An Abstract Interpretation Framework for Termination

Patrick Cousot
CNRS, École Normale Supérieure, and INRIA, France
Courant Institute [*], NYU, USA
cousot@ens.fr, pcousot@cims.nyu.edu

Radhia Cousot
CNRS, École Normale Supérieure, and INRIA, France
rcousot@ens.fr

Abstract

Proof, verification and analysis methods for termination all rely on two induction principles: (1) a variant function or induction on data ensuring progress towards the end and (2) some form of induction on the program structure.

The abstract interpretation design principle is first illustrated for the design of new forward and backward proof, verification and analysis methods for *safety*. The safety collecting semantics defining the strongest safety property of programs is first expressed in a constructive fixpoint form. Safety proof and checking/verification methods then immediately follow by fixpoint induction. Static analysis of abstract safety properties such as invariance are constructively designed by fixpoint abstraction (or approximation) to (automatically) infer safety properties. So far, no such clear design principle did exist for termination so that the existing approaches are scattered and largely not comparable with each other.

For (1), we show that this design principle applies equally well to *potential and definite termination*. The trace-based termination collecting semantics is given a fixpoint definition. Its abstraction yields a fixpoint definition of the best variant function. By further abstraction of this best variant function, we derive the Floyd/Turing termination proof method as well as new static analysis methods to effectively compute approximations of this best variant function.

For (2), we introduce a generalization of the syntactic notion of structural induction (as found in Hoare logic) into a *semantic structural induction* based on the new semantic concept of *inductive trace cover* covering execution traces by *segments*, a new basis for formulating program properties. Its abstractions allow for generalized recursive proof, verification and static analysis methods by induction on both program structure, control, and data. Examples of particular instances include Floyd's handling of loop cut-points as well as nested loops, Burstall's intermittent assertion total correctness proof method, and Podelski-Rybalchenko transition invariants.

Categories and Subject Descriptors D.2.4 [*Software/Program Verification*]; D.3.1 [*Formal Definitions and Theory*]; F.3.1 [*Specifying and Verifying and Reasoning about Programs*].

General Terms Languages, Reliability, Security, Theory, Verification.

Keywords Abstract Interpretation, Induction, Proof, Safety, Static analysis, Variant function, Verification, Termination.

1. Introduction

Floyd/Turing program proof methods for invariance and termination [24, 40, 59] have inspired most sound static analysis methods.

For static *invariance* analysis by abstract interpretation [19, 21], a key step is to express the strongest invariant as a fixpoint and next to approximate this strongest invariant to automatically infer an abstract inductive invariant using the constructive fixpoint approximation methods.

For static *termination* analysis, the discovery of variant functions is either decidable in limited cases [54] or else is based on the Floyd/Turing idea of variant functions into well-founded sets obtained by observing quantities that strictly decrease within loops while remaining lower-bounded, or dually. So most termination analysis methods indirectly reduce to a relational invariance analysis hence can reuse classical static analysis methods.

The abstract interpretation design principle is instantiated with suitable abstractions for *safety* and *termination* analysis, proof, and checking/verification (either potential termination or definite termination for nondeterministic systems).

The first main idea for termination is that there exists a most precise variant function that can be expressed in fixpoint form by abstract interpretation of a termination collecting semantics itself abstracting the program operational trace semantics. This yields new static analysis methods automatically inferring abstractions of that variant function by the constructive fixpoint approximation methods of abstract interpretation.

The second main idea introduced in this paper both for safety and termination is that of *semantic structural induction*, including termination proofs, over *trace segment covers* and their abstractions. Trace segments are more powerful than binary relations between states which have been used traditionally in program termination proofs (for example, the transition invariants used in [53] are binary relation abstractions of the set of trace segments). Examples include structural induction on the program syntax (including loop invariants à la Floyd [40]), induction on data, à la Burstall [3], the covering of the transition relation closure by well-founded relations, à la Podelski-Rybalchenko [53], their combinations and generalizations.

2. Fixpoints, fixpoint induction, abstraction, and approximation

We express semantics as *fixpoints* of maps $f \in A \mapsto A$ i.e. elements $x \in A$ such that $x = f(x)$. We let $\mathsf{lfp}^{\sqsubseteq}_a f$ be the *least fixpoint* of $f \in A \mapsto A$ on the poset $\langle A, \sqsubseteq\rangle$ greater than or equal to $a \in A$, if any. The dual notion is that of *greatest fixpoint* $\mathsf{gfp}^{\sqsubseteq}_a f$. We write $\mathsf{lfp}^{\sqsubseteq} f$ if a is the infimum of A, and $\mathsf{lfp}\, f$ if the partial order $\sqsubseteq$ is clear from the context. By Tarski/Pataria's fixpoint theorem [50, 58], $\mathsf{lfp}^{\sqsubseteq}_a f = \bigsqcap\{P \in A \mid a \sqsubseteq P \wedge f(P) \sqsubseteq P\}$ exists for f increasing[1] on a complete lattice $\langle A, \sqsubseteq, a, \top, \sqcup, \sqcap\rangle$[2] or on a cpo $\langle A, \sqsubseteq, a, \sqcup\rangle$[3]. The *fixpoint iterates* are $f^0 \triangleq a$, $\forall n \in \mathbb{N} : f^{n+1} = f(f^n)$, $f^\omega \triangleq \bigsqcup_{n\in\mathbb{N}} f^n$ which is $\mathsf{lfp}^{\sqsubseteq}_a f$ when $a \sqsubseteq f(a)$ is a pre-fixpoint and f is continuous[4,5,6]. If f is increasing but not continuous, transfinite iterations may have to be used [22].

[1] $f \in A \mapsto A$ is *increasing* (also *monotone*, *isotone*, ...) on a poset $\langle A, \sqsubseteq\rangle$ if and only if $\forall x, y \in A : (x \sqsubseteq y) \implies (f(x) \sqsubseteq f(y))$ [36].

[2] A *complete lattice* $\langle A, \sqsubseteq, \bot, \top, \sqcup, \sqcap\rangle$ is a poset s. t. any subset has a least upper bound (lub) $\sqcup$, hence a greatest lower bound (glb) $\sqcap$, $\bot = \sqcup\emptyset$, $\top = \sqcap\emptyset$.

[3] A *complete partial order* (cpo) $\langle A, \sqsubseteq, \bot, \sqcup\rangle$ is a poset $\langle A, \sqsubseteq\rangle$ such that any increasing chain $C \subseteq A$ such that $\forall x, y \in C : x \sqsubseteq y \vee y \sqsubseteq x$ has a least upper bound (lub) $\sqcup C$, hence has an infimum $\bot = \sqcup\emptyset$ for the empty chain.

[4] $f \in A \mapsto A$ is *continuous* on a poset $\langle A, \sqsubseteq, \sqcup\rangle$ if and only if for all increasing chains $C \in \wp(A)$ such that its lub $\sqcup C$ does exist then the lub $\sqcup f[C]$ exists and is such that $\sqcup f[C] = f(\sqcup C)$.

[5] $\wp(X)$ or 2^X is the powerset of X i.e. the set of all subsets of a set X.

[6] The *post-image* (or *image*) of $X \in \wp(A)$ by a map $f \in A \mapsto B$ is $f[X] \triangleq \{f(x) \mid x \in X\} \in \wp(B)$.

[*] Work supported in part by the CMACS NSF Expeditions in Computing award 0926166.

POPL'12, January 25–27, 2012, Philadelphia, PA, USA.
Copyright © 2012 ACM 978-1-4503-1083-3/12/01...$10.00

Fixpoint induction follows immediately as a sound ($\Longleftarrow$) and complete ($\Longrightarrow$) proof method since for all $S \in A$,

$$\mathsf{lfp}_a^{\sqsubseteq} f \sqsubseteq S \quad \Longleftrightarrow \quad \exists P \in A : a \sqsubseteq P \wedge f(P) \sqsubseteq P \wedge P \sqsubseteq S \ .$$

S is called a *specification* or *invariant* and P is an *inductive invariant*. The idea is that to prove an invariant S, one has to check (in checking/verification methods), to guess (in proof methods) or to compute (in analysis methods) a stronger inductive invariant P.

Following [19, 21], abstraction is formalized by *Galois connections*[7] $\langle A, \sqsubseteq\rangle \xleftarrow[\alpha]{\gamma}\!\!\!\rightarrow \langle B, \preceq\rangle$ between posets $\langle A, \sqsubseteq\rangle$ and $\langle B, \preceq\rangle$ meaning that $\alpha \in A \mapsto B$, $\gamma \in B \mapsto A$ and $\forall x \in A : \forall y \in B : \alpha(x) \preceq y \iff x \sqsubseteq \gamma(y)$. We write $\langle A, \sqsubseteq\rangle \xleftarrow[\alpha]{\gamma}\!\!\!\twoheadrightarrow \langle B, \preceq\rangle$ when the *abstraction* α is surjective (hence the *concretization* γ is injective), $\langle A, \sqsubseteq\rangle \twoheadleftarrow\!\!\!\xrightarrow[\alpha]{\gamma} \langle B, \preceq\rangle$ when α is injective (hence γ is surjective), and $\langle A, \sqsubseteq\rangle \twoheadleftarrow\!\!\!\xrightarrow[\alpha]{\gamma}\!\!\!\twoheadrightarrow \langle B, \preceq\rangle$ when α is bijective.

Given a concrete fixpoint characterization $\mathsf{lfp}_a^{\sqsubseteq} f$ of program properties on complete lattices or cpos $\langle A, \sqsubseteq\rangle$ with $a \sqsubseteq f(a)$ and an abstraction $\langle A, \sqsubseteq\rangle \xleftarrow[\alpha]{\gamma}\!\!\!\rightarrow \langle B, \preceq\rangle$, the sufficient *commutation condition* $\alpha \circ f = \overline{f} \circ \alpha$ (respectively *semi-commutation condition* $\alpha \circ f \mathrel{\dot{\preceq}} \overline{f} \circ \alpha$)[8] implies the *fixpoint abstraction* $\alpha(\mathsf{lfp}_a^{\sqsubseteq} f) = \mathsf{lfp}_{\alpha(a)}^{\preceq} \overline{f}$ (resp. *fixpoint approximation* $\alpha(\mathsf{lfp}_a^{\sqsubseteq} f) \preceq \mathsf{lfp}_{\alpha(a)}^{\preceq} \overline{f}$) [21]. The [semi-]commutation condition can be restricted to the iterates of f from a or to the elements of A which are $\sqsubseteq$-less that or equal to $\mathsf{lfp}_a^{\sqsubseteq} f$. The result also holds when α is continuous [13]. In absence of existence of a best abstraction, similar results can be obtained using only one of the abstraction or concretization functions [26].

3. Transition semantics

We consider a *programming language* with nondeterministic *programs* P. The set of all *states* of P is $\Sigma[\![\mathrm{P}]\!]$. The *transition relation* $\tau[\![\mathrm{P}]\!] \in \wp(\Sigma[\![\mathrm{P}]\!] \times \Sigma[\![\mathrm{P}]\!])$ describes the possible transitions between a state and its immediate successor states during program execution [11, 21]. The program small-step operational semantics is the *transition system* $\langle\Sigma[\![\mathrm{P}]\!], \tau[\![\mathrm{P}]\!]\rangle$. When restricting to *initial states* $\mathsf{I}[\![\mathrm{P}]\!] \in \wp(\Sigma[\![\mathrm{P}]\!])$, we write $\langle\Sigma[\![\mathrm{P}]\!], \mathsf{I}[\![\mathrm{P}]\!], \tau[\![\mathrm{P}]\!]\rangle$. The *termination/blocking states* are $\beta_\tau[\![\mathrm{P}]\!] \triangleq \{s \in \Sigma[\![\mathrm{P}]\!] \mid \forall s' \in \Sigma[\![\mathrm{P}]\!] : \langle s,\ s'\rangle \notin \tau[\![\mathrm{P}]\!]\}$. For brevity we write X for $X[\![\mathrm{P}]\!]$ e.g. $\langle\Sigma,\ \tau\rangle$, $\langle\Sigma,\ \mathsf{I},\ \tau\rangle$, or β_τ.

4. Trace semantics

4.1 Traces

We let Σ^n ($\Sigma^0 \triangleq \emptyset$), $\Sigma^+ = \bigcup_{n\in\mathbb{N}} \Sigma^n$, $\Sigma^* \triangleq \Sigma^+ \cup \{\varepsilon\}$, Σ^∞, $\Sigma^{+\infty} \triangleq \Sigma^+ \cup \Sigma^\infty$, and $\Sigma^{*\infty} \triangleq \Sigma^* \cup \Sigma^\infty$ be the set of all *finite traces of length* $n \in \mathbb{N}$, *non-empty finite*, *finite*, *infinite*, *non-empty finite or infinite*, and *finite or infinite traces* over the states Σ where ε is the *empty trace*.

We define the following operations on traces, writing $|\sigma|$ for the length of the trace $\sigma \in \Sigma^{+\infty}$, $\sigma[n, m]$, $0 \leqslant n \leqslant m$ for the *subtrace* $\sigma_n, \sigma_{n+1}, \ldots, \sigma_{\min(m,|\sigma|-1)}$ of σ, and $\sigma\sigma'$ for the *concatenation* of $\sigma, \sigma' \in \Sigma^{*\infty}$ (with $\sigma\varepsilon = \varepsilon\sigma = \sigma$ and $\sigma\sigma' = \sigma$ when $\sigma \in \Sigma^\infty$).

We define the following operations on sets of traces writing S for the set of traces $\{\sigma \in \Sigma^1 \mid \sigma_0 \in S\}$ made of one state of $S \in \wp(\Sigma)$ (for example, the termination states $\beta_\tau \triangleq \{s \in \Sigma \mid \forall s' \in \Sigma : \langle s,\ s'\rangle \notin \tau\}$ can also be understood as traces of length one $\{\sigma \in \Sigma^1 \mid \forall s \in \Sigma : \langle\sigma_0,\ s\rangle \notin \tau\}$), t for the set of traces $\{\sigma \in \Sigma^2 \mid \langle\sigma_0,\ \sigma_1\rangle \in t\}$ made of two consecutive states of the relation $t \in \wp(\Sigma \times \Sigma)$, $T^+ \triangleq T \cap \Sigma^+$ for the *selection of the non-empty finite traces* of $T \in \wp(\Sigma^{*\infty})$, $T^\infty \triangleq T \cap \Sigma^\infty$ for the *selection of the infinite traces* of T, $TT' \triangleq \{\sigma\sigma' \mid \sigma \in T \wedge \sigma' \in T'\}$ for the *concatenation* of sets of traces, and $T \fatsemi T' \triangleq \{\sigma s\sigma' \mid s \in \Sigma \wedge \sigma s \in T \wedge s\sigma' \in T'\}$ for the *sequencing* of sets of traces $T, T' \in \wp(\Sigma^{*\infty})$.

4.2 Partial and complete / maximal trace semantics

The *partial trace semantics* $\Theta^{+\infty}[\![\mathrm{P}]\!] \in \wp(\Sigma^{+\infty}[\![\mathrm{P}]\!])$ of a program P is a set of non-empty execution traces. In particular, the *partial trace semantics generated by a transition system* $\langle\Sigma,\ \tau\rangle$ is $\tau^{\ddot{+}\infty}[\![\mathrm{P}]\!]$ such that[9]

$$\begin{aligned}
\tau^{\ddot{n}}[\![\mathrm{P}]\!] &\triangleq \Big\{\sigma \in \Sigma^n \ \Big|\ \forall i \in [0, n-1) : \langle\sigma_i,\ \sigma_{i+1}\rangle \in \tau[\![\mathrm{P}]\!]\Big\}, \quad n \geqslant 0\\
\tau^{\infty}[\![\mathrm{P}]\!] &\triangleq \Big\{\sigma \in \Sigma^\infty \ \Big|\ \forall i \in \mathbb{N} : \langle\sigma_i,\ \sigma_{i+1}\rangle \in \tau[\![\mathrm{P}]\!]\Big\}\\
\tau^{\ddot{+}}[\![\mathrm{P}]\!] &\triangleq \bigcup_{n>0} \tau^{\ddot{n}}[\![\mathrm{P}]\!], \qquad \tau^{\ddot{+}\infty}[\![\mathrm{P}]\!] \ \triangleq \ \tau^{\ddot{+}}[\![\mathrm{P}]\!] \cup \tau^{\infty}[\![\mathrm{P}]\!] \ .
\end{aligned}$$

The *complete* or *maximal trace semantics* $\tau^n[\![\mathrm{P}]\!] \triangleq \alpha_M(\tau^{\ddot{n}}[\![\mathrm{P}]\!])$, $\tau^+[\![\mathrm{P}]\!] = \alpha_M(\tau^{\ddot{+}}[\![\mathrm{P}]\!])$ and $\tau^{+\infty}[\![\mathrm{P}]\!] \triangleq \alpha_M(\tau^{\ddot{+}\infty}[\![\mathrm{P}]\!])$ are obtained by the abstraction $\langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle \xleftarrow[\alpha_M]{\gamma_M}\!\!\!\twoheadrightarrow \langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle$ where

$$\alpha_M(T) \ \triangleq \ \bigcup_{n\in\mathbb{N}}\Big\{\sigma \in T \cap \Sigma^n \ \Big|\ \sigma_{n-1} \in \beta_\tau[\![\mathrm{P}]\!]\Big\} \cup T^\infty$$

eliminates those finite partial computations that are not terminated.

4.3 Fixpoint trace semantics

The partial trace semantics of a program P can be given in fixpoint form [28].

$$\tau^{\ddot{+}}[\![\mathrm{P}]\!] \ = \ \mathsf{lfp}_\emptyset^{\subseteq} \overleftarrow{\phi}_\tau^{\ddot{+}}[\![\mathrm{P}]\!] \ = \ \mathsf{lfp}_\emptyset^{\subseteq} \overrightarrow{\phi}_\tau^{\ddot{+}}[\![\mathrm{P}]\!], \qquad \tau^{\infty}[\![\mathrm{P}]\!] \ = \ \mathsf{gfp}_{\Sigma^\infty}^{\subseteq} \overleftarrow{\phi}_\tau^{\infty}[\![\mathrm{P}]\!]$$

$$\tau^{\ddot{+}\infty}[\![\mathrm{P}]\!] \ = \ \mathsf{lfp}_\emptyset^{\subseteq} \overleftarrow{\phi}_\tau^{\ddot{+}}[\![\mathrm{P}]\!] \cup \mathsf{gfp}_{\Sigma^\infty}^{\subseteq} \overleftarrow{\phi}_\tau^{\infty}[\![\mathrm{P}]\!] \ = \ \mathsf{lfp}_{\Sigma^\infty}^{\sqsubseteq} \overleftarrow{\phi}_\tau^{\ddot{+}\infty}[\![\mathrm{P}]\!]$$

$$\begin{aligned}
\overleftarrow{\phi}_\tau^{\ddot{+}}[\![\mathrm{P}]\!]T &\triangleq \Sigma^1 \cup \tau[\![\mathrm{P}]\!] \fatsemi T & \qquad \overrightarrow{\phi}_\tau^{\ddot{+}}[\![\mathrm{P}]\!]T &\triangleq \Sigma^1 \cup T \fatsemi \tau[\![\mathrm{P}]\!]\\
\overleftarrow{\phi}_\tau^{\infty}[\![\mathrm{P}]\!]T &\triangleq \tau[\![\mathrm{P}]\!] \fatsemi T & \qquad \overleftarrow{\phi}_\tau^{\ddot{+}\infty}[\![\mathrm{P}]\!]T &\triangleq \Sigma^1 \sqcup \tau[\![\mathrm{P}]\!] \fatsemi T
\end{aligned}$$

where $\langle\wp(\Sigma^{*\infty}),\ \sqsubseteq,\ \Sigma^\infty,\ \Sigma^*,\ \sqcup,\ \sqcap\rangle$ is a complete lattice for the *computational order* $(T_1 \sqsubseteq T_2) \triangleq (T_1^+ \subseteq T_2^+) \wedge (T_1^\infty \supseteq T_2^\infty)$ and $(T_1 \sqcup T_2) \triangleq (T_1^+ \cup T_2^+) \cup (T_1^\infty \cap T_2^\infty)$. The fixpoint complete trace semantics of a program P is calculated by abstraction with α_M.

$\tau^{+\infty}[\![\mathrm{P}]\!] \ = \ \mathsf{lfp}_\emptyset^{\subseteq} \overleftarrow{\phi}_\tau^{+}[\![\mathrm{P}]\!] \cup \mathsf{gfp}_{\Sigma^\infty}^{\subseteq} \overleftarrow{\phi}_\tau^{\infty}[\![\mathrm{P}]\!] \ = \ \mathsf{lfp}_{\Sigma^\infty}^{\sqsubseteq} \overleftarrow{\phi}_\tau^{+\infty}[\![\mathrm{P}]\!]$ where $\overleftarrow{\phi}_\tau^{+}[\![\mathrm{P}]\!]T \ \triangleq \ \beta_\tau[\![\mathrm{P}]\!] \cup \tau[\![\mathrm{P}]\!] \fatsemi T$, and $\overleftarrow{\phi}_\tau^{+\infty}[\![\mathrm{P}]\!]T \ \triangleq \ \beta_\tau[\![\mathrm{P}]\!] \sqcup \tau[\![\mathrm{P}]\!] \fatsemi T$.

5. Properties

Following [19, 21], properties are represented by the set of elements which have these properties. So the properties of programs which semantics are sets of traces in $\wp(\Sigma^{+\infty})$ are sets of sets of traces in $\wp(\wp(\Sigma^{+\infty}))$.

The *collecting semantics* $\{\Theta^{+\infty}[\![\mathrm{P}]\!]\} \in \wp(\wp(\Sigma^{+\infty}))$ is the strongest program property[10] of a program with trace semantics $\Theta^{+\infty}[\![\mathrm{P}]\!]$.

The *trace property abstraction* of program properties is $\langle\wp(\wp(\Sigma^{+\infty})),\ \subseteq\rangle \xleftarrow[\alpha_\Theta]{\gamma_\Theta}\!\!\!\rightarrow \langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle$ such that

$$\alpha_\Theta(P) \ \triangleq \ \bigcup P \qquad \text{and} \qquad \gamma_\Theta(Q) \ \triangleq \ \wp(Q)\ .$$

The traditional safety/liveness program properties are relative to the trace property abstraction of the collecting semantics $\alpha_\Theta(\{\Theta^{+\infty}[\![\mathrm{P}]\!]\}) = \Theta^{+\infty}[\![\mathrm{P}]\!] \in \wp(\Sigma^{+\infty})$.

Some program properties are not trace properties [5]. An example is "all program executions are deterministic" which is $\{\{\sigma\} \mid$

[7] [21] also introduced formalizations of abstraction using closure operators, ideals, congruences, etc. and showed all of them to be equivalent to Galois connections.

[8] $\dot{\sqsubseteq}$ is the pointwise extension of a partial order $\sqsubseteq$ to maps $f \mathrel{\dot{\sqsubseteq}} g \triangleq \forall x : f(x) \sqsubseteq g(x)$.

[9] $[n, m] \triangleq \{n, n+1, \ldots, m\}$ is the closed interval, $\emptyset$ when $m < n$, while $[n, m) \triangleq \{n, n+1, \ldots, m-1\}$ is left closed and right opened, $\emptyset$ when $m \leqslant n$.

[10] *strongest* in that the collecting semantics implies all other program properties (where logical implication $A \implies B$ is interpreted as $A \subseteq B$).

$\sigma \in \Sigma^{+\infty}\} \in \wp(\wp(\Sigma^{+\infty}))$ [11]. The corresponding trace property abstraction is $\alpha_\Theta(\{\{\sigma\} \bigm| \sigma \in \Sigma^{+\infty}\}) = \Sigma^{+\infty} \in \wp(\Sigma^{+\infty})$ which would allow any non-deterministic behavior so that determinism in the concrete domain $\wp(\wp(\Sigma^{+\infty}))$ is completely lost in the abstract domain $\wp(\Sigma^{+\infty})$.

For safety and termination and from now on, we only have to consider trace properties, which form a complete Boolean lattice $\langle\wp(\Sigma^{+\infty}),\ \subseteq,\ \emptyset,\ \Sigma^{+\infty},\ \cup,\ \cap,\ \neg\rangle$ where the partial order $\subseteq$ is logical implication and the complement is $\neg X \triangleq \Sigma^{+\infty} \setminus X$ [12].

6. Safety trace semantics

We now illustrate the classical abstract interpretation framework by generalizing invariance verification and static analysis to arbitrary safety properties. Safety properties are abstractions of program trace properties (essentially forgetting about liveness properties).

6.1 Safety abstraction

The *prefix abstraction* of a set T of traces is the topological closure[13]

$$\begin{aligned} \mathsf{pf}(\sigma) &\triangleq \{\sigma' \in \Sigma^{+\infty} \bigm| \exists\sigma'' \in \Sigma^{*\infty} : \sigma = \sigma'\sigma''\} \\ \mathsf{pf}(T) &\triangleq \bigcup\{\mathsf{pf}(\sigma) \bigm| \sigma \in T\}\,. \end{aligned}$$

The prefix abstraction expresses the fact that program executions can only be observed for a finite period of time ($\forall T : \varepsilon \notin \mathsf{pf}(T)$).

The *limit abstraction* of a set of traces is the topological closure

$$\mathsf{lm}(T) \triangleq T \cup \{\sigma \in \Sigma^\infty \mid \forall n \in \mathbb{N} : \sigma[0,n] \in T\}\,.$$

The limit abstraction expresses the fact that when observing program executions for finite periods of time it is impossible to distinguish between non-terminating and unbounded finite executions.

The *safety abstraction* of a set of traces is the topological closure

$$\mathsf{sf} \triangleq \mathsf{lm} \circ \mathsf{pf} = \mathsf{pf} \circ \mathsf{lm} \circ \mathsf{pf}\,.$$

The safety abstraction provides the strongest program property resulting from finite observations of program executions (excluding the observation of infinite executions).

(Topological) closures $\rho \in A \mapsto A$ on a poset $\langle A,\ \leqslant\rangle$ are abstractions[14] $\langle A,\ \leqslant\rangle \xleftarrow[\rho]{1_A} \langle\rho[A],\ \leqslant\rangle$.

6.2 Safety trace properties

The *safety trace properties* are

$$\mathsf{SF} \triangleq \mathsf{sf}[\wp(\Sigma^{+\infty})] = \{\mathsf{sf}(P) \mid P \in \wp(\Sigma^{+\infty})\} = \{P \in \wp(\Sigma^{+\infty}) \mid \mathsf{sf}(P) = P\}\,.$$

We have the Galois isomorphism

$$\langle\mathsf{SF},\ \subseteq\rangle \xleftarrow[\mathsf{pf}^+]{\mathsf{lm}} \langle\mathsf{pf}^+[\wp(\Sigma^+)],\ \subseteq\rangle$$

where $\mathsf{pf}^+(T) = \mathsf{pf}(T)^+$ and so safety trace properties can equivalently be represented by their finite prefixes in Sect. 6.4 and 6.5.

6.3 Safety semantics

The *safety semantics* of a program P is its strongest safety property

$$\tau^{\mathsf{sf}}[\![\mathsf{P}]\!] \triangleq \mathsf{sf}(\tau^{\ddot{+}\infty}[\![\mathsf{P}]\!]) \simeq \mathsf{pf}^+ \circ \mathsf{sf}(\tau^{\ddot{+}\infty}[\![\mathsf{P}]\!])\,.$$

6.4 Fixpoint safety semantics

It follows, by fixpoint abstraction, that the safety semantics of a program P with operational semantics $\langle\Sigma,\ \tau\rangle$ is

$$\begin{aligned} \tau^{\mathsf{sf}}[\![\mathsf{P}]\!] &= \mathsf{lfp}^{\subseteq}_{\emptyset}\,\overrightarrow{\phi}^{\mathsf{sf}}_{\tau}[\![\mathsf{P}]\!] = \mathsf{lfp}^{\subseteq}_{\emptyset}\,\overleftarrow{\phi}^{\mathsf{sf}}_{\tau}[\![\mathsf{P}]\!] && \text{where} \\ \overrightarrow{\phi}^{\mathsf{sf}}_{\tau}[\![\mathsf{P}]\!]T &\triangleq \Sigma^1 \cup T \fatsemi \tau[\![\mathsf{P}]\!] && \text{forward trace transformer} \\ \overleftarrow{\phi}^{\mathsf{sf}}_{\tau}[\![\mathsf{P}]\!]T &\triangleq \Sigma^1 \cup \tau[\![\mathsf{P}]\!] \fatsemi T && \text{backward trace transformer.} \end{aligned}$$

6.5 Proofs in the safety trace domain

By fixpoint induction, one immediately gets new forward and backward sound and complete safety proof methods[15] generalizing invariance [37, 40, 48, 49]. For all safety specifications $S \in \mathsf{SF}$,

$$\begin{aligned} \tau^{\mathsf{sf}}[\![\mathsf{P}]\!] \subseteq S &\iff \exists P \in \mathsf{SF} : \Sigma^1 \subseteq P \wedge \tau[\![\mathsf{P}]\!] \fatsemi P \subseteq P \wedge P \subseteq S \\ &\iff \exists P \in \mathsf{SF} : \Sigma^1 \subseteq P \wedge P \fatsemi \tau[\![\mathsf{P}]\!] \subseteq P \wedge P \subseteq S\,. \end{aligned}$$

Observe that forward and backward safety semantics and proof methods are respectively equivalent. This property is preserved by relational abstractions in next Sect. 7, but this is not the general case (e.g. with abstractions of Sect. 7.6). [42] is an example of static analysis in the safety trace domain.

7. Invariance / reachability semantics

Invariance/reachability is an abstraction of safety and so invariance proof methods are abstractions of safety proof methods.

7.1 Relational abstraction

The *relational abstraction* $\langle\mathsf{SF},\ \subseteq\rangle \xleftarrow[\alpha^{\mathsf{R}}]{\gamma^{\mathsf{R}}} \langle\wp(\Sigma \times \Sigma),\ \subseteq\rangle$ such that

$$\begin{aligned} \alpha^{\mathsf{R}}(T) &\triangleq \{\langle\sigma_0,\ \sigma_{n-1}\rangle \mid n > 0 \wedge \sigma \in \Sigma^n \cap T\} \qquad (1) \\ \gamma^{\mathsf{R}}(R) &\triangleq \{\sigma \in \Sigma^n \mid n > 0 \wedge \langle\sigma_0,\ \sigma_{n-1}\rangle \in R\} \end{aligned}$$

abstracts traces by a relation between their initial and final states (so that intermediate computations are lost in that abstraction).

7.2 Relational invariance / reachability abstraction

Applied to a safety semantics which is prefix-closed, the relational abstraction provides a relation between initial and current states (where, in particular, "initial" can be any state).

The abstraction $\alpha^{\mathsf{R}} \circ \mathsf{sf}$ is therefore equal to the *relational reachability abstraction* $\langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle \xleftarrow[\alpha^{\mathsf{R}^*}]{\gamma^{\mathsf{R}^*}} \langle\wp(\Sigma \times \Sigma),\ \subseteq\rangle$ such that

$$\begin{aligned} \alpha^{\mathsf{R}^*}(T) &\triangleq \{\langle\sigma_0,\ \sigma_i\rangle \mid \exists n : 0 \leqslant i < n \wedge \sigma \in \Sigma^n \cap T\} \\ \gamma^{\mathsf{R}^*}(R) &\triangleq \{\sigma \in \Sigma^n \mid n > 0 \wedge \forall i \in [0,n) : \langle\sigma_0,\ \sigma_i\rangle \in R\} \end{aligned}$$

abstract traces by a relation between their initial and current states.

7.3 Relational invariance / reachability semantics

The relational invariance/reachability semantics of a program P is its strongest relational reachability property

$$\begin{aligned} &\tau^{\mathsf{R}}[\![\mathsf{P}]\!] \triangleq \alpha^{\mathsf{R}}(\tau^{+\infty}[\![\mathsf{P}]\!]) \\ &\tau^{\mathsf{R}^*}[\![\mathsf{P}]\!] \triangleq \alpha^{\mathsf{R}}(\tau^{\ddot{+}\infty}[\![\mathsf{P}]\!]) = \alpha^{\mathsf{R}^*}(\tau^{+\infty}[\![\mathsf{P}]\!]) = \alpha^{\mathsf{R}}(\tau^{\mathsf{sf}}[\![\mathsf{P}]\!]) = \alpha^{\mathsf{R}^*}(\tau^{\mathsf{sf}}[\![\mathsf{P}]\!])\,. \end{aligned}$$

7.4 Fixpoint relational invariance / reachability semantics

The commutation condition applied to the transformer of the safety semantics $\tau^{\mathsf{sf}}[\![\mathsf{P}]\!]$ yields the fixpoint characterization of the relational reachability semantics of a program P with operational semantics $\langle\Sigma,\ \tau\rangle$

$$\tau^{\mathsf{R}^*}[\![\mathsf{P}]\!] = \mathsf{lfp}^{\subseteq}_{\emptyset}\,\overrightarrow{\phi}^{\mathsf{R}^*}_{\tau}[\![\mathsf{P}]\!] = \mathsf{lfp}^{\subseteq}_{\emptyset}\,\overleftarrow{\phi}^{\mathsf{R}^*}_{\tau}[\![\mathsf{P}]\!]$$

where[16]

[11] Assuming inputs, if any, to be part of the states.

[12] $X \setminus Y \triangleq \{x \in X \mid x \notin Y\}$ is the *set difference*.

[13] A *topological closure* on a poset $\langle A,\ \leqslant,\ \vee\rangle$ with partial-order $\leqslant$ and lub $\vee$, if any, is a map $\rho \in A \mapsto A$ which is *extensive* $\forall x \in A : x \leqslant \rho(x)$, *idempotent* $\forall x \in A : \rho(\rho(x)) = \rho(x)$, and finite *lub-preserving* $\forall x, y \in A : \rho(x \vee y) = \rho(x) \vee \rho(y)$. This implies that ρ is *increasing*. A *closure* is extensive, idempotent, and increasing.

[14] 1_A is the *identity* map (respectively relation) on the set A mapping any element $x \in A$ to itself $1_A(x) = x$ (resp. $1_A \triangleq \{\langle x,\ x\rangle \mid x \in A\}$).

[15] In case a temporal logic is used for expressing the inductive safety invariant, this is *relative completeness* subject to an *expressivity* hypothesis of the temporal logic ensuring $P \in \mathsf{SF}$ to be expressible in the logic, see e.g. [10].

[16] The *post-image* (or *right-image*) of $X \in \wp(A)$ by a relation $r \in \wp(A \times B)$ is $r[X] \triangleq \{y \mid \exists x \in X : \langle x,\ y\rangle \in r\}$ also written $\mathbf{post}[r]X$.

$$\overrightarrow{\phi}_{\tau}^{\mathsf{R}^*}[\![\mathtt{P}]\!](R) \triangleq 1_\Sigma \cup R \circ \tau[\![\mathtt{P}]\!] \qquad \text{forward transformer}$$
$$\overleftarrow{\phi}_{\tau}^{\mathsf{R}^*}[\![\mathtt{P}]\!](R) \triangleq 1_\Sigma \cup \tau[\![\mathtt{P}]\!] \circ R \qquad \text{backward transformer.}$$

7.5 Relational invariance / reachability proof methods

Applying fixpoint induction to the fixpoint relational reachability semantics, we get sound and complete forward and backward proof methods for a specification $S \in \wp(\Sigma \times \Sigma)$ [23], respectively generalizing [40, 49] and [37, 48].

$$\begin{aligned}\tau^{\mathsf{R}^*}[\![\mathtt{P}]\!] \subseteq S &\iff \exists R \in \wp(\Sigma\times\Sigma) : 1_\Sigma \subseteq R \wedge R \circ \tau[\![\mathtt{P}]\!] \subseteq R \wedge R \subseteq S\\ &\iff \exists R \in \wp(\Sigma\times\Sigma) : 1_\Sigma \subseteq R \wedge \tau[\![\mathtt{P}]\!] \circ R \subseteq R \wedge R \subseteq S\ .\end{aligned}$$

7.6 Variations on invariance / reachability proof methods

Further abstractions yield other classical proof methods. It is possible to restrict to the *initial states* $\mathsf{I} \in \wp(\Sigma)$, $\langle \wp(\Sigma \times \Sigma), \subseteq\rangle \xleftarrow[\alpha_{\mathsf{I}}]{\gamma_{\mathsf{I}}} \langle \wp(\Sigma \times \Sigma), \subseteq\rangle$ where

$$\alpha_{\mathsf{I}}(R) \triangleq \{\langle s,\ s'\rangle \mid s \in \mathsf{I} \wedge \langle s,\ s'\rangle \in R\} \tag{2}$$

and the *final states* $\mathsf{F} \in \wp(\Sigma)$, $\langle \wp(\Sigma \times \Sigma), \subseteq\rangle \xleftarrow[\alpha^{\mathsf{F}}]{\gamma^{\mathsf{F}}} \langle \wp(\Sigma \times \Sigma), \subseteq\rangle$ where

$$\alpha^{\mathsf{F}}(R) \triangleq \{\langle s,\ s'\rangle \mid \langle s,\ s'\rangle \in R \wedge s \in \mathsf{F}\}\ . \tag{3}$$

It is also possible to use an invariant so as to restrict to the reachable states $\langle \wp(\Sigma \times \Sigma), \subseteq\rangle \xleftarrow[\alpha^{r}]{\gamma^{r}} \langle \wp(\Sigma), \subseteq\rangle$ where

$$\alpha^{r}(R) \triangleq \{s' \mid \langle s,\ s'\rangle \in R\}\ . \tag{4}$$

Combining (2) and (4) we get forward invariance [40, 49] while (3) and the inverse of (4) yield backward invariance (called "subgoal induction" in [48]).

Proofs by reductio ad absurdum [23, 35] are obtained by $\langle \wp(\Sigma \times \Sigma), \subseteq\rangle \xleftarrow[\widetilde{\alpha}]{\widetilde{\gamma}} \langle \wp(\Sigma \times \Sigma), \supseteq\rangle$ where $\widetilde{\alpha}(R) \triangleq \neg R$.

8. Termination trace collecting semantics

Our objective is now to apply the abstract interpretation methodology of Sect. 2, as illustrated in Sect. 6—7 for the safety properties and their invariance abstractions, to termination.

Starting from a collecting trace semantics, we define termination properties by abstraction, derive fixpoint charaterizations by fixpoint abstraction, conceive proof and verification methods by fixpoint induction, and design static analysis methods by fixpoint approximation using widening [19].

8.1 Termination property

The *termination property* states either that all executions in the trace semantics $\Theta^{+\infty}[\![\mathtt{P}]\!]$ of a program $\mathtt{P}$ must always be finite

$$\Theta^{+\infty}[\![\mathtt{P}]\!] \subseteq \Sigma^{+}[\![\mathtt{P}]\!] \qquad \textit{definite termination}$$

or that the trace semantics $\Theta^{+\infty}[\![\mathtt{P}]\!]$ may be finite (hence must not always be infinite)

$$\Theta^{+\infty}[\![\mathtt{P}]\!] \cap \Sigma^{+}[\![\mathtt{P}]\!] \neq \emptyset \qquad \textit{potential termination.}$$

The *infinite extension abstraction*

$$\begin{aligned}\alpha^{\omega}(T) \triangleq T \cup \{\sigma_1\sigma_2 \in \Sigma^{\infty} \mid \sigma_1 \in \Sigma^{+} \wedge (\exists \sigma_2' \in \Sigma^{\infty} : \sigma_1\sigma_2' \in T \vee\\ \forall \sigma_2' \in \Sigma^{*} : \sigma_1\sigma_2' \notin T)\}\end{aligned}$$

is a topological closure and so $\langle \wp(\Sigma^{+\infty}), \subseteq\rangle \xleftarrow[\alpha^{\omega}]{\gamma^{\omega}} \langle \alpha^{\omega}[\wp(\Sigma^{+\infty})], \subseteq\rangle$ where γ^{ω} is the identity. We have

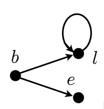
$$\begin{aligned}\tau^{+\infty}[\![\mathtt{P}]\!] \subseteq \Sigma^{+}[\![\mathtt{P}]\!] &\iff \alpha^{\omega}(\tau^{+\infty}[\![\mathtt{P}]\!]) \subseteq \Sigma^{+}[\![\mathtt{P}]\!],\\ \tau^{+\infty}[\![\mathtt{P}]\!] \cap \Sigma^{+}[\![\mathtt{P}]\!] \neq \emptyset &\iff \alpha^{\omega}(\tau^{+\infty}[\![\mathtt{P}]\!]) \cap \Sigma^{+}[\![\mathtt{P}]\!] \neq \emptyset\end{aligned}$$

and so, if necessary, we only need to consider semantics closed by α^{ω}.

8.2 Termination trace abstraction

The termination trace abstraction eliminates the program execution traces not starting by a state from which execution may/must terminate.

Example 1. Consider the example of the non-deterministic program `b:[ l:loop [] e:skip ]` with states $\{b, l, e\}$, transitions $\{\langle b,\ l\rangle, \langle b,\ e\rangle, \langle l,\ l\rangle\}$ and complete trace semantics $\{be, e, bllll\ldots, llll\ldots\}$. ☐



8.2.1 Potential termination trace abstraction

The *potential termination* or *may-terminate trace semantics* eliminates infinite traces.

Example 2. The potential termination trace semantics of program `b:[ l:loop [] e:skip ]` in Ex. **1** is $\{be, e\}$ since an execution starting in state b *may* terminate (by choosing a transition to state e). ☐

The corresponding *potential termination abstraction* is $\langle \wp(\Sigma^{+\infty}), \subseteq\rangle \xleftarrow[\alpha^{\mathsf{mt}}]{\gamma^{\mathsf{mt}}} \langle \wp(\Sigma^{+}), \subseteq\rangle$ and $\langle \wp(\Sigma^{+\infty}), \sqsubseteq\rangle \xleftarrow[\alpha^{\mathsf{mt}}]{\gamma'^{\mathsf{mt}}} \langle \wp(\Sigma^{+}), \subseteq\rangle$ where

$$\alpha^{\mathsf{mt}}(T) \triangleq T \cap \Sigma^{+},\quad \gamma^{\mathsf{mt}}(S) \triangleq S \cup \Sigma^{\infty} \quad\text{and}\quad \gamma'^{\mathsf{mt}}(S) \triangleq S\ .$$

The abstraction forgets about non-terminating executions. This abstraction corresponds to Dijkstra's weakest liberal/angelic precondition [37]. It is considered in [11] (together with backward reachability) to automatically compute *necessary* conditions for termination (in example **1**, this analysis would yield the potential termination states $\{b, e\}$ proving definite non-termination in state l).

8.2.2 Definite termination trace abstraction

The *definite termination* or *must-terminate trace semantics* eliminates all traces potentially branching, through local non-determinism, to non-termination.

Example 3. The definite termination trace semantics of program `b:[ l:loop [] e:skip ]` in Ex. **1** is $\{e\}$ since in state b there is a possibility of non-termination (by choosing a transition to state l). ☐

A trace is in the definite termination semantics if and only if it is finite, independently of the potential non-deterministic choices along that trace. The corresponding *definite termination abstraction* is

$$\alpha^{\mathsf{Mt}}(T) \triangleq \{\sigma \in T^{+} \mid \mathsf{pf}(\sigma) \cap \mathsf{pf}(T^{\infty}) = \emptyset\}$$

$\alpha^{\mathsf{Mt}} \in \langle \wp(\Sigma^{+\infty}), \sqsubseteq\rangle \hookrightarrow\!\!\!\!\rightarrow \langle \wp(\Sigma^{+}), \subseteq\rangle$ is a *retract*[17] and onto but not continuous[18]. However, on the following we consider only *transition closed* semantics [35] i.e. generated by a transition system (see counter example 5).

Example 4. If $T = \{ab, aba, ba, bb, ba^{\omega}\}$ then $\alpha^{\mathsf{mt}}(T) = \{ab, aba, ba, bb\}$ and $\alpha^{\mathsf{Mt}}(T) = \{ab, aba\}$ since $\mathsf{pf}(\sigma) \cap \mathsf{pf}(ba^{\omega}) = \emptyset$ for $\sigma = ab, aba$. ☐

This abstraction corresponds to Dijkstra's weakest/demonic precondition that is to the *definite termination analysis* we are mostly interested in for transition systems.

8.3 Termination trace semantics

The *potential termination collecting semantics* of a program $\mathtt{P}$ is therefore defined as

[17] A *retract* $r \in \langle A, \sqsubseteq\rangle \hookrightarrow \langle B, \leqslant\rangle$ where $B \subseteq A$ is increasing and idempotent. We write $r \in \langle A, \sqsubseteq\rangle \hookrightarrow\!\!\!\!\rightarrow \langle B, \leqslant\rangle$ when it is onto.

[18] Consider the $\sqsubseteq$-increasing chain $T_n \triangleq \{0\} \cup \{0i^{\omega} \mid i \geqslant n\}$, $n \geqslant 0$. We have $\bigsqcup_{n\geqslant 0} \alpha^{\mathsf{Mt}}(T_n) = \emptyset$ while $\bigcap_{n\geqslant 0}\{0i^{\omega} \mid i \geqslant n\} = \emptyset$ so that $\alpha^{\mathsf{Mt}}(\bigsqcup_{n\geqslant 0} T_n) = \alpha^{\mathsf{Mt}}(\{a\}) = \{a\}$.

$$\tau^{\mathsf{mt}}[\![\mathtt{P}]\!] \triangleq \alpha^{\mathsf{mt}}(\tau^{+\infty}[\![\mathtt{P}]\!]) \qquad \text{potential termination semantics}$$

while the *definite termination collecting semantics* of a program P is defined as

$$\tau^{\mathsf{Mt}}[\![\mathtt{P}]\!] \triangleq \alpha^{\mathsf{Mt}}(\tau^{+\infty}[\![\mathtt{P}]\!]) \qquad \text{definite termination semantics.}$$

8.4 Fixpoint termination trace semantics

By abstraction of the fixpoint trace semantics of Sect. 4.3, the strongest termination property of a program P with operational semantics $\langle\Sigma[\![\mathtt{P}]\!],\ \tau[\![\mathtt{P}]\!]\rangle$ and termination states $\beta_\tau[\![\mathtt{P}]\!]$ is

$$\begin{array}{rcll}
\tau^{\mathsf{mt}}[\![\mathtt{P}]\!] & = & \mathsf{lfp}^{\subseteq}_{\emptyset}\ \overleftarrow{\phi}^{\mathsf{mt}}_{\tau}[\![\mathtt{P}]\!] & \text{potential termination}\\
\overleftarrow{\phi}^{\mathsf{mt}}_{\tau}[\![\mathtt{P}]\!]T & \triangleq & \beta_\tau[\![\mathtt{P}]\!] \cup \tau[\![\mathtt{P}]\!] \mathbin{;} T & \\
\tau^{\mathsf{Mt}}[\![\mathtt{P}]\!] & = & \mathsf{lfp}^{\subseteq}_{\emptyset}\ \overleftarrow{\phi}^{\mathsf{Mt}}_{\tau}[\![\mathtt{P}]\!] & \text{definite termination}\\
\overleftarrow{\phi}^{\mathsf{Mt}}_{\tau}[\![\mathtt{P}]\!]T & \triangleq & \beta_\tau[\![\mathtt{P}]\!] \cup (\tau[\![\mathtt{P}]\!] \mathbin{;} T \cap \neg(\tau[\![\mathtt{P}]\!] \mathbin{;} \neg T)) &
\end{array}$$

where the term $\neg(\tau[\![\mathtt{P}]\!] \mathbin{;} \neg T)$ eliminates potential transitions towards non-terminating executions.

8.5 Proofs in the termination trace domain

Fixpoint induction provides formal methods to check fixpoint over-approximations, either $\tau^{\mathsf{mt}}[\![\mathtt{P}]\!] \subseteq S$ or $\tau^{\mathsf{Mt}}[\![\mathtt{P}]\!] \subseteq S$. Over-approximations yield necessary but not sufficient termination conditions which may introduce spurious infinite traces for which the proof cannot be done. The proof method is therefore useful to prove invariance under termination assumptions[19] but not for may/must termination.

On the contrary, termination proofs require fixpoint under-approximations $S \subseteq \tau^{\mathsf{mt}}[\![\mathtt{P}]\!]$ or $S \subseteq \tau^{\mathsf{Mt}}[\![\mathtt{P}]\!]$. Under-approximations yield sufficient but not necessary termination conditions and so may eliminate some termination cases for which the termination proof could have been done automatically. Fixpoint under-approximation proof methods have been proposed e.g. by [15, Sect. 11] and would yield the requested termination proof methods. More classically, we will favor over-approximations for static analysis.

9. Termination domain

Programs may not always potentially/definitely terminate in all states. So one problem is to determine for which states $\mathsf{I} \in \wp(\Sigma)$ do executions starting from these states may/must terminate.

9.1 Termination domain abstraction

This potential/definite termination domain semantics is provided by the *weakest precondition abstraction* $\langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle \underset{\alpha^{\mathsf{w}}}{\overset{\gamma^{\mathsf{w}}}{\longleftrightarrow}} \langle\wp(\Sigma),\ \subseteq\rangle$ of the termination trace semantics, such that

$$\alpha^{\mathsf{w}}(T) \triangleq \{\sigma_0 \mid \sigma \in T\} \qquad \text{precondition abstraction.}$$

9.2 Termination domain semantics

$$\begin{array}{rcll}
\tau^{\mathsf{wmt}}[\![\mathtt{P}]\!] & \triangleq & \alpha^{\mathsf{w}}(\tau^{\mathsf{mt}}[\![\mathtt{P}]\!]) & \text{potential termination}\\
\tau^{\mathsf{wMt}}[\![\mathtt{P}]\!] & \triangleq & \alpha^{\mathsf{w}}(\tau^{\mathsf{Mt}}[\![\mathtt{P}]\!]) & \text{definite termination.}
\end{array}$$

Using Dijkstra's notations [37], $\tau^{\mathsf{wmt}}[\![\mathtt{P}]\!] = \mathsf{wlp}[\![\mathtt{P}]\!]\mathbf{true}$ and $\tau^{\mathsf{wMt}}[\![\mathtt{P}]\!] = \mathsf{wp}[\![\mathtt{P}]\!]\mathbf{true}$.

9.3 Fixpoint termination domain semantics

By fixpoint abstraction of the termination trace semantics in Sect. 8.4 using transformer commutation, we get Dijkstra's fixpoint weakest (liberal) termination precondition semantics [38][20]

$$\begin{array}{rcll}
\tau^{\mathsf{wmt}}[\![\mathtt{P}]\!] & = & \mathsf{lfp}^{\subseteq}_{\emptyset}\ \overrightarrow{\phi}^{\mathsf{wmt}}_{\tau}[\![\mathtt{P}]\!] & \text{weakest liberal termin. precond.}\\
\overrightarrow{\phi}^{\mathsf{wmt}}_{\tau}[\![\mathtt{P}]\!](R) & \triangleq & \beta_\tau[\![\mathtt{P}]\!] \cup \tau[\![\mathtt{P}]\!]^{-1}[R] & \\
\tau^{\mathsf{wMt}}[\![\mathtt{P}]\!] & = & \mathsf{lfp}^{\subseteq}_{\emptyset}\ \overrightarrow{\phi}^{\mathsf{wMt}}_{\tau}[\![\mathtt{P}]\!] & \text{weakest termination precondition}\\
\overrightarrow{\phi}^{\mathsf{wMt}}_{\tau}[\![\mathtt{P}]\!](R) & \triangleq & \beta_\tau[\![\mathtt{P}]\!] \cup (\tau[\![\mathtt{P}]\!]^{-1}[R] \cap \neg\tau[\![\mathtt{P}]\!]^{-1}[\neg R])\ . &
\end{array}$$

9.4 Proof and static analysis in the termination domain

As was the case in Sect. 8.5, fixpoint induction is useful for over-approximations, which can be automatically inferred by static analysis [11, 12]. On the contrary, termination proofs require under-approximations [15, Sect. 11] proof methods. Although static under-approximation analysis is possible (e.g. [34]), this is not the termination proof technique which is used in practice [38].

10. Termination proofs for the trace semantics generated by a transition system

In practice a termination proof is decomposed in two parts. First a necessary termination condition is found by over-approximating $\tau^{\mathsf{wmt}}[\![\mathtt{P}]\!]$ or $\tau^{\mathsf{wMt}}[\![\mathtt{P}]\!]$. Then this necessary termination condition is shown to be sufficient by Floyd/Turing variant function method (e.g. [17]) or inversely (e.g. [8]). This corresponds to different abstractions, specific to the trace semantics generated by a transition system, that we now elaborate.

10.1 Transition-based termination proofs

A program which trace semantics is generated by a transition system $\langle\Sigma,\ \tau\rangle$ definitely terminates if and only if the program transition relation is well-founded[21].

$$\tau^{+\infty}[\![\mathtt{P}]\!] \subseteq \Sigma^{+}[\![\mathtt{P}]\!] \iff \langle\Sigma,\ \tau\rangle \text{ is well-founded.}$$

In practice one considers traces starting from initial states $\mathsf{I} \in \wp(\Sigma)$, e.g. I is the termination domain of Sect. 9. In that case a program which trace semantics is generated by a transition system $\langle\Sigma,\ \tau\rangle$ definitely terminates for traces starting from initial states $\mathsf{I} \in \wp(\Sigma)$ if and only if the program transition relation restricted to reachable states is well-founded.

$$\alpha^{\mathsf{i}}(\mathsf{I})(\tau^{+\infty}[\![\mathtt{P}]\!]) \subseteq \Sigma^{+}[\![\mathtt{P}]\!] \iff \langle\alpha^{\mathsf{r}}(\alpha^{\mathsf{i}}(\mathsf{I})(\tau^{+\infty}[\![\mathtt{P}]\!])),\ \tau\rangle \text{ is well-founded}$$

where the *initialization abstraction* $\langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle \underset{\alpha^{\mathsf{i}}(\mathsf{I})}{\overset{\gamma^{\mathsf{i}}(\mathsf{I})}{\longleftrightarrow}} \langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle$ is

$$\begin{array}{rcll}
\alpha^{\mathsf{i}} & \in & \wp(\Sigma) \mapsto (\Sigma^{+\infty} \mapsto \Sigma^{+\infty}) & \text{initialization abstraction}\\
\alpha^{\mathsf{i}}(\mathsf{I})T & \triangleq & \{\sigma \in T \mid \sigma_0 \in \mathsf{I}\} &
\end{array}$$

and the *reachable states abstraction* $\langle\wp(\Sigma^{+\infty}),\ \subseteq\rangle \underset{\alpha^{\mathsf{r}}}{\overset{\gamma^{\mathsf{r}}}{\longleftrightarrow}} \langle\wp(\Sigma),\ \subseteq\rangle$ is

$$\alpha^{\mathsf{r}}(T) \triangleq \{\, s \mid \exists \sigma \in \Sigma^{*}, \sigma' \in \Sigma^{*\infty} : \sigma s \sigma' \in T \,\} \qquad \text{reachability abstraction.}$$

The transition-based termination proof method is sound and complete. As noticed in Sect. 9, the precondition I can be inferred automatically by static analysis. Moreover, an over-approximation $R \supseteq \alpha^{\mathsf{r}}(\alpha^{\mathsf{i}}(\mathsf{I})(\tau^{+\infty}[\![\mathtt{P}]\!])) = \tau[\![\mathtt{P}]\!]^{*}[\mathsf{I}]$ [22] of the reachable states can be computed by classical abstract interpretation algorithms [19].

[19] e.g. for Ex. **1**, $\{b, e, l\}$ is invariant, $\{b, e\}$ is invariant under potential termination hypothesis, and $\{e\}$ is invariant under definite termination hypothesis.

[20] The pre-image of $Y \in \wp(A)$ by a relation $r \in \wp(A \times B)$ is $r^{-1}[Y] \triangleq \{x \mid \exists y \in Y : \langle x,\ y\rangle \in r\}$ also written $\mathbf{pre}[r]Y$ while $\neg r^{-1}[\neg Y] \triangleq \{x \mid \forall y : y \in Y \implies \langle x,\ y\rangle \in r\}$ is $\widetilde{\mathbf{pre}}[r]Y$.

[21] A relation $\prec \in \wp(\mathbb{W} \times \mathbb{W})$ on a set $\mathbb{W}$ is *well-founded* if and only if there is no strictly decreasing infinite chain $x_0 \succ x_1 \succ \ldots \succ x_n \succ x_{n+1} \succ \ldots$ of elements $x_0, x_1, \ldots, x_n, x_{n+1}, \ldots$ of $\mathbb{W}$. $\langle\mathbb{W},\ \prec\rangle$ is called a *well-founded set*. A *(total) well-order* is well-founded (total) strict order relation $\prec$. The set of all well-founded relations in $\wp(\mathbb{W} \times \mathbb{W})$ is written $\mathcal{W}\!f(\mathbb{W} \times \mathbb{W})$.

[22] t^{*} is the reflexive transitive closure of a binary relation t.

10.2 Transition abstraction

If the program semantics $\Theta^{+\infty}[\![\mathrm{P}]\!]$ is not generated by a transition system we might consider the transition abstraction $\langle \Sigma, \overrightarrow{\alpha}(\Theta^{+\infty}[\![\mathrm{P}]\!])\rangle$ where the *transition abstraction* $\langle \wp(\Sigma^{+\infty}), \subseteq\rangle \xleftrightarrows[\overrightarrow{\alpha}]{\overrightarrow{\gamma}} \langle \wp(\Sigma \times \Sigma), \subseteq\rangle$ is

$$\overrightarrow{\alpha}(T) \triangleq \{\langle s, s'\rangle \mid \exists \sigma, \sigma' : \sigma s s' \sigma' \in T\} \qquad \text{transition abstraction}$$

but the following counter-example shows that the condition is sufficient but not necessary.

Counter-example 5. Let $T \triangleq \{ab, ba\}$ be a trace semantics. The corresponding transition relation $\tau \triangleq \overrightarrow{\alpha}(T) = \{\langle a, b\rangle, \langle b, a\rangle\}$ generates the infinite trace *abababa*... and so the transition relation τ restricted to the reachable states $\{a, b\}$ is not well-founded. □

Another counter-example is fairness [35]. In the following, we consider complete/maximal trace semantics T that are *transition closed* (also *generated by a transition system*) that is $\overrightarrow{\alpha}(T) = T$ or equivalently T is closed by elimination of strict prefixes, closed by extension by fusion, and closed by limits [35, Th. 2.6.8].

11. Variant semantics

It remains to design verification and static analysis methods to show that $\langle R, \tau\rangle$ is well-founded where

$$R \supseteq \alpha^{\mathsf{r}}(\alpha^{\mathsf{i}}(I)(\tau^{+\infty}[\![\mathrm{P}]\!])) = \tau[\![\mathrm{P}]\!]^*[I]$$

over-approximates the reachable states. There are two important remarks.

1. If $\tau \subseteq r$ and $\langle R, r\rangle$ is well-founded then $\langle R, \tau\rangle$ is well-founded.
2. $\langle R, \tau\rangle$ is well-founded if and only if there exists a variant function $v \in \Sigma \not\mapsto \mathbb{W}$[23] into a well-founded set $\langle \mathbb{W}, \prec\rangle$ which domain is R[24].

So for the traces generated by a transition system, termination can be proved by mapping invariant states to a well-founded relation which is the principle of Floyd/Turing variant function method.

11.1 Variant function

A *variant function* $v \in \Sigma \not\mapsto \mathbb{W}$ is a partial function from the set of states into a well-founded set $\langle \mathbb{W}, \prec\rangle$ where $\prec$ is a well-founded relation on the set $\mathbb{W}$ (and $\preccurlyeq$ is its non-strict version). With appropriate hypotheses on states and the transition relation, the co-domain of the variant function can be fixed a priori and the variant function can be found by constraint solving e.g. [17, 54]. However, these methods are not as general as Floyd/Turing's method.

In mathematics, the ordinals provide a standard well-founded set thanks to ranking functions mapping each element of a well-founded set to its ordinal rank. So, up to a ranking function, the well-founded set $\langle \mathbb{W}, \prec\rangle$ can always be chosen as the class $\langle \mathbb{O}, <\rangle$ of ordinals. The intuition is that any execution σ starting in a state $\sigma_0 \in \mathsf{dom}(v)$ must terminate in "at most" $v(\sigma_0)$ execution steps while an execution σ starting in a state $\sigma_0 \notin \mathsf{dom}(v)$ might not terminate. We have $\tau \subseteq \{\langle s, s'\rangle \in \Sigma^2 \mid s \in \mathsf{dom}(v) \wedge v(s) \succ v(s')\}$ and this relation is well-founded on states, proving termination.

11.2 Variant abstraction

A variant function is an abstraction of a set of finite traces. It is a partial function which domain is the set of terminating states. Its value is an upper bound of the remaining number of "steps" to termination. It may be transfinite for unbounded non-determinism with unbounded execution trace lengths. Let us define

$$\begin{aligned}
\alpha^{\mathsf{rk}} &\in \wp(\Sigma \times \Sigma) \mapsto (\Sigma \not\mapsto \mathbb{O}) && \text{ranking abstraction}\\
\alpha^{\mathsf{rk}}(r)s &\triangleq 0 \quad \text{when} \quad \forall s' \in \Sigma : \langle s, s'\rangle \notin r\\
\alpha^{\mathsf{rk}}(r)s &\triangleq \mathsf{sup}\left\{\alpha^{\mathsf{rk}}(r)s' + 1 \;\middle|\; s' \in \mathsf{dom}(\alpha^{\mathsf{rk}}(r)) \wedge \langle s, s'\rangle \in r\right\}^{25}.
\end{aligned}$$

$\alpha^{\mathsf{rk}}(r)s$ extracts the well-founded part of relation r and provides the rank of the elements s of its domain. $\alpha^{\mathsf{v}}(T)$ does the same for the transition relation by abstracting the set T of finite traces

$$\begin{aligned}
\alpha^{\mathsf{v}} &\in \wp(\Sigma^{+}) \mapsto (\Sigma \not\mapsto \mathbb{W}) && \text{variant abstraction}\\
\alpha^{\mathsf{v}}(T) &\triangleq \lambda s \bullet \alpha^{\mathsf{rk}}(\overrightarrow{\alpha}(T))s\,.
\end{aligned}$$

It follows that the abstraction $\langle \wp(\Sigma^{+\infty}), \sqsubseteq\rangle \xleftarrow[\alpha^{\mathsf{v}} \circ \alpha^{\mathsf{mt}}]{\gamma'^{\mathsf{mt}} \circ \gamma^{\mathsf{v}}} \langle \Sigma \not\mapsto \mathbb{W}, \sqsubseteq^{\mathsf{v}}\rangle$ holds for potential termination and $\langle \wp(\Sigma^{+\infty}), \sqsubseteq\rangle \rightarrow \langle \Sigma \not\mapsto \mathbb{W}, \sqsubseteq^{\mathsf{v}}\rangle$ for definite termination. These abstractions state, by def. of $\sqsubseteq$, that adding finite execution traces or suppressing infinite traces can only, by def. of $\sqsubseteq^{\mathsf{v}}$, augment the termination domain and, maybe, increase execution times. It follows that the *computational variant order* is

$$v \sqsubseteq^{\mathsf{v}} v' \triangleq \mathsf{dom}(v) \subseteq \mathsf{dom}(v') \wedge \forall x \in \mathsf{dom}(v) : v(x) \preccurlyeq v'(x)\,.$$

11.3 Variant semantics

A variant function can always be found by abstraction of the termination semantics into a *variant semantics*

$$\begin{aligned}
\tau^{\mathsf{mv}}[\![\mathrm{P}]\!] &\triangleq \alpha^{\mathsf{v}}(\tau^{\mathsf{mt}}[\![\mathrm{P}]\!]) && \text{potential termination variant}\\
\tau^{\mathsf{Mv}}[\![\mathrm{P}]\!] &\triangleq \alpha^{\mathsf{v}}(\tau^{\mathsf{Mt}}[\![\mathrm{P}]\!]) && \text{definite termination variant.}
\end{aligned}$$

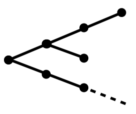
This yields new termination proof methods and static analysis methods by abstraction of this fixpoint definition.

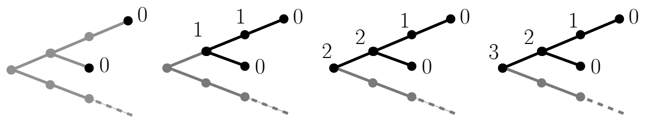
11.4 Fixpoint variant semantics

By fixpoint abstraction of the fixpoint termination trace semantics of Sect. 8.4, we get the fixpoint characterization of the variant semantics[26,27]

$$\begin{aligned}
\tau^{\mathsf{mv}}[\![\mathrm{P}]\!] &= \mathsf{lfp}_{\dot\emptyset}^{\sqsubseteq^{\mathsf{v}}} \overleftarrow{\phi}_{\tau}^{\mathsf{mv}}[\![\mathrm{P}]\!] && \text{potential termination}\\
\overleftarrow{\phi}_{\tau}^{\mathsf{mv}}[\![\mathrm{P}]\!](v)s &\triangleq (\!\!\lbrack\, s \in \beta_\tau[\![\mathrm{P}]\!] \;?\; 0 \;\char"003A\; \mathsf{sup}\left\{v(s') + 1 \;\middle|\right.\\
&\qquad \left. s' \in \mathsf{dom}(v) \wedge \langle s, s'\rangle \in \tau[\![\mathrm{P}]\!]\right\} \rbrack\!\!)\\
\tau^{\mathsf{Mv}}[\![\mathrm{P}]\!] &= \mathsf{lfp}_{\dot\emptyset}^{\sqsubseteq^{\mathsf{v}}} \overleftarrow{\phi}_{\tau}^{\mathsf{Mv}}[\![\mathrm{P}]\!] && \text{definite termination}\\
\overleftarrow{\phi}_{\tau}^{\mathsf{Mv}}[\![\mathrm{P}]\!](v)s &\triangleq (\!\!\lbrack\, s \in \beta_\tau[\![\mathrm{P}]\!] \;?\; 0 \;\char"003A\; \mathsf{sup}\left\{v(s') + 1 \;\middle|\right.\\
&\qquad s' \in \mathsf{dom}(v) \wedge \langle s, s'\rangle \in \tau[\![\mathrm{P}]\!] \wedge\\
&\qquad \left. \forall s'' : \langle s, s''\rangle \in \tau[\![\mathrm{P}]\!] \implies s'' \in \mathsf{dom}(v)\right\} \rbrack\!\!)\,.
\end{aligned}$$

Example 6. Consider the trace semantics as represented on the right. We have represented below the fixpoint iterates for the corresponding potential and definite variant functions. Unlabelled states are outside the variant function domain.





Potential termination

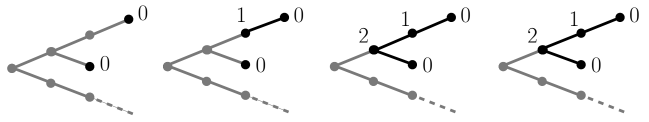
[23] $A \not\mapsto B$ (resp. $A \mapsto B$) is the set of *partial* (resp. *total*) *maps* from set A into set B. We write $\mathsf{dom}(f)$ for the domain of a partial function $f \in A \not\mapsto B$ and $\mathsf{codom}(f)$ for its co-domain. If $f \in A \mapsto B$ then $\mathsf{dom}(f) = A$.

[24] For a proof, take $\langle \mathbb{W}, \prec\rangle$ to be the ordinals $\langle \mathbb{O}, <\rangle$ and v to be the ordinal rank of elements of R for the well-founded relation τ.

[25] This can be generalized from $\langle \mathbb{O}, <\rangle$ to well-orders $\langle \mathbb{W}, \prec\rangle$ using $\mathsf{succ}(x) \triangleq \{y \in \mathbb{W} \mid x < y \wedge \nexists z \in \mathbb{W} : x < z < y\}$ and sup is an upper-bound. For ordinals $\mathsf{succ}(x) = \{x + 1\}$ is the successor ordinal and sup is the lub.

[26] The partial map $\dot\emptyset \in \Sigma \not\mapsto \mathbb{O}$ is totally undefined and has $\mathsf{dom}(\dot\emptyset) \triangleq \emptyset$.

[27] The conditional is $(\!\!\lbrack\, \mathsf{true} \;?\; a \;\char"003A\; b \rbrack\!\!) \triangleq a$ and $(\!\!\lbrack\, \mathsf{false} \;?\; a \;\char"003A\; b \rbrack\!\!) \triangleq b$.



Definite termination

The potential variant can be used as a run-time check of definite non-termination (since beyond 4 execution steps termination is inevitable). This general observation is not in contradiction with the fact that termination is not checkable at runtime since here it relies on a prior static analysis considering all possible executions. □

Example 7. The definite termination variant semantics $\mathsf{lfp}^{\sqsubseteq^{\mathsf{v}}}_{\dot\emptyset}\ \overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!]$ of the following program P

```
int main () { int x; while (x > 0) { x = x - 2; }}
```

is the limit ν^{ω} of the iterates ν^{n}, $n \in \mathbb{N}$ of $\overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!]$ from $\dot\emptyset$. Considering only one loop head control point so that the state can be reduced to the value x of x, we have

$$\overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!](\nu)x \triangleq \llparenthesis\, x \leqslant 0 \mathrel{?} 0 \mathrel{:} \mathsf{sup}\{\,\nu(x-2)+1 \mid x-2 \in \mathsf{dom}(\nu)\}\,\rrparenthesis\ .$$

The program being deterministic, the potential termination equation $\nu = \overleftarrow{\phi}^{\mathsf{mv}}_{\tau}[\![\mathtt{P}]\!](\nu)$ is similar. The fixpoint iterates are[28,29]

$$\begin{array}{lcl}
\nu^{0} & = & \dot\emptyset \\
\nu^{1} & = & \lambda\, x \in [-\infty, 0] \bullet 0 \\
\nu^{2} & = & \lambda\, x \in [-\infty, 0] \bullet 0 \mathbin{\dot\cup} \lambda\, x \in [1, 2] \bullet 1 \\
\nu^{3} & = & \lambda\, x \in [-\infty, 0] \bullet 0 \mathbin{\dot\cup} \lambda\, x \in [1, 2] \bullet 1 \mathbin{\dot\cup} \lambda\, x \in [3, 4] \bullet 2 \\
\ldots & & \\
\nu^{n} & = & \lambda\, x \in [-\infty, 0] \bullet 0 \mathbin{\dot\cup} \lambda\, x \in [1, 2 \times (n-1)] \bullet (x+1) \div 2 \\
\ldots & & \\
\nu^{\omega} & = & \lambda\, x \in [-\infty, 0] \bullet 0 \mathbin{\dot\cup} \lambda\, x \in [1, +\infty] \bullet (x+1) \div 2\ .
\end{array}$$
□

11.5 Termination proof method

The variant semantics is sound and complete to prove termination of a program P for initial states I since

$$\begin{array}{rl}
\alpha^{\mathsf{i}}(\mathsf{I})(\tau^{+\infty}[\![\mathtt{P}]\!]) \subseteq \Sigma^{+}[\![\mathtt{P}]\!] & \Longleftrightarrow\quad \mathsf{I} \subseteq \mathsf{dom}(\tau^{\mathsf{Mv}}[\![\mathtt{P}]\!]) \\
& \Longleftrightarrow\quad \exists \nu \in \Sigma \not\mapsto \mathbb{O} : \mathsf{lfp}^{\sqsubseteq^{\mathsf{v}}}_{\dot\emptyset}\ \overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!] \sqsubseteq^{\mathsf{v}} \nu \wedge \mathsf{I} \subseteq \mathsf{dom}(\nu) \\
\alpha^{\mathsf{i}}(\mathsf{I})(\tau^{+\infty}[\![\mathtt{P}]\!]) \cap \Sigma^{+}[\![\mathtt{P}]\!] \neq \emptyset & \Longleftrightarrow\quad \mathsf{I} \subseteq \mathsf{dom}(\tau^{\mathsf{mv}}[\![\mathtt{P}]\!]) \\
& \Longleftrightarrow\quad \exists \nu \in \Sigma \not\mapsto \mathbb{O} : \mathsf{lfp}^{\sqsubseteq^{\mathsf{v}}}_{\dot\emptyset}\ \overleftarrow{\phi}^{\mathsf{mv}}_{\tau}[\![\mathtt{P}]\!] \sqsubseteq^{\mathsf{v}} \nu \wedge \mathsf{I} \subseteq \mathsf{dom}(\nu)
\end{array}$$

Applying fixpoint induction to check for the least fixpoint over-approximation, we get a termination proof method. We have

$$\begin{array}{l}
\quad \exists \nu \in \Sigma \not\mapsto \mathbb{O} : \tau^{\mathsf{Mv}}[\![\mathtt{P}]\!] \sqsubseteq^{\mathsf{v}} \nu \\
\Leftrightarrow \exists \nu : \mathsf{lfp}^{\sqsubseteq^{\mathsf{v}}}_{\dot\emptyset}\ \overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!] \sqsubseteq^{\mathsf{v}} \nu \qquad \wr\text{fixpoint semantics of Sect. 11.4}\int \\
\Leftrightarrow \exists \nu : \exists \nu' : \dot\emptyset \sqsubseteq^{\mathsf{v}} \nu' \wedge \overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!]\nu' \sqsubseteq^{\mathsf{v}} \nu' \wedge \nu' \sqsubseteq^{\mathsf{v}} \nu \qquad \wr\text{fixpoint ind.}\int \\
\Leftrightarrow \exists \nu' : \overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!]\nu' \sqsubseteq^{\mathsf{v}} \nu' \qquad \wr\text{def. } \sqsubseteq^{\mathsf{v}} \text{ and choosing } \nu = \nu'\int \\
\Leftrightarrow \exists \nu : \lambda\, s \bullet \llparenthesis\, s \in \beta_{\tau}[\![\mathtt{P}]\!] \mathrel{?} 0 \mathrel{:} \mathsf{sup}\{\nu(s') + 1 \mid \exists s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \wedge s' \in \mathsf{dom}(\nu) \wedge \forall s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in \\
\quad \mathsf{dom}(\nu)\}\,\rrparenthesis \sqsubseteq^{\mathsf{v}} \nu \qquad \wr\text{def. } \overleftarrow{\phi}^{\mathsf{Mv}}_{\tau}[\![\mathtt{P}]\!]\int \\
\Leftrightarrow \exists \nu : \lambda\, s \bullet \mathsf{sup}\{\nu(s') + 1 \mid \exists s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \wedge s' \in \mathsf{dom}(\nu) \wedge \forall s' : \\
\quad \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in \mathsf{dom}(\nu)\} \sqsubseteq^{\mathsf{v}} \nu \\
\quad \wr\text{since } \forall s : \nu(s') \geqslant 0 \text{ and } \exists s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \text{ implies } s \notin \beta_{\tau}[\![\mathtt{P}]\!]\int \\
\Leftrightarrow \exists \nu : \mathsf{dom}(\lambda\, s \bullet \mathsf{sup}\{\nu(s') + 1 \mid \exists s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \wedge s' \in \\
\quad \mathsf{dom}(\nu) \wedge \forall s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in \mathsf{dom}(\nu)\}) \subseteq \\
\quad \mathsf{dom}(\nu) \wedge \forall s \in \mathsf{dom}(\nu) : \mathsf{sup}\{\nu(s') + 1 \mid \exists s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \wedge s' \in \\
\quad \mathsf{dom}(\nu) \wedge \forall s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in \mathsf{dom}(\nu)\} \leqslant \nu(s) \\
\qquad \wr\text{def. } \sqsubseteq^{\mathsf{v}} \text{ for ordinals}\int \\
\Leftrightarrow \exists \nu : \forall s \in \mathsf{dom}(\nu) : \Big(\exists s' \in \mathsf{dom}(\nu) : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!]\Big) \Longrightarrow \Big(\forall s' : \\
\quad \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in \mathsf{dom}(\nu) \wedge \nu(s') < \nu(s)\Big) \qquad \wr\text{def. }\mathsf{sup}\int \\
\Leftrightarrow \exists \langle \mathbb{W},\ \prec\rangle : \exists \nu \in \Sigma \not\mapsto \mathbb{W} : \forall s \in \mathsf{dom}(\nu) : \Big(\exists s' \in \mathsf{dom}(\nu) : \langle s, \\
\quad s'\rangle \in \tau[\![\mathtt{P}]\!]\Big) \Longrightarrow \Big(\forall s' : \langle s, s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in \mathsf{dom}(\nu) \wedge \nu(s') \prec \\
\quad \nu(s)\Big) \quad \wr\text{since an ordinal is the order type of a well-founded set}\int \\
\Leftrightarrow \exists I \in \wp(\Sigma) : \exists \langle \mathbb{W},\ \prec\rangle : \exists \nu \in \Sigma \not\mapsto \mathbb{W} : \forall s \in I : \Big(\exists s' \in I : \langle s, \\
\quad s'\rangle \in \tau[\![\mathtt{P}]\!]\Big) \Longrightarrow \Big(\forall s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in I \wedge \nu(s') \prec \nu(s)\Big) \\
\qquad \wr\text{choosing } I = \mathsf{dom}(\nu).\int
\end{array}$$

This calculational design yields the following *definite termination induction principle*

$$\begin{array}{ll}
& \alpha^{\mathsf{i}}(\mathsf{I})(\tau^{+\infty}[\![\mathtt{P}]\!]) \subseteq \Sigma^{+}[\![\mathtt{P}]\!] \qquad\qquad \text{definite termination proof} \\
\Longleftrightarrow & \exists I \in \wp(\Sigma) : \exists \langle \mathbb{W},\ \prec\rangle : \exists \nu \in \Sigma \not\mapsto \mathbb{W} : \mathsf{I} \subseteq \mathsf{dom}(\nu) \wedge \forall s \in I : \\
& \quad \Big(\exists s' \in I : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!]\Big) \Longrightarrow \\
& \qquad \Big(\forall s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \Longrightarrow s' \in I \wedge \nu(s') \prec \nu(s)\Big)\ .
\end{array}$$

A similar calculational design, yields the *potential termination induction principle*

$$\begin{array}{ll}
& \alpha^{\mathsf{i}}(\mathsf{I})(\tau^{+\infty}[\![\mathtt{P}]\!]) \cap \Sigma^{+}[\![\mathtt{P}]\!] \neq \emptyset \qquad \text{potential termination proof} \\
\Longleftrightarrow & \exists I \in \wp(\Sigma) : \exists \langle \mathbb{W},\ \prec\rangle : \exists \nu \in \Sigma \not\mapsto \mathbb{W} : \mathsf{I} \subseteq \mathsf{dom}(\nu) \wedge \forall s \in I : \\
& \quad (\exists s' \in I : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!]) \Longrightarrow \\
& \qquad (\exists s'' \in I : \langle s,\ s''\rangle \in \tau[\![\mathtt{P}]\!] \wedge s'' \in I \wedge \nu(s'') \prec \nu(s))\ .
\end{array}$$

Observe that the fixpoint variant semantics of Sect. 11.4 is calculated backwards (the variant function increases on previous steps) but that the termination induction principles proceed forward (the variant function decreases on next steps).

Example 8. A similar induction principle is proposed in [35, Ch. 5.2.3] for relational inevitability proofs (a state must be reached that relates to the initial state as given by a specification relation Ψ). The following example is used in [35, Ch. 5.2.5] to show that, the invariant and variant function must also be relational, that is relate the current and initial state: $\Sigma \triangleq \{1, 2, 3\}$, $\mathsf{I} \triangleq \{1, 2\}$, $\tau \triangleq \{\langle x,\ x+1\rangle \mid x, x+1 \in \Sigma\}$, $\Psi \triangleq \tau$. We can prove termination with assertions, no relational invariants being needed. For the above example, choose $I = \Sigma$, $\langle \mathbb{W},\ \prec\rangle = \langle \Sigma,\ <\rangle$, $\nu(1) = 2$, $\nu(2) = 1$, $\nu(3) = 0$. This example shows that termination proofs are simpler than inevitability proofs. □

Example 9. For the program of Ex. 7, the definite termination proof for the simplified transition system

$$\tau[\![\mathtt{P}]\!] \quad \triangleq \quad \{\langle x,\ x'\rangle \mid x > 0 \wedge x' = x + 1\}$$

requires guessing $I = \mathbb{Z}$, $\langle \mathbb{W},\ \prec\rangle = \langle \mathbb{N},\ <\rangle$, $\nu = \lambda\, x \bullet \llparenthesis\, x \leqslant 0 \mathrel{?} 0 \mathrel{:} (x+1) \div 2\,\rrparenthesis$ and proving $\forall x, x' \in \mathbb{Z} : (x > 0 \wedge x' = x + 1) \Longrightarrow (\forall x'' : x'' = x + 1 \Longrightarrow \nu(x'') < \nu(x))$. □

Because Turing/Floyd method uses the reachability abstraction α^{r} of (4), it is not possible to directly relate states occurring at different times during computations. This is why the program is transformed by using auxiliary variables to relate the current values of the variables to their past values. This induces a transformed transition system, which under the reachability abstraction α^{r} is equivalent to the relational abstraction of the original transition system by the relational abstraction (1).

Example 10. Continuing Ex. 9, the program is transformed into

```
int main () { int x, x0;
    while (x > 0) { x0 = x; x = x - 2; }}
```

which consists in reasoning on the transformed transition system

[28] $\dot\cup$ joins partial functions with disjoint domains $f_1 \mathbin{\dot\cup} f_2(x) \triangleq f_1(x)$ if $x \in \mathsf{dom}(f_1)$ and $f_1 \mathbin{\dot\cup} f_2(x) \triangleq f_2(x)$ if $x \in \mathsf{dom}(f_2)$ where $\mathsf{dom}(f_1) \cap \mathsf{dom}(f_2) = \emptyset$.

[29] $\div$ is the integer division.

$$\tau_0[\![\mathrm{P}]\!] \quad \triangleq \quad \{\langle\langle x_0,\ x\rangle,\ \langle x_0',\ x'\rangle\rangle \mid x_0' = x \wedge \langle x,\ x'\rangle \in \tau[\![\mathrm{P}]\!]\}\,. \qquad \square$$

This is an abstraction $\langle \wp(\Sigma\times\Sigma),\ \subseteq\rangle \underset{\alpha^0}{\overset{\gamma^0}{\longleftrightarrow}} \langle\wp(\Sigma^2\times\Sigma^2),\ \subseteq\rangle$ such that

$$\alpha^0(\tau) \quad \triangleq \quad \{\langle\langle x_0,\ x\rangle,\ \langle x_0',\ x'\rangle\rangle \mid x_0' = x \wedge \langle x,\ x'\rangle \in \tau\}\,.$$

The benefit is that a relational abstraction α^{R} used with τ is equivalent to a non-relational reachability abstraction α^r for $\alpha^0(\tau)$. However, in both cases, a limitation is that, for a given control point, it is only possible to refer to one past instant of time when control is at that program point, which is a limitation when compared to the more flexible reasoning by induction on traces (see Sect. 15.3).

12. Variant abstraction analysis

We get a termination static analysis by abstracting the variant semantics. We need an abstraction $\langle \Sigma \not\mapsto \mathbb{O},\ \sqsubseteq^{\mathsf{v}}\rangle \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} \langle A,\ \sqsubseteq\rangle$ of functions. Many abstractions of functions have been proposed e.g. [20, 30] that can be reused for termination static analysis. As a simple example, we consider a piecewise linear variant abstraction. The purpose of this new abstract domain is to illustrate the abstraction of fixpoint definitions of variant functions with widening, many more abstractions being necessary to cover all practical cases.

12.1 Piecewise linear variant abstraction

Let us consider a program with integer variables $\mathbb{X} = \mathtt{x}_1, \ldots, \mathtt{x}_n$, $n > 0$. We first apply an abstraction of states extracting the numerical variables in the form of an environment $\alpha_{\mathbb{X}} \in \Sigma \mapsto (\mathbb{X} \mapsto \mathbb{Z})$ so that, by composition, we are left with an abstraction $\langle(\mathbb{X} \mapsto \mathbb{Z}) \not\mapsto \mathbb{O},\ \sqsubseteq^{\mathsf{v}}\rangle \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} \langle A,\ \sqsubseteq\rangle$. Encoding the partial map by a total map (using "$\perp$" for "undefined/not in the domain" and abstracting higher-order ordinals by "$\top$" ("unknown/infinite", e.g. in case of non-termination or unbounded nondeterminism), we can choose $(\mathbb{X} \mapsto \mathbb{Z}) \mapsto \mathbb{N} \cup \{\perp, \top\}$. There is no loss of information for bounded determinism and unbounded executions are still allowed but disregarded by the abstraction. We can now further abstract by piecewise linear functions.

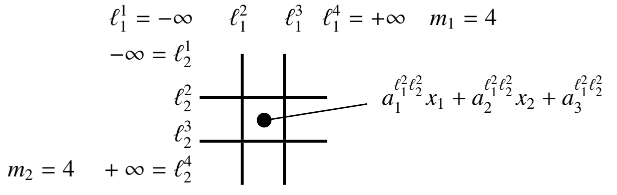
The values x_i of each variable $\mathtt{x}_i \in \mathbb{X}$, $i \in [1, n]$ are segmented into $\ell_i^1 = -\infty < \cdots < \ell_i^{j_i} < \cdots < \ell_i^{m_i} = +\infty$. This provides a partition of the space $\mathbb{Z}^n$ of values $x_1, \ldots, x_n$ of the variables $\mathtt{x}_1, \ldots, \mathtt{x}_n$. The blocks of the partition are therefore $[\ell_i^{j_i}, \ell_i^{j_i+1})$, $i \in [1, n]$, $j_i \in [1, m_i)$. In practice machine integers are bounded, in which case $-\infty$ and $+\infty$ are the smallest and largest machine integers. The number of blocks in the partitions can also be limited by widening thus favoring efficiency of the abstract domain to the detriment of precision.

12.1.1 The abstract domain of piecewise linear variants

The positive value of the variant function for elements $\vec{x} = x_1, \ldots, x_n$ of each block $[\ell_i^{j_i}, \ell_i^{j_i+1})$ of the partition is a linear expression $\vec{a}^{\,\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}}.\vec{x}$ of the form [30]

$$a_1^{\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}}\,x_1 + \ldots + a_i^{\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}}\,x_i + \ldots + a_n^{\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}}\,x_n + a_{n+1}^{\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}}$$

where the coefficients $a_k^{\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}} \in \mathbb{Q}$, $k \in [1, n+1]$ are rationals (or $\perp/\top$). For example, in two dimensions



[30] More rigorously, we should write the dot product $\vec{a}^{\,\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}} \cdot (\vec{x}, 1)$.

The abstract domain is therefore (omitting the case of blocks with $\perp$ for "not in the domain" and $\top$ for "unknown")

$$\begin{aligned} A \quad \triangleq \quad & \Big\{\lambda\vec{x} \in \mathbb{Z}^n \bullet \prod_{\substack{i\in[1,n],\\ j_i\in[1,m_i)}} [\![\, \ell_i^{j_i} \leqslant x_i < \ell_i^{j_i+1} \ ?\ \vec{a}_1^{\,\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}}.\vec{x} \ \text{⁏}\ \perp]\!] \\ & \ \Big|\ \forall i \in [1, n] : \ell_i^1 = -\infty < \cdots < \ell_i^{j_i} < \cdots < \ell_i^{m_i} = +\infty \wedge \\ & \quad \vec{a}^{\,\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}} \in \mathbb{Q}^{n+1} \wedge \\ & \quad\quad \forall j_i \in [1, m_i), x_i \in [\ell_i^{j_i}, \ell_i^{j_i+1}) :\ \vec{a}_1^{\,\ell_1^{j_1}\ldots\ell_i^{j_i}\ldots\ell_n^{j_n}}.\vec{x} \geqslant 0 \Big\} \quad . \end{aligned}$$

When the $\ell_i^{j_i} \in \mathbb{Q}$, $i \in [1, n]$, $j_i \in [1, m_i)$ are rationals, this abstraction essentially reuses the classical abstractions of intervals [18, 19], linear inequalities [31] and segmentation [33]. An immediate generalization consists in using consecutive segments with symbolic bounds as done in [33] for array content analysis. A further generalization consists in using decision trees [32] instead of a segmentation of the domain of the abstract variant function.

12.1.2 Piecewise linear variant abstract transformers

The abstract transformer $\overleftarrow{\phi}{}^{\sharp\,\mathsf{mv}}_{\tau}[\![\mathrm{P}]\!]$ abstracting the concrete transformer $\overleftarrow{\phi}{}^{\mathsf{mv}}_{\tau}[\![\mathrm{P}]\!]$ of Sect. 11.4 is applied blockwise by computing the abstract pre-image of each block by assignments or tests. The condition in tests may split the block into sub-blocks for which the condition is true or false.

Example 11. Here is an example of first iteration of the backward termination analysis of an exit preceded by a test. The initialization of the fixpoint iterates by $\lambda x \in [-\infty, +\infty] \bullet \perp$ indicates potential non-termination. The exit enforces termination in 0 steps. The test splits the block $[-\infty, +\infty]$ into $[-\infty, 0]$ and $[1, +\infty]$.

```
/* λx•⟦ x ∈ [−∞, 0] ? 0 ⁏ x ∈ [1, +∞] ? ⊥ ⟧ */
if (x <= 0) { /* λx∈[−∞, +∞]•0 */
   exit; /* λx∈[−∞, +∞]•⊥ */ }
else { /* λx∈[−∞, +∞]•⊥ */ ... }
```
$\square$

An assignment backward propagates the linear variant functions by blocks which are incremented by 1 step, but for those corresponding to non-termination.

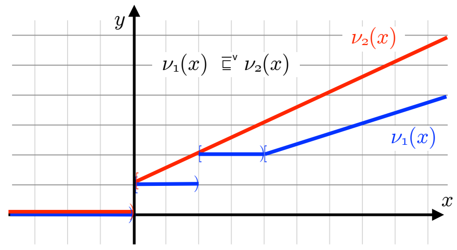
Example 12. Assuming $-\infty - 2 = -\infty$ and $+\infty + 2 = +\infty$, the backward termination analysis of the following assignment is

```
/* λx•⟦ x ∈ [−∞, 2] ? 1 ⁏ x ∈ [3, +∞] ? ⊥ ⟧ */
x = x - 2;
/* λx•⟦ x ∈ [−∞, 0] ? 0 ⁏ x ∈ [1, +∞] ? ⊥ ⟧ */
```
$\square$

12.1.3 Piecewise linear variant abstract order

The abstract order $\overline{\sqsubseteq}^{\mathsf{v}}$ first unifies segments of the domain into a common refined partition by segmentation of each variable (as in [33, Sect. 11.4: Segmentation unification]) and then compares the linear expressions blockwise, assuming $\perp$ is the infimum and $\top$ is the supremum (so that the domain comparison of Sect. 9 is done implicitly by the fact that the "undefined" $\perp$ is used outside this domain).

Example 13.



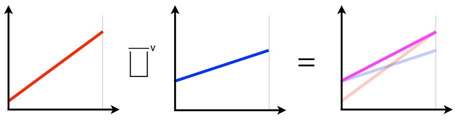
$\square$

12.1.4 Piecewise linear variant abstract join

Similarly, the join $\nu_1 \mathrel{\dot{\sqcup}^{\mathsf{v}}} \nu_2$ first unifies blocks of the partitioned domains of ν_1 and ν_2 into a common refined partition. Then the linear expressions are joined blockwise. This blockwise join $\overline{\sqcup}^{\mathsf{v}}$ is $\vec{a}.\vec{x}$ defined for each block $\ell_1^{j_1} \ldots \ell_i^{j_i} \ldots \ell_n^{j_n}$, $i \in [1, n]$, $j_i \in [1, m_i]$ of the partition such that $\forall i \in [1, n]$, $\forall x_i \in [\ell_i^{j_i}, \ell_i^{j_i+1}]$, $\forall \vec{a}\,' \in \mathbb{Q}^{n+1}$,

- $\vec{a}^{\,\ell_1^{j_1} \ldots \ell_i^{j_i} \ldots \ell_n^{j_n}}.\vec{x} \leqslant \vec{a}.\vec{x}$
- $\vec{a}^{\,\ell_1^{j_1} \ldots \ell_i^{j_i} \ldots \ell_n^{j_n}}.\vec{x} \leqslant \vec{a}\,'.\vec{x} \implies \vec{a}.\vec{x} \leqslant \vec{a}\,'.\vec{x}$.

Example 14.



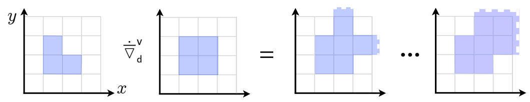
□

A coarser partition can also be used in the join (as in [33, Sect. 11.4: Segmentation unification]) which is less precise but enforces faster convergence.

12.1.5 Piecewise linear variant abstract widening

Finally, the widening $\nu_1 \mathrel{\dot{\overline{\triangledown}}^{\mathsf{v}}} \nu_2$ follows the idea introduced in [20] of widening functions by widening the domain of their parameters with a domain widening $\dot{\overline{\triangledown}}^{\mathsf{v}}_{\mathsf{d}}$ and then their results with a range widening $\dot{\overline{\triangledown}}^{\mathsf{v}}_{\mathsf{r}}$. So the blocks of the partitioned domains of ν_1 and ν_2 are first widened using e.g. interval widening $\dot{\overline{\triangledown}}^{\mathsf{v}}_{\mathsf{d}}$ (possibly with thresholds) of the blocks with respect to their neighbors in all directions.

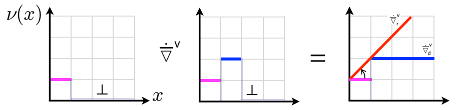
Example 15. An interval widening for a two-dimensional domain $\langle x,\ y\rangle \in \mathbb{Z}^2$ yields



□

Then the range-widening $\dot{\overline{\triangledown}}^{\mathsf{v}}_{\mathsf{r}}$ increases the gradient (i.e. slope in two dimensions) of the variant function of each block in the directions of its domain-widened neighbors to over-approximate their respective variants functions (extended to the widened domains).

Example 16.



$\nu(x)$
$\perp$
x
$\dot{\overline{\triangledown}}^{\mathsf{v}}$
$\perp$
=
$\dot{\overline{\triangledown}}^{\mathsf{v}}_{\mathsf{r}}$
$\dot{\overline{\triangledown}}^{\mathsf{v}}_{\mathsf{d}}$

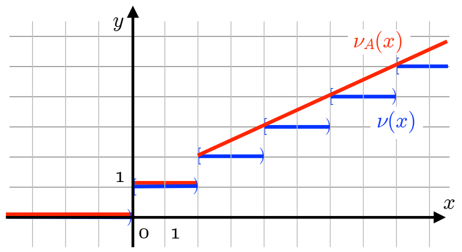
□

To enforce convergence, the widening may have to skip to finitely many given thresholds of gradients before abandoning the constraint to $\top$.

Example 17. We use two loop unrollings to stabilize iterations before widening [56].

$$\begin{aligned}
\nu_A^0 &= \boldsymbol{\lambda}\, x \in [-\infty, +\infty] \bullet \perp \\
\nu_A^1 &= \boldsymbol{\lambda}\, x \bullet \llbracket\, x \in [-\infty, 0] \mathrel{?} 0 \mathrel{⁏} x \in [1, +\infty] \mathrel{?} \perp \rrbracket \\
\nu_A^2 &= \boldsymbol{\lambda}\, x \in [-\infty, 0] \bullet 0 \mathbin{\dot{\cup}} \boldsymbol{\lambda}\, x \in [1, 2] \bullet 1 \mathbin{\dot{\cup}} \boldsymbol{\lambda}\, x \in [3, +\infty] \bullet \perp \\
\nu'^{\,3}_A &= \boldsymbol{\lambda}\, x \bullet \llbracket\, x \in [-\infty, 0] \mathrel{?} 0 \mathrel{⁏} x \in [1, 2] \mathrel{?} 1 \mathrel{⁏} x \in [3, 4] \mathrel{?} 2 \\
&\qquad\qquad \mathrel{⁏} x \in [5, +\infty] \mathrel{?} \perp \rrbracket \\
\nu_A^3 &= \nu_A^2 \mathrel{\dot{\overline{\triangledown}}^{\mathsf{v}}} \nu'^{\,3}_A \\
\nu'^{\,4}_A &= \boldsymbol{\lambda}\, x \bullet \llbracket\, x \in [-\infty, 0] \mathrel{?} 0 \mathrel{⁏} x \in [1, 2] \mathrel{?} 1 \mathrel{⁏} x \in [3, +\infty] \mathrel{?} \frac{x}{2} + 1 \rrbracket \\
\nu_A^4 &= \nu_A^3 \ .
\end{aligned}$$

The over-approximation of ν in Ex. 7, by ν_A is as follows



. □

Notice that the domain of termination is widened which is an over-approximation which might include non-termination cases. However, the iterates with widening stop at a post-fixpoint ν_A

$$\overleftarrow{\phi}^{\sharp\,\mathsf{mv}}_{\tau}\llbracket \mathtt{P} \rrbracket(\nu_A) \mathrel{\dot{\sqsubseteq}^{\mathsf{v}}} \nu_A$$

which, by definition of the abstract partial order $\dot{\sqsubseteq}^{\mathsf{v}}$ ensures that ν_A is decreasing on blocks for which it is defined. Termination is therefore proven for blocks with either 0 or a strictly decreasing variant. By undecidability, there might be blocks which variant value is $\top$ indicating insufficient precision to conclude.

12.2 Non-linear variant abstraction

Besides classical linear relational abstractions (e.g. octagons [46], polyhedra [31], etc.) which can be used pointwise as in Sect. 12.1, the variant function in each block of the partition can also be non-linear (e.g. polynomials [47], exponentials [39], etc.).

13. Relational variant semantics

To use relational abstractions for static termination analysis, we can further abstract variant functions into relations.

13.1 Relational variant abstraction

A variant function ν can be abstracted as the pair of an abstraction of its domain $\mathsf{dom}(\nu)$ by a set abstraction (such as e.g. intervals) and an abstraction of its value by (a relational abstraction of) the down-closed relation r which over-approximates the variant function on its domain that is $\forall s \in \mathsf{dom}(\nu), w \in \Sigma : \langle s,\ w\rangle \in r \implies w \preccurlyeq \nu(s)$. The abstraction is therefore (the first component is redundant but useful for static analysis)

$$\alpha^{\mathsf{rv}}(\nu) \ \triangleq\ \langle \mathsf{dom}(\nu),\ \alpha^{\downarrow}(\{\langle s,\ \nu(s)\rangle \mid s \in \mathsf{dom}(\nu)\})\rangle$$

where the *down-closure* of a relation $r \in \wp(\Sigma \times \mathbb{W})$ is

$$\alpha^{\downarrow}(r) \ \triangleq\ \{\langle s,\ w'\rangle \mid \exists w : w' \preccurlyeq w \wedge \langle s,\ w\rangle \in r\} \ .$$

Observe that the effect of the down-closure is to replace equalities by inequalities for which numerous abstract domains are available. Moreover, an over-approximation of the first component is known by Sect. 9 but for correction we either need an under-approximation or prove termination for this over-approximation, which is the usual option. For the second component, an over-approximation is correct (this over-estimates the termination time). We have[31]

$$\langle \Sigma \not\mapsto \mathbb{W},\ \sqsubseteq^{\mathsf{v}}\rangle \xrightleftharpoons[\alpha^{\mathsf{v}}]{\gamma^{\mathsf{v}}} \langle \wp(\Sigma) \times \alpha^{\downarrow}[\wp(\Sigma \times \mathbb{W})],\ \subseteq \times \subseteq\rangle \ .$$

13.2 Relational variant semantics

The *relational variant semantics* of a program $\mathtt{P}$ is

$\tau^{\mathsf{mrv}}\llbracket \mathtt{P} \rrbracket$	$\triangleq$	$\alpha^{\mathsf{rv}}(\tau^{\mathsf{mv}}\llbracket \mathtt{P} \rrbracket)$	potential termination relational variant
$\tau^{\mathsf{Mrv}}\llbracket \mathtt{P} \rrbracket$	$\triangleq$	$\alpha^{\mathsf{rv}}(\tau^{\mathsf{Mt}}\llbracket \mathtt{P} \rrbracket)$	definite termination relational variant.

[31] $\leqslant \times \sqsubseteq$ is the componentwise partial order $\langle x,\ y\rangle \leqslant \times \sqsubseteq \langle x',\ y'\rangle \iff x \leqslant x' \wedge y \sqsubseteq y'$.

13.3 Fixpoint relational variant semantics

By fixpoint abstraction of the fixpoint variant semantics of Sect. 11.4, we get, by calculational design, the fixpoint definite and potential relational variant semantics[32].

$$\begin{array}{rcll}
\tau^{\mathsf{mrv}}[\![\mathtt{P}]\!] & = & \mathsf{lfp}^{\subseteq\times\subseteq}_{\dot{\emptyset}}\ \overleftarrow{\phi}^{\,\mathsf{mrv}}_{\tau}[\![\mathtt{P}]\!] & \text{potential termination}
\end{array}$$

$$\begin{array}{l}
\overleftarrow{\phi}^{\,\mathsf{mrv}}_{\tau}[\![\mathtt{P}]\!]\langle D,\ r\rangle \ \triangleq\ \text{let } D' = D \cup \beta_\tau[\![\mathtt{P}]\!] \cup \tau[\![\mathtt{P}]\!]^{-1}[D] \text{ in}\\
\quad \langle D',\ \{\langle s,\ 0\rangle \mid s \in \beta_\tau[\![\mathtt{P}]\!]\} \cup \alpha^{\downarrow}(\{\langle s,\ \rho+1\rangle \mid\\
\qquad \wedge\ \exists s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \wedge s' \in D \wedge \langle s',\ \rho\rangle \in r\})\rangle
\end{array}$$

$$\begin{array}{rcll}
\tau^{\mathsf{Mrv}}[\![\mathtt{P}]\!] & = & \mathsf{lfp}^{\subseteq\times\subseteq}_{\dot{\emptyset}}\ \overleftarrow{\phi}^{\,\mathsf{Mrv}}_{\tau}[\![\mathtt{P}]\!] & \text{definite termination}
\end{array}$$

$$\begin{array}{l}
\overleftarrow{\phi}^{\,\mathsf{Mrv}}_{\tau}[\![\mathtt{P}]\!](\langle D,\ r\rangle)s \ \triangleq\ \text{let } D' = D \cup \beta_\tau[\![\mathtt{P}]\!] \cup (\tau[\![\mathtt{P}]\!]^{-1}[D] \cap \tau[\![\mathtt{P}]\!]^{\widetilde{-1}}[D])\\
\quad \text{in } \langle D',\ \{\langle s,\ 0\rangle \mid s \in \beta_\tau[\![\mathtt{P}]\!]\} \cup \{\langle s,\ \rho+1\rangle \mid\\
\qquad \wedge\ \exists s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \wedge s' \in D \wedge \langle s',\ \rho\rangle \in r\\
\qquad\quad \wedge\ \forall s' : \langle s,\ s'\rangle \in \tau[\![\mathtt{P}]\!] \implies s' \in D \wedge \exists \rho' : \langle s',\ \rho'\rangle \in r\}\rangle\ .
\end{array}$$

The over-approximation of D is classical in static analysis [19, 21] so we concentrate on the over-approximation of the relational variant r.

14. Transition-based termination analysis

We consider the case when states $s \in \Sigma$ consist of a pair $\langle \xi,\ \mu\rangle$ of a control state ξ (used for state or trace partitioning) and a memory state μ. The memory state maps variables $\mathtt{x} \in \mathbb{X}$ to numerical values $\mu(\mathtt{x}) \in \mathbb{Z}$ (for simplicity all other types are ignored in the examples). We consider a relational abstraction $\langle \alpha^{\downarrow}[\wp(\Sigma \times \mathbb{W})],\ \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\longleftrightarrow}} \langle A,\ \sqsubseteq\rangle$ of the fixpoint relational variant semantics of Sect. 13.2. In practice, we choose $\mathbb{W} = \mathbb{N}$ and adjoint an extra variable # to contain the value of ρ.

We can use octagons [46], polyhedra [31], polynomials [47], exponentials [39], their numerous variants, possibly partitioned on states [12], traces [56], or conditions of decision trees [32].

Example 18. Consider the program of Ex. 7, where a forward interval analysis has determined the invariants given as comments.

```
int main () { int x;
    /* x:[-2147483648, 2147483647] */
    while (x > 0) {
       /* x:[1, 2147483647] */
       x = x - 2;
       /* x:[-1, 2147483645] */
    }
    /* x:[-2147483648, 0] */
}
```

The abstraction of the fixpoint equations of Sect. 13.3 is given below in logical form (representing a set by its characteristic predicate) with restriction to the reachable states over-approximated by the interval analysis.

$$\begin{array}{l}
r(x,\#) \ \Longleftarrow\ (-2147483648 \leqslant x \leqslant \# = 0) \vee (\exists x', \#' :\\
\qquad x \in [1, 2147483647] \wedge x' = x - 2 \wedge \# \leqslant \#' + 1 \wedge r(x', \#'))\ .
\end{array}$$

Inverting the assignment yields the classical simplification

$$\begin{array}{l}
r(x,\#) \ \Longleftarrow\ (-2147483648 \leqslant x \leqslant \# = 0) \vee (\exists \#' :\\
\qquad x \in [1, 2147483647] \wedge \# \leqslant \#' + 1 \wedge r(x-2, \#'))\ .
\end{array}$$

Partitioning into $r_1(x,\#) = r(x,\#) \wedge x \leqslant 0$ and $r_2(x,\#) = r(x,\#) \wedge x > 0$, the iterates for $r_1(x,\#)$ immediately converge while the iterates for $r_2(x,\#)$ abstracted with octagons [46] are

$$\begin{array}{rcll}
r_2^0(x,\#) & = & \mathsf{false} & \\
r_2^1(x,\#) & = & \exists \#' : x \in [1, 2147483647] \wedge \# \leqslant \#' + 1 \wedge r_1(x-2, \#') & \\
& = & x = 1 \wedge \# \leqslant 1 & \\
r_2^2(x,\#) & = & \exists \#' : x \in [1, 2147483647] \wedge \# \leqslant \#' + 1 \wedge & \\
& & \qquad (r_1(x-2,\#') \vee r_2^1(x-2,\#')) & \\
& = & (x = 1 \wedge \# \leqslant 1) \vee (x = 3 \wedge \# \leqslant 2) & \\
& = & 1 \leqslant x \leqslant \# \leqslant 2 & \text{octagon abstraction of } \vee\\
r_2^3(x,\#) & = & \exists \#' : x \in [1, 2147483647] \wedge \# \leqslant \#' + 1 \wedge & \\
& & \qquad (r_1(x-2,\#') \vee r_2^2(x-2,\#')) & \\
& = & (x = 1 \wedge \# \leqslant 1) \vee (2 \leqslant x \leqslant \# + 1 \wedge \# \leqslant 3) & \\
& = & 1 \leqslant x \leqslant \# \leqslant 3 & \text{octagon abstraction of } \vee\\
& = & 1 \leqslant x \leqslant \# \wedge x \leqslant 2147483647 & \text{widening with } r_2^2(x,\#)\\
r_2^4(x,\#) & = & r_2^3(x,\#) &
\end{array}$$

proving termination since # strictly decreases around the loop and remains positive. Of course direct resolution methods [17, 54] would find the same result. However tests are excluded within loops in [54] while the presence of tests is not impairing the above octagon abstraction or the piecewise linear variant abstraction of Sect. 12.1. For example, the loop body `if (odd (x)) { x = x - 1; } else { x = x - 2 }` with state partitioning on the conditional branches yields the same results. ☐

15. Semantic structural induction

Semantic structural induction is by induction on the structure of computations as opposed to transitional verification based on an induction on the program steps as in Floyd/Turing method [40, 59]. This point of view generalizes syntactic structural induction on program syntax as in Hoare logic [43], replacing the syntactic by a semantic point of view using the concept of structural inductive cover. We start by the simple case of structuring states in next Sect. 15.1 before generalizing to the more concrete trace computations in Sect. 15.3 and their abstractions in Sect. 16.

15.1 Inductive state cover

Many inductive formal definitions and verification methods can be formalized in a language-independent way by an inductive cover of the set Σ of states (examples are given in next Sect. 15.2).

Definition 1. An *inductive state cover* of a non-empty set $\chi \in \wp(\Sigma)$ of states is tree encoded as a set $C \in \mathfrak{C}(\chi)$ of (finite) sequences S of non-empty members $B \in \wp(\chi) \setminus \{\emptyset\}$ such that

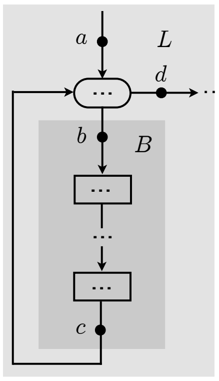
1. if $S\,S' \in C$ then $S \in C$ (prefix-closure)
2. if $S \in C$ then $\exists S' : S = \chi S'$ (root)
3. if $S\,BB' \in C$ then $B \supset B'$ (well-foundedness)
4. if $S\,BB' \in C$ then $B \subseteq \bigcup\limits_{S\,BB' \in C} B'$ (cover). ☐

By the prefix-closure condition Def. 1.1, the inductive cover is a tree (so that proofs based on the cover C are by case analysis on the tree width and induction on the tree depth). By the root condition Def. 1.2, the tree is rooted at χ (which ensures that inductive proofs based on the cover C are valid for χ). By the strictly-decreasing condition Def. 1.3, the sequences S are necessarily finite so the immediate component relation between a node of the tree and its sons is well-founded. It follows that proofs on states can be done by induction on this well-founded relation. And, by the covering condition Def. 1.4, the states in a node are covered by the join of the states in its sons (which ensures that proofs based on the cover C do not forget any possible case). Inductive state covers are abstractions of inductive trace covers introduced in forthcoming Sect. 15.3 but are introduced first for simplicity. An example is [45].

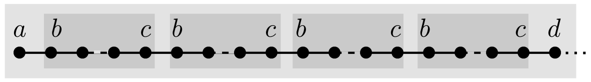
[32] The dual pre-image of $Y \in \wp(A)$ by a relation $r \in \wp(A \times B)$ is $r^{\widetilde{-1}}[Y] \triangleq \neg r^{-1}[\neg Y]$ also written $\widetilde{\mathbf{pre}}[r]Y$.

15.2 Examples of semantic structural induction

15.2.1 Loop invariants and variants



In Floyd's total correctness proof method, one typically provides a loop invariant and a loop variant function for termination. It is not necessary for the variant function to strictly decrease at each program step but only once around each loop iterate. This corresponds to a cover of the states of the loop according to their control component which induces a decomposition of executions into trace segments for the loop containing trace segments for the loop body considered as one step in the inductive reasoning on loop iterations.

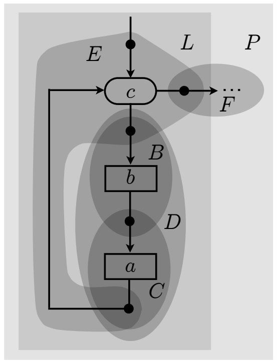


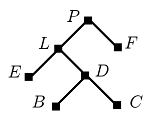
Moreover a different variant function is used for each loop so that this decomposition is applied recursively for nested loops.

15.2.2 Hoare logic

Inductive definition/verification in the form of structural induction on the program syntax originates from axiomatic semantics [43], denotational semantics [57], and operational semantics [51].

Hoare logic for a structured imperative language [43], and its extension to total correctness [44], can be understood as the inductive state cover based on the control states of a command (ignoring its memory states). For example, a while loop can be covered by the states which control is in the condition and the states which control is in the loop body. The states of the loop body can themselves be covered recursively, by structural induction on the program syntax. This structural induction on the program syntax can be understood as induction on a state cover which itself induces a cover of the execution traces by segments which states are in a block of the state cover. A termination proof by structural induction on the program syntax [44] has the advantage, a.o., to be able to handle unbounded non-determinism without requiring transfinite ordinals (equivalent to a lexicographic ordering on nested loops).

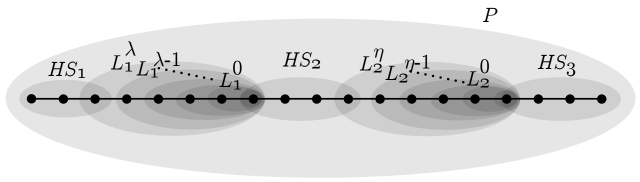


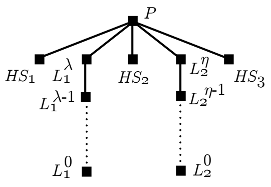


$\{P, PF, PL, PLE, PLD, PLDB, PLDC\}$

15.2.3 Burstall intermittent assertion proof method

Burstall's total correctness proof method [3, 29] can be understood as an inductive reasoning by recurrence on data (as well as control as in Floyd/Turing and Hoare's methods). Although Burstall's proof method [3] is equivalent in power to Floyd/Turing's method [25], it is much easier to use in practice. The formalization of Burstall's total correctness proof method [3] in [25] can be understood as a tree cover on both control and data. The example below shows how hand-simulation/symbolic execution (HS) and lemmas (L_1, L_2) apply to a particular execution trace.





The inductive cover contains the program P, the hand-simulation/symbolic execution blocks $P\ HS_1$, $P\ HS_2$, $P\ HS_3$, and two lemmas with respective blocks $P\ L_1^\lambda$, $P\ L_1^\lambda\ L_1^{\lambda-1}$, …, $P\ L_1^\lambda\ L_1^{\lambda-1}\ \cdots\ L_1^0$ and $P\ L_2^\eta$, $P\ L_2^\eta\ L_2^{\eta-1}$, …, $P\ L_2^\eta\ L_2^{\eta-1}\ \cdots\ L_2^0$ corresponding to proofs by recurrence on the data with respective ranks λ and η.

Observe that the termination analysis method of [9] can be seen as implicitly relying on Burstall's proof method.

15.3 Trace-based semantic structural induction

The previous examples of Sect. 15.2 show the need to go beyond purely syntactic, language-dependent induction and that induction on states can be generalized to induction on trace segments. Consequently, we introduce a general form of inductive reasoning on the semantic structure of computations, first starting by induction on blocks of trace segments and then their abstractions in Sect. 16.

15.3.1 Trace segment abstraction

We first observe that considering segments of traces is an abstraction. The *segment abstraction* $\langle \wp(\Sigma^{+\infty}),\ \subseteq\rangle \xleftrightarrow[\alpha^+]{\gamma^+} \langle \wp(\Sigma^{+\infty}),\ \subseteq\rangle$

$$\alpha^+(T) \triangleq \{\sigma \in \Sigma^{+\infty} \mid \exists \sigma' \in \Sigma^*, \sigma'' \in \Sigma^{*\infty} : \sigma'\sigma\sigma'' \in T\}$$

is the set of segments of traces of T. If $T, T' \in \wp(\Sigma^{+\infty})$, we define

$$T \Subset T' \triangleq T \subseteq \alpha^+(T') = \forall \sigma \in T : \exists \sigma', \sigma'' : \sigma'\sigma\sigma'' \in T'$$

to mean that all traces of T are segments of the traces of T'. We define the join

$$\biguplus_{i\in\Delta} T_i \triangleq \gamma^+\Big(\bigcup_{i\in\Delta} T_i\Big) = \{\sigma_{i_1} \dots \sigma_{i_n} \mid \forall k \in [1, n] : \sigma_{i_k} \in T_{i_k}\}$$

to be the set of all the traces made out of segments in the T_i, $i \in \Delta$.

15.3.2 Inductive trace segment cover

Definition 2. An *inductive trace segment cover* of a non-empty set $\chi \in \wp(\Sigma^{+\infty})$ of traces is a set $C \in \mathfrak{C}(\chi)$ of sequences S of members B of $\wp(\alpha^+(\chi))$ such that

1. if $S\,S' \in C$ then $S \in C$ (prefix-closure)
2. if $S \in C$ then $\exists S' : S =_\chi S'$ (root)
3. if $S\,BB' \in C$ then $B \supsetneq B'$ (well-foundedness)
4. if $S\,BB' \in C$ then $B \subseteq \biguplus_{S\,BB' \in C} B'$ (cover). ☐

Example 19. An example of inductive trace segment cover is trace partitioning [56]. ☐

Example 20. A variant function $v \in \Sigma \not\mapsto \mathbb{N}$ defines a trivial inductive trace cover. Each value $v \in \mathsf{codom}(v)$ defines segments starting with states σ such that $v(\sigma) = v$ of length at most v. ☐

The following definitions are classical for trees $C \in \mathfrak{C}(\chi)$.

$$\begin{aligned}
\mathsf{root}(C) &\triangleq \chi\\
\mathsf{leaves}(C) &\triangleq \{B \in \wp(\chi) \mid \exists S : S\,B \in C \land \forall S' : S\,BS' \notin C\}\\
\mathsf{inner}(C) &\triangleq \{B \in \wp(\chi) \mid \exists S, B', S' : S\,BB'S' \in C\}\\
\mathsf{nodes}(C) &\triangleq \mathsf{leaves}(C) \cup \mathsf{inner}(C)\\
\mathsf{sons}_C(B) &\triangleq \{B' \in \mathsf{nodes}(C) \mid \exists S, S' : S\,BB'S' \in C\}\,.
\end{aligned}$$

The *immediate component relation* $B' \rtimes_C B \triangleq B' \in \mathsf{sons}_C(B) = \exists S : SBB' \in C$ is well-founded, so that proofs on segments can be done by induction on this well-founded relation. The *component relation* $\rtimes_C^*$ is its reflexive transitive closure. The blocks of a cover C are $\mathsf{nodes}(C) \triangleq \{B \in \wp(\Sigma) \mid B \rtimes_C^* \Sigma\}$.

15.4 State cover induced by an inductive trace cover

Given an inductive trace cover $C \in \mathfrak{C}(\chi)$, $\chi \in \wp(\Sigma^{+\infty})$ of Def. 2, define the abstractions

$$\begin{array}{rcll}
\alpha^{\mathsf{ts}}(C) & \triangleq & \{\alpha^{\mathsf{ts}}(S) \mid S \in C\} & C \in \wp((\wp(\alpha^+(\chi)))^+) \\
\alpha^{\mathsf{ts}}(SS') & \triangleq & \alpha^{\mathsf{ts}}(S)\alpha^{\mathsf{ts}}(S') & S, S' \in (\wp(\alpha^+(\chi)))^+ \\
\alpha^{\mathsf{ts}}(B) & \triangleq & \{\alpha^{\mathsf{ts}}(\sigma) \mid \sigma \in B\} & B \in \wp(\alpha^+(\chi)) \\
\alpha^{\mathsf{ts}}(\sigma) & \triangleq & \{\sigma_i \mid i \in [0, |\sigma| - 1]\} & \sigma \in \alpha^+(\chi) .
\end{array}$$

Then $\alpha^{\mathsf{ts}}(C)$ is an inductive state cover in the sense of Def. 1.

15.5 Trace cover induced by a inductive state cover

Inversely, given an inductive state cover $C \in \mathfrak{C}(\chi)$, $\chi \in \wp(\Sigma)$ of Def. 1, define

$$\begin{array}{rcll}
\gamma^{\mathsf{st}}(C) & \triangleq & \{\gamma^{\mathsf{st}}(S) \mid S \in C\} & C \in \wp((\wp(\chi))^+) \\
\gamma^{\mathsf{st}}(SS') & \triangleq & \gamma^{\mathsf{st}}(S)\gamma^{\mathsf{st}}(S') & S, S' \in (\wp(\chi))^+ \\
\gamma^{\mathsf{st}}(B) & \triangleq & B^+ & B \in \wp(\chi)
\end{array}$$

We have $\langle \wp((\wp(\Sigma^+))^+), \subseteq\rangle \underset{\alpha^{\mathsf{ts}}}{\overset{\gamma^{\mathsf{st}}}{\leftrightarrows}} \langle \wp((\wp(\Sigma))^+), \subseteq\rangle$ and $\gamma^{\mathsf{ts}}(C)$ is an inductive trace cover of χ^+.

15.6 Syntactic trace cover

Similarly one can define the inductive state cover induced by the syntax of commands of a programming language by considering the states which control is in a given command. This in turns induces a trace cover which is the basis for e.g. Hoare logic or structural static analysis by induction on program commands, as opposed to induction on program transitions as in dataflow analysis.

15.6.1 Inductive proof method

We have a sound and complete *inductive proof method* of a semantic property $\Theta^{+\infty}[\![\mathrm{P}]\!] \cap \chi \in \mathcal{P}$ for an inductive trace cover $C \in \mathfrak{C}(\chi)$

$$\Theta^{+\infty}[\![\mathrm{P}]\!] \cap B \in \mathcal{P}, \quad B \in \mathsf{leaves}(C) \qquad \text{basis}$$

$$\frac{\forall B' \in \mathsf{sons}_C(B) : \Theta^{+\infty}[\![\mathrm{P}]\!] \cap B' \in \mathcal{P}}{\Theta^{+\infty}[\![\mathrm{P}]\!] \cap B \in \mathcal{P}}, \quad B \in \mathsf{inner}(C) \quad \text{induction}$$

In particular, for termination $\tau^{+\infty}[\![\mathrm{P}]\!] \subseteq \Sigma^+[\![\mathrm{P}]\!]$ with a trace cover $C \in \mathfrak{C}(\Sigma^{+\infty}[\![\mathrm{P}]\!])$, we get

$$\Theta^{+\infty}[\![\mathrm{P}]\!] \subseteq B \subseteq \Sigma^+, \quad B \in \mathsf{leaves}(C) \qquad \text{basis}$$

$$\frac{\forall B' \in \mathsf{sons}_C(B) : \Theta^{+\infty}[\![\mathrm{P}]\!] \subseteq B' \subseteq \Sigma^+}{\Theta^{+\infty}[\![\mathrm{P}]\!] \subseteq B \subseteq \Sigma^+}, \quad B \in \mathsf{inner}(C) \quad \text{induction}$$

Example 21. Another form of decomposition of reasonings on termination is proposed by the transition invariants proof method of Podelski-Rybalchenko [53] based on a relational semantics [15].

The transition invariants proof method of [53] can be seen as the α^{R} abstraction of the above inductive proof method based on an inductive trace cover of height 1 with root $\Sigma^+[\![\mathrm{P}]\!]$ and sons $\alpha^+(T_1)$, $\ldots$, $\alpha^+(T_n)$ where $T_1, \ldots, T_n \in \wp(\Sigma^+[\![\mathrm{P}]\!])$ such that

$$\Theta^{+\infty}[\![\mathrm{P}]\!] \subseteq \Sigma^+[\![\mathrm{P}]\!] \iff \forall i \in [1, n] : \Theta^{+\infty}[\![\mathrm{P}]\!] \cap T_i \subseteq \Sigma^+ .$$

The generalization by inductive trace covers is both on the use of trace segments (instead of their relational abstraction of Sect. 7.1), and the possibility of recursive application of the method by induction, including on data, à la Burstall [3]. $\square$

16. Abstract semantic structural induction

Assume that we can prove a program trace property in the concrete using an inductive trace cover. Can we prove an abstract program property using the abstraction of the inductive trace cover? We have seen an example in Sect. 15.5. The question is whether this observation is general.

16.1 Abstract inductive cover

Definition 3. An *inductive abstract cover* of a trace semantics $\chi \in \wp(\Sigma^{+\infty})$ is an element $C \in A_C$ of an abstract domain A_C such that

$$\langle \wp((\wp(\Sigma^+))^+), \subseteq\rangle \underset{\alpha^{\mathsf{ta}}}{\overset{\gamma^{\mathsf{at}}}{\leftrightarrows}} \langle A_C, \sqsubseteq_C\rangle$$

and $\gamma^{\mathsf{ts}}(C)$ is an inductive trace cover of χ. $\square$

A standard way to define such inductive abstract covers is to follow the example of Sect. 15.5 generalized to a block abstraction $\langle \wp(\Sigma^+), \subseteq\rangle \underset{\alpha^{\mathsf{ta}}}{\overset{\gamma^{\mathsf{at}}}{\leftrightarrows}} \langle A_B, \sqsubseteq_B\rangle$. We get the cover abstraction $\langle \wp((\wp(\Sigma^+))^+), \subseteq\rangle \underset{\alpha^{\mathsf{ta}}}{\overset{\gamma^{\mathsf{at}}}{\leftrightarrows}} \langle \wp((A_B)^+), \subseteq\rangle$ by generalizing α^{ta} to sequences of abstract blocks and sets of such abstract sequences as follows

$$\begin{array}{rcll}
\gamma^{\mathsf{at}}(SS') & \triangleq & \gamma^{\mathsf{at}}(S)\gamma^{\mathsf{at}}(S') & S, S' \in (A_B)^+ \\
\gamma^{\mathsf{at}}(C) & \triangleq & \{\gamma^{\mathsf{at}}(S) \mid S \in C\} & C \in \wp((A_B)^+) .
\end{array}$$

Then A_C is chosen to be the set of elements $C \in \wp((A_B)^+)$ of sequences S of members B of A_B such that

1. if $SS' \in C$ then $S \in C$ (prefix-closure)
2. if $S \in C$ then $\exists S' : S = \alpha^{\mathsf{at}}(\chi)S'$ (root)
3. if $SBB' \in C$ then $\gamma^{\mathsf{at}}(B) \supsetneq \gamma^{\mathsf{at}}(B')$ (well-foundedness)
4. if $SBB' \in C$ then $\gamma^{\mathsf{at}}(B) \subseteq \biguplus_{SBB' \in C} \gamma^{\mathsf{at}}(B')$ (cover).

It follows that any $C \in A_C$ is an inductive abstract cover of the trace semantics $\chi \in \wp(\Sigma^{+\infty})$ in the sense of Def. 3.

Example 22. The transition invariant proof method of [53] follows from the relational abstraction $\langle \wp(\Sigma^+), \subseteq\rangle \underset{\alpha^{\mathsf{ta}}}{\overset{\gamma^{\mathsf{at}}}{\leftrightarrows}} \langle \wp(\Sigma \times \Sigma), \subseteq\rangle$ [15] where $\alpha^{\mathsf{ta}}(B) \triangleq \{\langle \sigma_0, \sigma_n - 1\rangle \mid n > 0 \wedge \sigma \in B \cap \Sigma^n\}$ is limited to the trace covers of the form given in Ex. 21. $\square$

16.2 Abstract inductive proof

The inductive proof method of Sect. 15.6.1 can be abstracted as follows.

$$\alpha^{\mathsf{at}}(\Theta^{+\infty}[\![\mathrm{P}]\!]) \sqsubseteq_C B, \quad B \in \mathsf{leaves}(C) \qquad \text{basis}$$

$$\frac{\forall B' \in \mathsf{sons}_C(B) : \alpha^{\mathsf{at}}(\Theta^{+\infty}[\![\mathrm{P}]\!]) \sqsubseteq_C B'}{\alpha^{\mathsf{at}}(\Theta^{+\infty}[\![\mathrm{P}]\!]) \sqsubseteq_C B^+}, \quad B \in \mathsf{inner}(C) \qquad \text{induction}$$

The proofs $\alpha^{\mathsf{at}}(\Theta^{+\infty}[\![\mathrm{P}]\!]) \sqsubseteq_C B$ can be done in the abstract by fixpoint induction using a fixpoint abstraction of the fixpoint definition of the trace semantics $\Theta^{+\infty}[\![\mathrm{P}]\!]$.

17. Related work

Most directly relevant work has been cited in the text. For programs with unbounded executions, any finite homomorphic abstraction must introduce a loop so that finite model-checking [4] or bounded model-checking [2] are unapplicable (or unsound) to prove termination (or non-termination). Nevertheless, predicate abstraction [41] remains applicable since it is a finite encoding of an infinite abstract interpretation [16]. With predicate abstraction the end-user is left with the hard problem of providing candidate variant functions [14], as in [1]. Moreover [27] shows that infinitary abstractions with widening/narrowing, as considered in this paper, are definitely strictly more powerful than finite abstractions. The computation of variant functions by abstraction is new, and different from the counter-example guided ways to find disjunctive ranking functions, used in tools like Terminator [7] and derivatives.

18. Conclusion

Abstract interpretation has established constructive principles for reasoning about semantics. A semantics is a fixpoint so proving a semantic property at some level of abstraction consists in verifying properties of abstract fixpoints which have to be checked (in checking/verification methods), guessed (in proof methods), or automatically inferred or approximated (in static analysis methods).

This principle was mainly applied in the past to invariance and indirectly to termination by reduction to invariance. We have shown that the abstract interpretation principle directly applies to both safety (generalizing invariance) and termination.

Moreover we have generalized the classical syntactic structural induction into the language-independent semantic concept of *semantic structural induction* based on (abstractions of) inductive trace covers which includes induction on syntax, control states, memory states, and execution trace segments and thus generalizes all verification and static analysis methods.

This methodology allowed us to establish new *principles* for proving termination by abstract interpretation of a termination semantics. It remains to design a suitable collection of abstract domains beyond the examples proposed in this paper and the corresponding implementations.

The present abstract interpretation termination framework has to be extended to liveness [6, 53] and more generally to inevitability under fairness hypotheses [35, 52, 55].

References

[1] I. Balaban, A. Pnueli, and L. Zuck. Modular ranking abstraction. *Int. J. Found. Comput. Sci.*, 18(1):5–44, 2007.

[2] A. Biere, A. Cimatti, E. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.

[3] R. Burstall. Program proving as hand simulation with a little induction. *Information Processing*, 308–312. North-Holland, 1974.

[4] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[5] M. Clarkson and F. Schneider. Hyperproperties. *Journal of Computer Security*, 18(6):1157–1210, 2010.

[6] B. Cook and E. Koskinen. Making prophecies with decision predicates. *POPL*, 399–410, 2011.

[7] B. Cook, A. Gotsman, A. Podelski, A. Rybalchenko, and M. Vardi. Proving that programs eventually do something good. *POPL*, 265–276, 2007.

[8] B. Cook, S. Gulwani, T. Lev-Ami, A. Rybalchenko, and M. Sagiv. Proving conditional termination. *CAV*, LNCS *5123*, 328–340, 2008.

[9] B. Cook, A. Podelski, and A. Rybalchenko. Summarization for termination: no return! *Form. Methods Syst. Des.*, 35:369–387, 2009.

[10] S. Cook. Soundness and completeness of an axiom system for program verification. *SIAM J. Comput.*, 7:70–80, 1978.

[11] P. Cousot. *Méthodes itératives de construction et d'approximation de points fixes d'opérateurs monotones sur un treillis, analyse sémantique de programmes*. Thèse d'État ès sciences math., USMG, Grenoble, 1978.

[12] P. Cousot. Semantic foundations of program analysis. *Program Flow Analysis: Theory and Applications*, ch. 10, 303–342. Prentice-Hall, 1981.

[13] P. Cousot. The calculational design of a generic abstract interpreter. M. Broy and R. Steinbrüggen, eds., *Calculational System Design*. NATO ASI Series F. IOS Press, Amsterdam, 1999.

[14] P. Cousot. Partial completeness of abstract fixpoint checking. *SARA*, LNCS *1864*, 1–25, 2000.

[15] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *TCS*, 277(1–2):47–103, 2002.

[16] P. Cousot. Verification by abstract interpretation. *Proc. Int. Symp. on Verification – Theory & Practice*, LNCS *2772*, 243–268, 2003.

[17] P. Cousot. Proving program invariance and termination by parametric abstraction, Lagrangian relaxation and semidefinite programming. *VMCAI*, LNCS *3385*, 1–24, 2005.

[18] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. *Proc. 2nd Int. Symp. on Programming*, 106–130. Dunod, Paris, 1976.

[19] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. *POPL*, 238–252, 1977.

[20] P. Cousot and R. Cousot. Static determination of dynamic properties of recursive procedures. *Formal Description of Programming Concepts*, 237–277. North-Holland, 1977.

[21] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *POPL*, 269–282, 1979.

[22] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *P. J. of Math.*, 82(1):43–57, 1979.

[23] P. Cousot and R. Cousot. Induction principles for proving invariance properties of programs. *Tools & Notions for Program Construction: an Advanced Course*, 75–119. Cambridge University Press, Cambridge, UK, 1982.

[24] P. Cousot and R. Cousot. "À la Floyd" induction principles for proving inevitability properties of programs. *Algebraic methods in semantics*, 277–312. Cambridge University Press, Cambridge, UK, 1985.

[25] P. Cousot and R. Cousot. Sometime = always + recursion ≡ always, on the equivalence of the intermittent and invariant assertions methods for proving inevitability properties of programs. *Acta Informatica*, 24:1–31, 1987.

[26] P. Cousot and R. Cousot. Abstract interpretation frameworks. *JLC*, 2(4):511–547, 1992.

[27] P. Cousot and R. Cousot. Comparing the Galois connection and widening/narrowing approaches to abstract interpretation. *PLILP*, LNCS *631*, 269–295, 1992.

[28] P. Cousot and R. Cousot. Inductive definitions, semantics and abstract interpretation. *POPL*, 83–94, 1992.

[29] P. Cousot and R. Cousot. "À la Burstall" intermittent assertions induction principles for proving inevitable ability properties of programs. *TCS*, 120(1): 123–155, 1993.

[30] P. Cousot and R. Cousot. Higher-order abstract interpretation (and application to comportment analysis generalizing strictness, termination, projection and PER analysis of functional languages). *Int. Conf. on Comp. Lang.*, 95–112, 1994.

[31] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among variables of a program. *POPL*, 84–97, 1978.

[32] P. Cousot, R. Cousot, and L. Mauborgne. A scalable segmented decision tree abstract domain. *Time for Verification, Essays in Memory of A. Pnueli*, LNCS *6200*, 72–95, 2010.

[33] P. Cousot, R. Cousot, and F. Logozzo. A parametric segmentation functor for fully automatic and scalable array content analysis. *POPL*, 105–118, 2011.

[34] P. Cousot, R. Cousot, and F. Logozzo. Precondition inference from intermittent assertions and application to contracts on collections. *VMCAI*, LNCS *6538*, 150–168, 2011.

[35] R. Cousot. *Fondements des méthodes de preuve d'invariance et de fatalité de programmes parallèles*. Thèse d'État ès sciences math, INPL, Nancy, 1985.

[36] B. Davey and H. Priestley. *Introduction to Lattices and Order, 2nd Edition*. Cambridge University Press, 2002.

[37] E. Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *CACM*, 18(8):453–457, 1975.

[38] E. Dijkstra. *A Discipline of Programming*. Prentice-Hall, 1976.

[39] J. Feret. The arithmetic-geometric progression abstract domain. *VMCAI*, LNCS *3385*, 42–58, 2005.

[40] R. Floyd. Assigning meaning to programs. *Proc. Symp. in Applied Math.*, Vol. 19, 19–32. Amer. Math. Soc., 1967.

[41] S. Graf and H. Saïdi. Construction of abstract state graphs with PVS. *CAV*, LNCS *1254*, 72–83, 1997.

[42] M. Heizmann, J. Hoenicke, and A. Podelski. Refinement of trace abstraction. *SAS*, LNCS *5673*, 69–85, 2009.

[43] C. Hoare. An axiomatic basis for computer programming. *Communications of the Association for Computing Machinery*, 12(10):576–580, 1969.

[44] Z. Manna and A. Pnueli. Axiomatic approach to total correctness of programs. *Acta Inf.*, 3:243–263, 1974.

[45] K. McMillan and L. Zuck. Invisible invariants and abstract interpretation. *SAS*, LNCS *6887*, 249–262, 2011.

[46] A. Miné. The octagon abstract domain. *HOSC*, 19:31–100, 2006.

[47] D. Monniaux. Automatic modular abstractions for template numerical constraints. *Logical Methods in Comp. Sci.*, 6(3), 2010.

[48] J. Morris and B. Wegbreit. Subgoal induction. *CACM*, 20(4):209–222, 1977.

[49] P. Naur. Proofs of algorithms by general snapshots. *BIT*, 6:310–316, 1966.

[50] D. Pataria. A constructive proof of Tarski's fixed-point theorem for DCPO's. *Reported by M.H. Escardó in "Joins in the frame of nuclei"*, Applied Categorical Structures *11 (2) 117–124*, 2003.

[51] G. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[52] A. Pnueli, A. Podelski, and A. Rybalchenko. Separating fairness and well-foundedness for the analysis of fair discrete systems. *TACAS*, LNCS *3440*, 124–139, 2005.

[53] A. Podelski and A. Rybalchenko. Transition invariants. *LICS*, 32–41, 2004.

[54] A. Podelski and A. Rybalchenko. A complete method for the synthesis of linear ranking functions. *VMCAI*, LNCS *2937*, 239–251, 2004.

[55] A. Podelski and A. Rybalchenko. Transition predicate abstraction and fair termination. *POPL*, 132–144, 2005.

[56] X. Rival and L. Mauborgne. The trace partitioning abstract domain. *TOPLAS*, 29(5), 2007.

[57] D. Scott and C. Strachey. Towards a mathematical semantics for computer languages. Tech. rep. PRG-6, Oxford Univ. Comp. Lab., 1971.

[58] A. Tarski. A lattice theoretical fixpoint theorem and its applications. *P. J. of Math.*, 5:285–310, 1955.

[59] R. Turing. Checking a large routine. *Con. on High Speed Automatic Calculating Machines, Math. Lab., Cambridge, UK*, 67–69, 1949.