, \psi_2, \psi_3)$. Pick a trace τ from $\mathsf{seq}(T_0, T_1)$. By the definition of $\mathsf{seq}^\sharp$,

$$\mathsf{seq}^\sharp\big((\varphi_1, \varphi_2, \varphi_3),\ (\psi_1, \psi_2, \psi_3)\big) \ = \ \big(\varphi_1; \psi_1,\ \varphi_2 \vee (\varphi_1; \psi_2),\ \varphi_3 \vee \psi_3\big).$$

Hence, it suffices to prove that τ satisfies the three requirements in the definition of γ, which are determined by $(\varphi_1; \psi_1)$, $\varphi_2 \vee (\varphi_1; \psi_2)$, and $(\varphi_3 \vee \psi_3)$. To do this, we do the case analysis on τ.

1. The first case is that $\tau = \tau_0 \sigma \tau_1$ for some *finite* trace $\tau_0 \sigma$ in T_0 and a trace $\sigma \tau_1$ in T_1. Let's start with the first requirement given by $\varphi_1; \psi_1$:

$$\tau \in \mathsf{tState}^+ \ \implies \ (\text{`}\mathsf{first}(\mathsf{first}(\tau)),\ \mathsf{first}(\mathsf{last}(\tau))) \ \models \ (\varphi_1; \psi_1), \tag{26}$$

   Note that when τ is infinite, the requirement holds vacuously. Suppose that τ is finite. In this case, the suffix $\sigma\tau_1$ is finite as well. Since $\tau_0\sigma \in T_0$, $\sigma\tau_1 \in T_1$ and both traces are finite, these two traces satisfy the below condition imposed by φ_1 and ψ_1 in the definition of γ:

$$(\text{`}\mathsf{first}(\mathsf{first}(\tau_0\sigma)),\ \mathsf{first}(\sigma)) \ \models \ \varphi_1 \quad \wedge \quad (\text{`}\mathsf{first}(\sigma),\ \mathsf{first}(\mathsf{last}(\sigma\tau_1))) \ \models \ \psi_1.$$

   This implies that

$$(\text{`}\mathsf{first}(\mathsf{first}(\tau_0\sigma)),\ \mathsf{first}(\mathsf{last}(\sigma\tau_1))) \ \models \ (\varphi_1; \psi_1), \tag{27}$$

   because the $-;-$ operator for formulas models relational composition correctly. But $\mathsf{first}(\tau_0\sigma) = \mathsf{first}(\tau)$ and $\mathsf{last}(\tau) = \mathsf{last}(\sigma\tau_1)$. Thus, (26) follows from (27).

Next, we prove the second requirement given by $\varphi_2 \vee (\varphi_1; \psi_2)$:

$$\forall \sigma_0 \in \tau.\ \mathsf{iscall}(\sigma_0) \implies (\text{`}\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\sigma_0)) \models (\varphi_2 \vee (\varphi_1; \psi_2)). \quad (28)$$

Note that σ_0 appears in the prefix $\tau_0\sigma$ or in the suffix $\sigma\tau_1$. If the former holds,

$$(\text{`}\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\sigma_0)) = (\text{`}\mathsf{first}(\mathsf{first}(\tau_0\sigma)), \mathsf{first}(\sigma_0)) \models \varphi_2.$$

Hence, (28) holds. Now, suppose that σ_0 appears in the suffix $\sigma\tau_1$. Since $\sigma\tau_1$ satisfies the second requirement on ψ_2,

$$(\text{`}\mathsf{first}(\mathsf{first}(\sigma\tau_1)), \mathsf{first}(\sigma_0)) \models \varphi_2. \quad (29)$$

Furthermore, since $\tau_0\sigma$ satisfies the requirement from φ_1,

$$(\text{`}\mathsf{first}(\mathsf{first}(\tau_0\sigma)), \mathsf{first}(\mathsf{last}(\tau_0\sigma))) \models \varphi_1. \quad (30)$$

The desired (28) follows from (29) and (30), because the sequential composition $\varphi_1; \psi_2$ precisely means the relational composition.
Finally, we show the third requirement $\varphi_3 \vee \psi_3$. Pick σ_1 and σ_2 such that $\mathsf{open}(\sigma_1, \sigma_2, \tau)$. We should show that

$$(\text{`}\mathsf{first}(\sigma_1),\ \mathsf{first}(\sigma_2)) \models (\varphi_3 \vee \psi_3). \quad (31)$$

Note that $\tau = \tau_0\sigma\tau_1$ for $\tau_0\sigma$ in T_0 and $\sigma\tau_1$ in T_1. Furthermore, since $\tau_0\sigma$ is finite, if a state in $\tau_0\sigma$ is tagged with a procedure call, the corresponding return should appear in $\tau_0\sigma$ as well, because this is one of the conditions in the definition of the concrete semantic domain $\mathcal{D}$. Hence, either the states σ_1 and σ_2 tagged with procedure calls appear in $\tau_0\sigma$, or they both appear in $\sigma\tau_1$. In the first case,

$$(\text{`}\mathsf{first}(\sigma_1),\ \mathsf{first}(\sigma_2)) \models \varphi_3, \quad (32)$$

because $\tau_0\sigma$ satisfies the requirement regarding two call states. Similarly, in the second case, we have that

$$(\text{`}\mathsf{first}(\sigma_1),\ \mathsf{first}(\sigma_2)) \models \psi_3. \quad (33)$$

The desired (31) follows from (32) and (33).

2. The second case is that τ is an infinite trace from T_0. In this case, the first requirement given by $(\varphi_1; \psi_1)$ holds vacuously. The other two requirements also hold for a simple reason. Since the trace τ is in T_0, it satisfies the second and third requirements regarding φ_2 and φ_3. But, we have that

$$\varphi_2 \models \varphi_2 \vee (\varphi_1; \psi_2)$$
and
$$\varphi_3 \models \varphi_3 \vee \psi_3.$$

The second and third requirements are monotone with respect to formulas. Thus, τ also satisfies the two requirements given by weaker formulas $\varphi_2 \vee (\varphi_1; \psi_2)$ and $\varphi_3 \vee \psi_3$.

□

Lemma 32. *If both A and A' are in $\mathcal{A}_t$, the conditional statement $\mathsf{if}^\sharp_b(A, A')$ is in $\mathcal{A}_t$ as well.*

Proof. Suppose that both $A = (\varphi_1, \varphi_2, \varphi_3)$ and $A' = (\psi_1, \psi_2, \psi_3)$ are in $\mathcal{A}_t$. This means that $\varphi_1, \psi_1 \in \mathsf{TForm}$. To prove this lemma, it suffices to show that

$$(b_1 \wedge \varphi_1) \vee (b_2 \wedge \psi_1) \in \mathsf{Form}, \tag{34}$$

where b_1 and b_2 are defined as in the lemma. Note that b_2 is equivalent to $\neg b_1$. Thus, for all ${}^\backprime s$ in ${}^\backprime\mathsf{Var} \to \mathbb{Q}$, ${}^\backprime s$ satisfies b_1 or b_2. In the first case, the totality of φ_1 (i.e., $\varphi_1 \in \mathsf{TForm}$) implies that

$$\exists s \in (\mathsf{Var} \to \mathbb{Q}).\ ({}^\backprime s, s) \models (b_1 \wedge \varphi_1) \vee (b_2 \wedge \psi_1).$$

In the second case, the same conclusion follows from the totality of ψ_1. Since ${}^\backprime s$ is chosen arbitrarily, we have just shown the required (34). □

Lemma 33. *For all A and A' in $\mathcal{A}_t$, we have that*

$$\mathsf{if}_b(\gamma(A), \gamma(A')) \subseteq \gamma(\mathsf{if}^\sharp_b(A, A')).$$

Proof. Pick $A = (\varphi_1, \varphi_2, \varphi_3)$ and $A' = (\psi_1, \psi_2, \psi_3)$ from $\mathcal{A}_t$. Let

$$b_1 = \mathsf{preprime}(b), \quad b_2 = \mathsf{preprime}(\mathsf{neg}(b)).$$

We will show that every $\tau \in \mathsf{if}_b(\gamma(A), \gamma(A'))$ belongs to $\gamma(\mathsf{if}^\sharp_b(A, A'))$. Choose an arbitrary τ from $\mathsf{if}_b(\gamma(A), \gamma(A'))$, and let σ be $\mathsf{first}(\tau)$. Then, $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$, or $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{false}$. In the proof, we will consider the former case only, since the latter can be proved similarly. Suppose that $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$. Then, for all $s \in (\mathsf{Var} \to \mathbb{Q})$,

$$({}^\backprime\mathsf{first}(\sigma), s) \models b_1. \tag{35}$$

Furthermore, by the definition of if_b and the assumption that $[\![b]\!](\mathsf{first}(\sigma)) = \mathsf{true}$, the trace τ should be in $\gamma(A)$, which means that

$$\begin{aligned}
&(\tau \in \mathsf{tState}^+ \implies ({}^\backprime\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\mathsf{last}(\tau))) \models \varphi_1) \quad \text{and} \\
&(\forall \sigma \in \tau.\, \mathsf{iscall}(\sigma) \implies ({}^\backprime\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\sigma)) \models \varphi_2) \quad \text{and} \\
&(\forall \sigma_1, \sigma_2.\, \mathsf{open}(\sigma_1, \sigma_2, \tau) \implies ({}^\backprime\mathsf{first}(\sigma_1), \mathsf{first}(\sigma_2)) \models \varphi_3).
\end{aligned} \tag{36}$$

From (35) and (36), the below property follows:

$$\begin{aligned}
&(\tau \in \mathsf{tState}^+ \implies ({}^\backprime\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\mathsf{last}(\tau))) \models (b_1 \wedge \varphi_1) \vee (b_2 \wedge \psi_1)) \quad \text{and} \\
&(\forall \sigma \in \tau.\, \mathsf{iscall}(\sigma) \implies ({}^\backprime\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\sigma)) \models (b_1 \wedge \varphi_2) \vee (b_2 \wedge \psi_2)) \quad \text{and} \\
&(\forall \sigma_1, \sigma_2.\, \mathsf{open}(\sigma_1, \sigma_2, \tau) \implies ({}^\backprime\mathsf{first}(\sigma_1), \mathsf{first}(\sigma_2)) \models \varphi_3 \vee \psi_3).
\end{aligned}$$

Thus, τ is in $\gamma(\mathsf{if}^\sharp_b(A, A'))$, as desired. □

Lemma 34. *If A is in $\mathcal{A}_t$, so is $\mathsf{proc}^\sharp{}_f(A)$.*

Proof. Suppose that $A = (\varphi_1, \varphi_2, \varphi_3)$ is in $\mathcal{A}_t$. This means that φ_1 is total, i.e., it belongs to TForm. Since the first component of $\mathsf{proc}^\sharp{}_f(\varphi_1, \varphi_2, \varphi_3)$ is again φ_1, the totality of φ_1 implies that $\mathsf{proc}^\sharp{}_f(\varphi_1, \varphi_2, \varphi_3)$ is in $\mathcal{A}_t$, as desired. $\square$

Lemma 35. *For all $f \in \mathsf{PName}$ and all A in $\mathcal{A}_t$, we have that*

$$\mathsf{proc}_f(\gamma(A)) \subseteq \gamma(\mathsf{proc}^\sharp{}_f(A)).$$

Proof. Pick $A = (\varphi_1, \varphi_2, \varphi_3) \in \mathcal{A}_t$. Consider a trace τ in $\mathsf{proc}_f(\gamma(\varphi_1, \varphi_2, \varphi_3))$. We need to prove that

$$\tau \;\in\; \gamma\big(\mathsf{proc}^\sharp{}_f(\varphi_1, \varphi_2, \varphi_3)\big) \;=\; \gamma(\varphi_1,\ eq_{\mathsf{Var}} \vee \varphi_2,\ \varphi_2 \vee \varphi_3).$$

Recall that proc_f is defined to be the disjunction of three cases. In all three cases, there is some trace $\tau_1 \in \gamma(\varphi_1, \varphi_2, \varphi_3)$ such that

1. $\mathsf{first}(\tau_1) = \mathsf{first}(\tau)$, and
2. if τ is finite, so is τ_1 and $\mathsf{last}(\tau) = \mathsf{last}(\tau_1)$.

Furthermore, τ_1 satisfies the first requirement in the definition of $\gamma(\varphi_1, \varphi_2, \varphi_3)$, which is given by φ_1. So, τ satisfies the first requirement of $\gamma(\mathsf{proc}^\sharp{}_f(\varphi_1, \varphi_2, \varphi_3))$ as well.

In the rest of the proof, we prove the remaining two requirements for

$$\gamma(\mathsf{proc}^\sharp{}_f(\varphi_1, \varphi_2, \varphi_3)).$$

Let $T = \gamma(\varphi_1, \varphi_2, \varphi_3)$. Our proof will treat the three cases in the definition of proc_f separately.

1. The first case is that $\tau = \sigma\sigma^{(f,call)}\sigma^{(f,ret)}\sigma$ for some $\sigma \in T$. In this case, the second requirement in the definition of γ is:

   $$(`\mathsf{first}(\sigma), \mathsf{first}(\sigma^{(f,call)})) \models eq_{\mathsf{Var}} \vee \varphi_2,$$

   which holds because of eq_{Var} on the RHS. For the third requirement, we note that there are no σ_1 and σ_2 in τ satisfying $\mathsf{open}(\sigma_1, \sigma_2, \tau)$. Hence, the requirement holds vacuously.
2. The second case is that $\tau = \sigma\sigma^{(f,call)}\tau_2\sigma_2^{(f,ret)}\sigma_2$ for some $\sigma\tau_2\sigma_2 \in (T \cap \mathsf{tState}^+)$. To prove the second requirement, consider $\sigma_3 \in \tau$ such that $\mathsf{iscall}(\sigma_3)$. If σ_3 is the second element in τ, it is $\sigma^{(f,call)}$, so

   $$(`\mathsf{first}(\sigma), \mathsf{first}(\sigma_3)) \models eq_{\mathsf{Var}}. \tag{37}$$

   Otherwise, $\sigma_3 \in \sigma\tau_2\sigma_2$. Since $\sigma\tau_2\sigma_2$ is in $T = \gamma(\varphi_1, \varphi_2, \varphi_3)$, it satisfies the requirement given by φ_2. This implies that

   $$(`\mathsf{first}(\sigma), \mathsf{first}(\sigma_3)) \models \varphi_2. \tag{38}$$

The satisfaction relationships (37) and (38) imply the desired property:

$$(\text{`}\mathsf{first}(\sigma), \mathsf{first}(\sigma_3)) \models eq_{\mathsf{Var}} \vee \varphi_2.$$

For the third requirement, pick σ_3, σ_4 such that $\mathsf{open}(\sigma_3, \sigma_4, \tau)$. If σ_3 is not the second element of τ, we have that

$$\mathsf{open}(\sigma_3,\ \sigma_4,\ \sigma\tau_2\sigma_2).$$

Thus, $(\text{`}\mathsf{first}(\sigma_3), \mathsf{first}(\sigma_4)) \models \varphi_3$, and the desired third requirement follows from this. Otherwise, i.e., σ_3 is the second element of τ, it is $\sigma^{(f,call)}$. Thus, $\mathsf{first}(\sigma_3) = \mathsf{first}(\sigma)$. Since σ_4 is a call and it appears in $\sigma\tau_2\sigma_2 \in T$, it has to satisfy

$$(\text{`}\mathsf{first}(\sigma), \mathsf{first}(\sigma_4)) \models \varphi_2.$$

Now this satisfaction relationship and $\mathsf{first}(\sigma_3) = \mathsf{first}(\sigma)$ imply that the desired third requirement holds.

3. The last case is that $\tau \in \sigma\sigma^{(f,call)}\tau_2$ for $\sigma\tau_2 \in (T \cap \mathsf{tState}^\infty)$. The proof of this case is almost identical to the one for the second. Simply replacing $\sigma\tau_2\sigma_2$ there by $\sigma\tau_2$ gives the proof of this third case.

□

Lemma 36. *For all assignments $x{:=}e$, $\mathsf{asgn}^\sharp_{x,e}$ is in $\mathcal{A}_t$. Furthermore, it satisfies the following soundness requirements:*

$$\mathsf{asgn}_{x,e} \subseteq \gamma(\mathsf{asgn}^\sharp_{x,e}).$$

Proof. The first claim of the lemma about the membership to $\mathcal{A}_t$ holds, because the formula $eq_{\mathsf{Var}-\{x\}} \wedge (x = e[\text{`}x/x])$ is in TForm. Next, we show that

$$\mathsf{asgn}_{x,e} \subseteq \gamma(\mathsf{asgn}^\sharp_{x,e}).$$

Pick τ from $\mathsf{asgn}_{x,e}$. By the definition of $\mathsf{asgn}_{x,e}$, the trace τ should be of the form $\sigma_1\sigma_2$ such that

1. σ_1 and σ_2 are *none*-tagged states and
2. $\mathsf{first}(\sigma_2) = \mathsf{first}(\sigma_1)[x \mapsto [\![e]\!]\mathsf{first}(\sigma_1)]$.

This characterization of τ implies that τ does not include any function calls, and also that τ's starting and ending states σ_1 and σ_2 satisfy

$$(\text{`}\mathsf{first}(\sigma_1),\ \ \mathsf{first}(\sigma_2)) \models eq_{\mathsf{Var}-\{x\}} \wedge (x = e[\text{`}x/x]).$$

From these, it follows that τ is in $\gamma(\mathsf{asgn}^\sharp_{x,e})$, as desired. □

Lemma 37. *A binary operator $\nabla : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ is a widening operator, if it turns every sequence in $\mathcal{A}$ into one with a stable element and it satisfies the condition below:*

$$\big((\varphi_1, \varphi_2, \varphi_3)\nabla(\psi_1, \psi_2, \psi_3) = (\delta_1, \delta_2, \delta_3)\big) \implies \forall i \in \{1, 2, 3\}.\ \big(\psi_i \models \delta_i\big),$$

where $\models$ is the semantic entailment.

Proof. We need to prove two properties of $\triangledown$. Firstly, it can be restricted to a map from $\mathcal{A}_t \times \mathcal{A}_t$ to $\mathcal{A}_t$. Secondly, it computes an upper bound of its arguments:

$$\forall A, A' \in \mathcal{A}_t. \quad \gamma(A') \;\subseteq\; \gamma(A \triangledown A').$$

Pick $A = (\varphi_1, \varphi_2, \varphi_3)$ and $A' = (\psi_1, \psi_2, \psi_3)$ from $\mathcal{A}_t$. Let $(\delta_1, \delta_2, \delta_3)$ be $A \triangledown A'$. Since A, A' are in $\mathcal{A}_t$, their first components φ_1 and ψ_1 define total relations (i.e., $\varphi_1, \psi_1 \in \mathsf{TForm}$). Note that δ_1 is weaker than φ_1 and ψ_1 by assumption. Hence, δ_1 also defines a total relation, so $(\delta_1, \delta_2, \delta_3)$ is in $\mathcal{A}_t$, as desired by the first property. Now, we move on to the second property: $\gamma(A') \subseteq \gamma(A \triangledown A')$. Pick a trace τ from $\gamma(A')$. By the definition of γ, we have that

$$\begin{aligned}
&(\tau \in \mathsf{tState}^+ \implies (\text{`}\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\mathsf{last}(\tau))) \models \psi_1) \quad \text{and}\\
&(\forall \sigma \in \tau.\, \mathsf{iscall}(\sigma) \implies (\text{`}\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\sigma)) \models \psi_2) \quad \text{and}\\
&(\forall \sigma_1, \sigma_2.\, \mathsf{open}(\sigma_1, \sigma_2, \tau) \implies (\text{`}\mathsf{first}(\sigma_1), \mathsf{first}(\sigma_2)) \models \psi_3).
\end{aligned}$$

By assumption, ψ_i semantically implies δ_i for all $i \in \{1, 2, 3\}$. Thus, the three conjuncts above imply

$$\begin{aligned}
&(\tau \in \mathsf{tState}^+ \implies (\text{`}\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\mathsf{last}(\tau))) \models \delta_1) \quad \text{and}\\
&(\forall \sigma \in \tau.\, \mathsf{iscall}(\sigma) \implies (\text{`}\mathsf{first}(\mathsf{first}(\tau)), \mathsf{first}(\sigma)) \models \delta_2) \quad \text{and}\\
&(\forall \sigma_1, \sigma_2.\, \mathsf{open}(\sigma_1, \sigma_2, \tau) \implies (\text{`}\mathsf{first}(\sigma_1), \mathsf{first}(\sigma_2)) \models \delta_3).
\end{aligned}$$

That is, τ belongs to $\gamma(A \triangledown A')$, as desired. □

Lemma 38. *The operator $\triangledown_k : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$ is a widening operator.*

Proof. All the subroutines and parameters used in the definition of $\triangledown_k$ overapproximate their input formulas. From this and the definition of $\triangledown_k$ above, it follows that

$$\big((\varphi_1, \varphi_2, \varphi_3) \triangledown_k (\psi_1, \psi_2, \psi_3) = (\delta_1, \delta_2, \delta_3)\big) \implies \forall i \in \{1, 2, 3\}.\, \big(\psi_i \models \delta_i\big),$$

where $\models$ is the semantic entailment. Thus, by Lemma 37, to prove this lemma, we just need to show that $\triangledown_k$ turns every sequence into one with a stable element. Note that the formula δ_i in the result of the widening is in the range of bound_k, so it cannot have more than k outermost disjuncts. Furthermore, δ_i in the result of the widening is true, or it has one more disjunct than φ_i, or it is the same as φ_i. These imply that for every sequence $\{A_n = (\delta_1^n, \delta_2^n, \delta_3^n)\}_{n \in \mathbb{N}}$ in $\mathcal{A}$, if we construct the widened sequence $\{A'_n = ((\delta')_1^n, (\delta')_2^n, (\delta')_3^n)\}_{n \in \mathbb{N}}$ by

$$A'_1 = A_1 \quad \text{and} \quad A'_{n+1} = A'_n \triangledown_k A_{n+1},$$

then for all $i \in \{1, 2, 3\}$, every disjunct in $(\delta')_i^n$ is included in $(\delta')_i^{n+1}$, unless $(\delta')_i^{n+1}$ is true. Thus, the sequence $\{(\delta')_i^n\}_{n \in \mathbb{N}}$ goes over the bound k and remains true forever, or it hits a limit element before reaching the bound k. This implies that $\{A'_n\}_{n \in \mathbb{N}}$ has a stable point. □

Finally, we prove Lemma 10

Lemma 10. For all $A \in \mathcal{A}_t$, if $\textsc{satisfyLiv}^\sharp(A) = \mathsf{true}$, we have that $\gamma(A) \in \textsc{LivProperty}$.

Proof. Consider $A \in \mathcal{A}_t$ such that $\textsc{satisfyLiv}^\sharp(A) = \mathsf{true}$. Pick a trace $\tau \in \gamma(A)$. For the sake of contradiction, suppose that τ includes an infinite subsequence of open calls. That is, there exists $\{\tau_i\sigma_i\}_{i\in\mathbb{N}}$ such that

$$(\tau \;=\; \tau_1\sigma_1\tau_2\sigma_2\ldots) \quad \wedge \quad (\forall i, j \in \mathbb{N}.\; i < j \implies \mathsf{open}(\sigma_i, \sigma_j, \tau)).$$

By the definition of γ, we should have that

$$\forall i \in \mathbb{N}.\; (\text{`}\mathsf{first}(\sigma_i), \mathsf{first}(\sigma_j)) \models \varphi_3.$$

Furthermore, the formula φ_3 is disjunctively well-founded, since $\textsc{satisfyLiv}^\sharp(A) = \mathsf{true}$. Hence, the result of Podelski and Rybalchenko in [19] implies that the sequence $\sigma_1\sigma_2\ldots$ is finite. But, this is impossible, since $\sigma_1\sigma_2\ldots$ is an infinite sequence. We have just derived the desired contradiction. $\square$

B.3 An Example of the Analysis

We illustrate the abstract interpreter with the command below:

$$C \;\equiv\; \texttt{fix}\; f.\, \Big(\, \texttt{if}\; \big(x \le 0\big)\; \big(x{:=}x\big)\; \big(x{:=}x{-}1;\; f();\; x{:=}x{+}1\big)\,\Big)$$

Note that this command is not tail recursive, but it always terminates.

To simplify presentation, we will assume that x is the only program variable. We also assume that lower returns lower bounds of the form $\text{`}x \ge 0$ only. Similarly, we assume that $\mathsf{upper}$ computes upper bounds of the form $\text{`}x \le 0$ only.

Our abstract interpreter calculates the abstract semantics of C by an iterative fixpoint computation. The first iteration of this computation works as follows. It picks the environment η_0 defined by:

$$A_0 \;=\; (\mathsf{false}, \mathsf{false}, \mathsf{false}), \qquad\qquad \eta_0 \;=\; [f \mapsto A_0].$$

Then, the abstract interpreter analyzes the true and false branches of the conditional statement in f:

$$\begin{aligned} [\![x{:=}x]\!]^\sharp\eta_0 &= (\text{`}x{=}x,\; \mathsf{false},\; \mathsf{false}), \\ [\![x{:=}x{-}1; f(); x{:=}x{+}1]\!]^\sharp\eta_0 &= (\mathsf{false},\;\; \mathsf{false},\;\; \mathsf{false}). \end{aligned}$$

Finally, it computes the abstract meaning of the body of f:

$$\begin{aligned} A_1 &= A_0 \mathbin{\triangledown} \big(\mathsf{proc}^\sharp{}_f([\![\texttt{if}\; (x \le 0)\; (x{:=}x)\; (x{:=}x{-}1; f(); x{:=}x{+}1)]\!]^\sharp\eta_0)\big) \\ &= (\mathsf{false}, \mathsf{false}, \mathsf{false}) \mathbin{\triangledown} \mathsf{proc}^\sharp{}_f(\text{`}x \le 0 \wedge \text{`}x{=}x,\;\; \mathsf{false},\;\; \mathsf{false}) \\ &= (\mathsf{false}, \mathsf{false}, \mathsf{false}) \mathbin{\triangledown} (\text{`}x \le 0 \wedge \text{`}x{=}x,\;\; \text{`}x{=}x,\;\; \mathsf{false}) \\ &= (\text{`}x \le 0 \wedge \text{`}x{=}x,\;\; \text{`}x{=}x,\;\; \mathsf{false}). \end{aligned}$$

The second fixpoint iteration proceeds similarly. It picks the environment η_1:

$$\eta_1 \quad = \quad [f \mapsto A_1].$$

Then, it computes the abstract semantics of the true and false branches:

$$\begin{aligned}
[\![x{:=}x]\!]^\sharp \eta_1 &= (\text{`}x{=}x,\ \mathsf{false},\ \mathsf{false}),\\
[\![x{:=}x{-}1; f(); x{:=}x{+}1]\!]^\sharp \eta_1 &= \big(\ (\exists a'b'.\ \text{`}x{-}1{=}a' \wedge a' \leq 0 \wedge a'{=}b' \wedge b'{+}1{=}x),\\
&\quad\ (\exists a'.\ \text{`}x{-}1{=}a' \wedge a'{=}x),\\
&\quad\ \mathsf{false}\ \big).
\end{aligned}$$

Finally, the abstract interpreter combines the above two abstract values, and finishes the second iteration:

$$\begin{aligned}
A_2 &= A_1 \triangledown \mathsf{proc}^\sharp{}_f([\![\mathtt{if}\ (x \leq 0)\ (x{:=}x)\ (x{:=}x{-}1; f(); x{:=}x{+}1)]\!]^\sharp \eta_1)\\
&= A_1 \triangledown \big(\ (\text{`}x \leq 0 \wedge \text{`}x{=}x) \vee (\text{`}x > 0 \wedge \exists a'b'.\ \text{`}x{-}1{=}a' \wedge a' \leq 0 \wedge a'{=}b' \wedge b'{+}1{=}x),\\
&\qquad\qquad (\text{`}x{=}x) \vee (\text{`}x > 0 \wedge \exists a'.\ \text{`}x{-}1{=}a' \wedge a'{=}x),\\
&\qquad\qquad (\text{`}x > 0 \wedge \exists a'.\ \text{`}x{-}1{=}a' \wedge a'{=}x)\ \big)\\
&= \big(\ (\text{`}x \leq 0 \wedge \text{`}x{=}x) \vee (\text{`}x \geq 0 \wedge \text{`}x{=}x),\\
&\qquad (\text{`}x{=}x) \vee (\text{`}x \geq 0 \wedge \text{`}x{-}1 \geq x),\\
&\qquad (\text{`}x \geq 0 \wedge \text{`}x{-}1 \geq x)\ \big).
\end{aligned}$$

The computed A_2 is the fixpoint, and becomes the result of analyzing the command C.

After the fixpoint computation, the abstract interpreter checks whether A_2 satisfies $\textsc{satisfyLiv}^\sharp$. In this case, we have that $\textsc{satisfyLiv}^\sharp(A_2) = \mathsf{true}$, because the third component of A_2 is a well-founded relation. Hence, by Lemma 10, the concrete meaning of C belongs to $\textsc{LivProperty}$, i.e., $[\![C]\!]$ does not contain an infinite subsequence of open calls. In still other words, C terminates.