Electronic Notes in Theoretical Computer Science 45 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume45.html 24 pages

A Case Study in Abstract Interpretation Based Program Transformation: Blocking Command Elimination [1]

Patrick Cousot [2]

Département d'informatique
École Normale Supérieure
45 rue d'Ulm
75230 Paris cedex 05, France

Radhia Cousot [3]

Laboratoire d'informatique, LIX
CNRS & École Polytechnique
91128 Palaiseau cedex, France

Abstract

We illustrate the design of correct semantics-based program transformations by abstract interpretation on blocking code elimination.

1 Introduction

Static program analysis is closely related to program transformation since a preliminary program analysis is necessary in order to collect information about the program runtime behaviors which is then used to decide which offline transformations are applicable [12,14]. Abstract interpretation [4,6,8,9] has been used as a formal basis for static program analysis. Abstract interpretation can also be used to define a semantics-based program transformation framework. This is a new approach to the formal design of program transformations and a

[1] This work is dedicated to Neil D. Jones on the occasion of his 60th anniversary. It was presented as part of the invited talk at the "Special session honoring Neil D. Jones" of the Seventeenth Conference on the Mathematical Foundations of Programming Semantics, Århus, Denmark , May 23 – 27, 2001. It was supported in part by the european FP5 project IST-1999-20527 Daedalus.

[2] cousot@ens.fr, www.di.ens.fr/~cousot

[3] rcousot@lix.polytechnique.fr, lix.polytechnique.fr/~rcousot

new application of the abstract interpretation theory. The idea is to formally design syntactic (that is source level) program transformations by abstraction of transformations of the program semantics. Abstract interpretation is used to formalize the correspondence between semantic and syntactic transformations. This yields the necessary formal basis for (hopefully mechanically) constructing correct program transformation tools and may be to systematize their design.

The framework is applied to *blocking code elimination*, which consists in eliminating blocking commands other than stop commands in imperative nondeterministic programs leaving non-terminating behaviors unchanged. The final algorithm is very simple and could have been designed empirically without error but this case study is simple enough to exemplify our approach.

We believe that this unified abstract interpretation based framework for reasoning on program transformation should be applicable to a wide variety of semantics-based program manipulations including constant propagation [15], transition compression [11], slicing [22], partial evaluation [13], continuation passing style transformation [18], call-by-name to call-by-value transformation [19], fold/unfold [3], deforestation [21], compilation [17], etc.

2 A Few Elements of Abstract Interpretation

2.1 Fixpoints

We write $\mathsf{lfp}^{\leq}_{\perp} F$ for the $\leq$-least fixpoint of F $\leq$-greater than or equal to $\perp$, when it exists. We write $\mathsf{lfp}^{\leq} F$ and $\mathsf{lfp}\ F$ when $\perp$ and $\leq$ are understood from the context. Dually, $\mathsf{gfp}^{\leq}_{\top} F$ is the $\leq$-greatest fixpoint of F $\leq$-less than or equal to $\top$, when it exists.

Theorem 2.1 (Least fixpoint) *Let $\mathsf{po}\langle \mathfrak{L}; \leq\rangle$ be a partially ordered set $\mathfrak{L}$ with a binary relation $\leq$ which is a partial order (reflexive, antisymmetric and transitive). Assume that F is a monotone operator on $\mathsf{po}\langle \mathfrak{L}; \leq\rangle$. Assume that $\perp \in \mathfrak{L}$ is such that $\perp \leq F(\perp)$. Let $L \subseteq \mathfrak{L}$ be a subset of $\mathfrak{L}$ such that $\perp \in L$, $\forall x \in L : x \leq F(x) \Rightarrow F(x) \in L$ and if $\langle x_i,\ i \in \Delta\rangle$ is an $\leq$-increasing chain of elements of L then the least upper bound (lub) $\vee_{i\in\Delta} x_i$ exists in $\mathsf{po}\langle \mathfrak{L}; \leq\rangle$ and satisfies $\vee_{i\in\Delta} x_i \in L$. Then $\mathsf{lfp}^{\leq}_{\perp} F$ exists, is unique and belongs to L.*

Proof. The proof easily derives from [7]. It is based on the iterative definition of fixpoints in the tradition of Tarski [20] and Kleene [16] $F^0 \triangleq \perp$, $F^{\delta+1} \triangleq F(F^\delta)$ for successor ordinals $\delta+1 \in \mathbb{O}$, $F^\lambda \triangleq \vee_{\delta<\lambda} F^\delta$ for limit ordinal $\lambda \in \mathbb{O}$.□

Most often, we express the semantics in least fixpoint form $\mathsf{lfp}^{\leq} F$ where the semantic transformer $F \in \mathfrak{L} \stackrel{\text{m}}{\longmapsto} \mathfrak{L}$ is a monotone operator on the complete partial order (cpo) $\mathsf{cpo}\langle \mathfrak{L}; \leq, \perp, \vee\rangle$. Dually, we can also use a greatest fixpoint $\mathsf{gfp}^{\leq} F$ of a monotone semantic transformer $F \in \mathfrak{L} \stackrel{\text{m}}{\longmapsto} \mathfrak{L}$ on the dual complete partial order (co-cpo) $\mathsf{ccpo}\langle \mathfrak{L}; \leq, \top, \wedge\rangle$.

Theorem 2.2 (Least fixpoint iterates) *Under the hypotheses of Th. 2.1, for all $x \in \mathfrak{L}$ such that $\bot \leq x \leq \mathsf{lfp}^{\leq}_{\bot} F$, the iterates $F^0 \triangleq x$, $F^{\delta+1} \triangleq F(F^\delta)$ for successor ordinals $\delta+1 \in \mathbb{O}$ and $F^\lambda \triangleq \vee_{\delta<\lambda} F^\delta$ for limit ordinal $\lambda \in \mathbb{O}$ are ultimately stationary and converge to $\mathsf{lfp}^{\leq}_{\bot} F$* [4] *.*

Proof. By monotony and transfinite induction, the iterates of F starting from x are sandwiched between the iterates of F starting from $\bot$ which are ultimately stationary and converge to $\mathsf{lfp}^{\leq}_{\bot} F$ and the iterates of F starting from $\mathsf{lfp}^{\leq}_{\bot} F$ which are all equal to $\mathsf{lfp}^{\leq}_{\bot} F$ by the fixpoint property, proving the ultimate convergence of all the iterates to that fixpoint. □

2.2 Abstraction

An abstraction $\alpha(S)$ of a concrete semantics S is defined by a Galois connection $\mathsf{po}\langle \mathfrak{L}; \leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathsf{po}\langle \overline{\mathfrak{L}}; \overline{\leq}\rangle$ between the concrete domain $\mathsf{po}\langle \mathfrak{L}; \leq\rangle$ and the abstract domain $\mathsf{po}\langle \overline{\mathfrak{L}}; \overline{\leq}\rangle$ which are both posets [5]. By definition, we have $\forall X \in \mathfrak{L} : \forall Y \in \overline{\mathfrak{L}} : \alpha(X) \overline{\leq} Y \Leftrightarrow X \leq \gamma(Y)$. It follows that α preserves existing lubs, by duality γ preserves existing greatest lower bounds (glbs) and one adjoint uniquely determines the other. We write $\mathsf{po}\langle \mathfrak{L}; \leq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows\!\!\!\!\rightarrow}} \mathsf{po}\langle \overline{\mathfrak{L}}; \overline{\leq}\rangle$ when α is surjective (or equivalently γ is injective) and $\mathsf{po}\langle \mathfrak{L}; \leq\rangle \underset{\alpha}{\overset{\gamma}{\leftarrow\!\!\!\!\leftrightarrows}} \mathsf{po}\langle \overline{\mathfrak{L}}; \overline{\leq}\rangle$ when α is injective (or equivalently γ is surjective).

Given $f \in A \longmapsto B$, a standard example is $\alpha(X) \triangleq \{f(x) \mid x \in X\}$ so that

$$\mathsf{po}\langle \wp(A); \subseteq\rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathsf{po}\langle \wp(B); \subseteq\rangle \tag{1}$$

where $\gamma(Y) \triangleq \{x \in A \mid f(x) \in Y\}$.

2.3 Fixpoint Coabstraction

We have the following sufficient condition for two fixpoints to have the same abstraction $\alpha_1(\mathsf{lfp}^{\leq_1} F_1) = \alpha_2(\mathsf{lfp}^{\leq_2} F_2)$ which is based on the iterative definition of fixpoints [7] :

Theorem 2.3 (Fixpoint coabstraction) *Let $F_1 \in \mathfrak{L}_1 \overset{m}{\longmapsto} \mathfrak{L}_1$ and $F_2 \in \mathfrak{L}_2 \overset{m}{\longmapsto} \mathfrak{L}_2$ be respective monotone operators on the complete partial orders $\mathsf{cpo}\langle \mathfrak{L}_1; \leq_1, \bot_1, \vee_1\rangle$ and $\mathsf{cpo}\langle \mathfrak{L}_2; \leq_2, \bot_2, \vee_2\rangle$. Let $\mathsf{cpo}\langle \overline{\mathfrak{L}}; \overline{\leq}, \overline{\bot}, \overline{\vee}\rangle$ be a complete partial order. Let $\alpha_1 \in \mathfrak{L}_1 \overset{\bot,\uparrow}{\longmapsto} \overline{\mathfrak{L}}$ and $\alpha_2 \in \mathfrak{L}_2 \overset{\bot,\uparrow}{\longmapsto} \overline{\mathfrak{L}}$ be $\bot$-strict* [6] *Scott-*

[4] Note that the iterates starting from x need not be an increasing chain.

[5] Other equivalent formalizations (e.g. using closure operators) are given in [8] and weaker ones, not assuming the existence of a best approximation, are provided in [9].

[6] A function f is *$\bot$-strict*, written $f : D \overset{\bot}{\longmapsto} E$, if and only if $f(\bot) = \bot$.

continuous [7] *abstraction functions satisfying the following* local coabstraction condition [8] *:*

$$(2) \qquad \forall x \in \mathfrak{L}_1 : \forall y \in \mathfrak{L}_2 : \alpha_1(x) = \alpha_2(y) \Rightarrow \alpha_1(F_1(x)) = \alpha_2(F_2(y)) \ .$$

Then $\alpha_1(\mathsf{lfp}^{\leq_1} F_1) = \alpha_2(\mathsf{lfp}^{\leq_2} F_2)$.

Proof. Let F_1^δ, $\delta \in \mathbb{O}$ and F_2^δ, $\delta \in \mathbb{O}$ be the respective transfinite iterates for F_1 and F_2 [7]. By monotony, they are increasing chains which are therefore well-defined in the respective complete partial orders $\mathsf{cpo}\langle \mathfrak{L}_1; \leq_1, \bot_1, \vee_1\rangle$ and $\mathsf{cpo}\langle \mathfrak{L}_2; \leq_2, \bot_2, \vee_2\rangle$.

α_1 and α_2 are $\bot$-strict so that $\alpha_1(\bot_1) = \overline{\bot} = \alpha_2(\bot_2)$ hence $F_1^0 = F_2^0$.

Let $\delta + 1$ be a successor ordinal such that $\alpha_1(F_1^\delta) = \alpha_2(F_2^\delta)$ by induction hypothesis. By the local coabstraction condition (2), we have $\alpha_1(F_1^{\delta+1}) = \alpha_1(F_1(F_1^\delta)) = \alpha_2(F_2(F_2^\delta)) = \alpha_2(F_2^{\delta+1})$.

Let λ be a limit ordinal such that by induction hypothesis, $\forall \delta < \lambda$: $\alpha_1(F_1^\delta) = \alpha_2(F_2^\delta)$. Then, by continuity of α_1 and α_2 and induction hypothesis, we have $\alpha_1(F_1^\lambda) = \alpha_1(\bigvee_{\delta<\lambda}^{1} F_1^\delta) = \overline{\bigvee}_{\delta<\lambda} \alpha_1(F_1^\delta) = \overline{\bigvee}_{\delta<\lambda} \alpha_2(F_2^\delta) = \alpha_2(\bigvee_{\delta<\lambda}^{2} F_2^\delta) = \alpha_2(F_2^\lambda)$.

By transfinite induction, we conclude that $\forall \delta \in \mathbb{O}$: $\alpha_1(F_1^\delta) = \alpha_2(F_2^\delta)$. Let $\epsilon_1 \in \mathbb{O}$ and $\epsilon_2 \in \mathbb{O}$ be such that $\epsilon_1 = \mathsf{lfp}^{\leq_1} F_1$ and $\epsilon_2 = \mathsf{lfp}^{\leq_2} F_2$ [7]. We have $\alpha_1(\mathsf{lfp}^{\leq_1} F_1) = \alpha_1(F_1^{\epsilon_1}) = \alpha_1(F_1^{\max(\epsilon_1,\epsilon_2)}) = \alpha_2(F_2^{\max(\epsilon_1,\epsilon_2)}) = \alpha_2(F_2^{\epsilon_2}) = \alpha_2(\mathsf{lfp}^{\leq_2} F_2)$. □

2.4 Locally Complete Fixpoint Abstraction

In particular when $\alpha_1 = \alpha$ and α_2 is the identity, Th. 2.3 yields a sufficient condition for complete (or exact) fixpoint abstractions $\alpha(\mathsf{lfp}^{\leq} F) = \mathsf{lfp}^{\overline{\leq}} \overline{F}$, which provides guidelines for designing $\mathsf{lfp}^{\overline{\leq}} \overline{F}$ from $\mathsf{lfp}^{\leq} F$ (or dually) in fixpoint form [8, theorem 7.1.0.4(3)], [10, lemma 4.3], [2, fact 2.3] [9] :

Corollary 2.4 (Fixpoint transfer) *Let* $F \in \mathfrak{L} \stackrel{m}{\longmapsto} \mathfrak{L}$ *be a monotonic operator on the* $\mathsf{cpo}\langle \mathfrak{L}; \leq, \bot, \vee\rangle$*, let* $\overline{F} \in \overline{\mathfrak{L}} \stackrel{m}{\longmapsto} \overline{\mathfrak{L}}$ *be a monotone operator on the* $\mathsf{cpo}\langle \overline{\mathfrak{L}}; \overline{\leq}, \overline{\bot}, \overline{\vee}\rangle$ *and let* $\alpha \in \mathfrak{L} \stackrel{\bot,\vee}{\longmapsto} \overline{\mathfrak{L}}$ *be a* $\bot$*-strict Scott-continuous abstraction function satisfying the* commutation condition $\overline{F} \circ \alpha = \alpha \circ F$ [10] .

---

[7] A function f is *Scott-continuous*, written $f : D \stackrel{\vee}{\longmapsto} E$, if and only if it preserves the lub of any directed subset of D [1] (so that it is monotone).

[8] As in Th. 2.1, it is sufficient to assume that α_i is $\bot$-strict, preserves the least upper bound of the iterates of F_i starting from $\bot_i$, $i = 1, 2$ and that the local coabstraction condition holds for these iterates or a given superset of the iterates.

[9] The composition of relations r_1 and r_2 is $r_1 \circ r_2 \triangleq \{\langle x, z\rangle \mid \exists y : \langle x, y\rangle \in r_1 \wedge \langle y, z\rangle \in r_2\}$ whence the composition of functions is $f \circ g(x) \triangleq f(g(x))$.

[10] As in Th. 2.1, it is sufficient to assume that α is $\bot$-strict, preserves the least upper bound of the iterates of F starting from $\bot$ and that the commutation condition holds for these iterates.

Then $\alpha(\mathsf{lfp}^{\leq} F) = \mathsf{lfp}^{\overline{\leq}} \overline{F}$.

2.5 Fixpoint Approximation

Due to undecidability, it is often impossible to abstract a fixpoint $\alpha(\mathsf{lfp}^{\leq} F) = \mathsf{lfp}^{\overline{\leq}} \overline{F}$ exactly and to require simultaneously the abstract fixpoint $\mathsf{lfp}^{\overline{\leq}} \overline{F}$ to be effectively computable. In that case, abstract interpretation theory offers fixpoint approximation methods so that $\alpha(\mathsf{lfp}^{\leq} F) \mathrel{\overline{\leq}} \mathsf{lfp}^{\overline{\leq}} \overline{F}$ [6,8,9]. Let us recall these basic fixpoint approximation results in a generalized form:

Theorem 2.5 (Least fixpoint upper approximation) *Let* $F \in \mathfrak{L} \stackrel{m}{\longmapsto} \mathfrak{L}$ *be a monotonic operator on the complete partial order* $\mathsf{cpo}\langle \mathfrak{L}; \leq, \bot, \vee \rangle$ *and let* $\overline{F} \in \overline{\mathfrak{L}} \stackrel{m}{\longmapsto} \overline{\mathfrak{L}}$ *be a monotone operator on* $\mathsf{cpo}\langle \overline{\mathfrak{L}}; \overline{\leq}, \overline{\bot}, \overline{\vee} \rangle$.

Assume that the $\bot$*-strict Scott-continuous abstraction function* $\alpha \in \mathfrak{L} \stackrel{\bot,\uparrow}{\longmapsto} \overline{\mathfrak{L}}$ *is such that for all* $x \in \mathfrak{L}$ *such that* $x \leq F(x)$ *there exists* $y \leq x$ *such that* $\alpha(F(x)) \mathrel{\overline{\leq}} \overline{F}(\alpha(y))$.

Then $\alpha(\mathsf{lfp}^{\leq} F) \mathrel{\overline{\leq}} \mathsf{lfp}^{\overline{\leq}} \overline{F}$.

Proof. Let F^δ and $\overline{F}^\delta$, $\delta \in \mathbb{O}$ be the respective ordinal-termed $\leq$ and $\overline{\leq}$-increasing ultimately stationary chains of transfinite iterates of F and $\overline{F}$ [7]. We have $\alpha(F^0) = \alpha(\bot) = \overline{\bot} = \overline{F}^0$ by strictness of α and definition of the iterates. Assume $\alpha(F^\delta) \mathrel{\overline{\leq}} \overline{F}^\delta$ by induction hypothesis. We have $F^\delta \leq F(F^\delta) = F^{\delta+1}$ so that, by hypothesis, $\exists y \leq F^\delta$ such that $\alpha(F^{\delta+1}) \mathrel{\overline{\leq}} \overline{F}(\alpha(y))$. By monotony of $\overline{F}$ and α, $\overline{F}(\alpha(y)) \leq \overline{F}(\alpha(F^\delta))$ whence by transitivity, induction hypothesis, monotony of $\overline{F}$ and definition of the iterates, $\alpha(F^{\delta+1}) \mathrel{\overline{\leq}} \overline{F}(\alpha(F^\delta)) \mathrel{\overline{\leq}} \overline{F}(\overline{F}^\delta) = \overline{F}^{\delta+1}$. Given a limit ordinal λ, assume $\alpha(F^\delta) \mathrel{\overline{\leq}} \overline{F}^\delta$ for all $\delta < \lambda$. Then by definition of the iterates, continuity of α, induction hypothesis and definition of lubs, $\alpha(F^\lambda) = \alpha(\bigvee\limits_{\delta<\lambda} F^\delta) = \bigvee\limits_{\delta<\lambda} \alpha(F^\delta) \mathrel{\overline{\leq}} \bigvee\limits_{\delta<\lambda} \overline{F}^\delta = \overline{F}^\lambda$. By transfinite induction, we conclude $\forall \delta \in \mathbb{O} : \alpha(F^\delta) \mathrel{\overline{\leq}} \overline{F}^\delta$.

Let ϵ and ϵ' be the respective ordinals such that $F^\epsilon = \mathsf{lfp}^{\leq} F$ and $\overline{F}^{\epsilon'} = \mathsf{lfp}^{\overline{\leq}} \overline{F}$. In particular $\alpha(\mathsf{lfp}^{\leq} F) = \alpha(F^\epsilon) = \alpha(F^{\max\{\epsilon,\epsilon'\}}) \mathrel{\overline{\leq}} \overline{F}^{\max\{\epsilon,\epsilon'\}} = \overline{F}^{\epsilon'} = \mathsf{lfp}^{\overline{\leq}} \overline{F}$. □

The dual of the above Th. 2.5 leads to the *approximation of greatest fixpoints from below*. We also need to approximate greatest fixpoints *from above*, as follows:

Theorem 2.6 (Greatest fixpoint upper approximation 1) *Assume that* $F \in \mathfrak{L} \stackrel{m}{\longmapsto} \mathfrak{L}$ *is a monotonic operator on the co-cpo* $\mathsf{ccpo}\langle \mathfrak{L}; \leq, \top, \wedge \rangle$ *and that* $\overline{F} \in \overline{\mathfrak{L}} \stackrel{m}{\longmapsto} \overline{\mathfrak{L}}$ *is a monotone operator on* $\mathsf{ccpo}\langle \overline{\mathfrak{L}}; \overline{\leq}, \overline{\top}, \overline{\wedge} \rangle$.

Let the Scott-co-continuous abstraction function $\alpha \in \mathfrak{L} \stackrel{\downarrow}{\longmapsto} \overline{\mathfrak{L}}$ *be such that for all* $x \in \mathfrak{L}$ *such that* $F(x) \leq x$ *there exists* $y \leq x$ *such that* $\alpha(F(x)) \mathrel{\overline{\leq}} \overline{F}(\alpha(y))$.

Then $\alpha(\mathsf{gfp}^{\leq}_{\top} F) \overline{\leq} \mathsf{gfp}^{\overline{\leq}}_{\alpha(\top)} \overline{F}$.

Proof. Let F^δ and $\overline{F}^\delta$, $\delta \in \mathbb{O}$ be the respective ordinal-termed $\leq$ and $\overline{\leq}$-decreasing ultimately stationary chains of transfinite iterates of F and $\overline{F}$ respectively starting from $\top$ and $\overline{\top}$ [7].

We have $\alpha(F^0) = \alpha(\top) = \overline{F}^0$ by definition of the iterates.

Assume $\alpha(F^\delta) \overline{\leq} \overline{F}^\delta$ by induction hypothesis. We have $F^{\delta+1} = F(F^\delta) \leq F^\delta$ so that, by hypothesis, there exists $y \leq F^\delta$ such that $\alpha(F^{\delta+1}) \overline{\leq} \overline{F}(\alpha(y))$. By monotony of $\overline{F}$ and α, $\overline{F}(\alpha(y)) \leq \overline{F}(\alpha(F^\delta))$ whence by transitivity, induction hypothesis, monotony of $\overline{F}$ and definition of the iterates, $\alpha(F^{\delta+1}) \overline{\leq} \overline{F}(\alpha(F^\delta)) \overline{\leq} \overline{F}(\overline{F}^\delta) = \overline{F}^{\delta+1}$.

Given a limit ordinal λ, assume $\alpha(F^\delta) \overline{\leq} \overline{F}^\delta$ for all $\delta < \lambda$. Then by definition of the iterates, co-continuity of α, induction hypothesis and definition of glbs, $\alpha(F^\lambda) = \alpha(\bigwedge_{\delta<\lambda} F^\delta) = \overline{\bigwedge_{\delta<\lambda}} \alpha(F^\delta) \overline{\leq} \overline{\bigwedge_{\delta<\lambda}} \overline{F}^\delta = \overline{F}^\lambda$.

By transfinite induction, we conclude that $\forall \delta \in \mathbb{O} : \alpha(F^\delta) \overline{\leq} \overline{F}^\delta$.

Let ϵ and ϵ' be the respective ordinals such that $F^\epsilon = \mathsf{gfp}^{\leq} F$ and $\overline{F}^{\epsilon'} = \mathsf{gfp}^{\overline{\leq}} \overline{F}$. In particular $\alpha(\mathsf{gfp}^{\leq} F) = \alpha(F^\epsilon) = \alpha(F^{\max\{\epsilon,\epsilon'\}}) \overline{\leq} \overline{F}^{\max\{\epsilon,\epsilon'\}} = \overline{F}^{\epsilon'} = \mathsf{gfp}^{\overline{\leq}} \overline{F}$. □

A useful variant is:

Theorem 2.7 (Greatest fixpoint upper approximation 2) *Assume that* $F \in \mathfrak{L} \overset{m}{\longmapsto} \mathfrak{L}$ *is a monotonic operator on the co-cpo* $\mathsf{ccpo}\langle \mathfrak{L}; \leq, \top, \wedge \rangle$ *and that* $\overline{F} \in \overline{\mathfrak{L}} \overset{m}{\longmapsto} \overline{\mathfrak{L}}$ *is a monotone operator on* $\mathsf{ccpo}\langle \overline{\mathfrak{L}}; \overline{\leq}, \overline{\top}, \overline{\wedge} \rangle$.

Let $\mathsf{po}\langle \mathfrak{L}; \leq \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \mathsf{po}\langle \overline{\mathfrak{L}}; \overline{\leq} \rangle$ *be such that for all* $x \in \mathfrak{L}$ *such that* $F(x) \leq x$ *there exists* $y \leq x$ *such that* $\alpha(F(x)) \overline{\leq} \overline{F}(\alpha(y))$.

Then $\alpha(\mathsf{gfp}^{\leq}_{\top} F) \overline{\leq} \mathsf{gfp}^{\overline{\leq}}_{\alpha(\top)} \overline{F}$.

Proof. The proof is similar to that of Th. 2.6 except for limit ordinals. Given a limit ordinal λ such that $\alpha(F^\delta) \overline{\leq} \overline{F}^\delta$ for all $\delta < \lambda$, we have $F^\delta \leq \gamma(\overline{F}^\delta)$ for all $\delta < \lambda$, by definition of Galois connections. Since $\langle F^\delta, \delta < \lambda \rangle$ and $\langle \overline{F}^\delta, \delta < \lambda \rangle$ are decreasing chains and γ is monotone, $\langle \gamma(\overline{F}^\delta), \delta < \lambda \rangle$ is also decreasing so that $\bigwedge_{\delta<\lambda} F^\delta$ and $\bigwedge_{\delta<\lambda} \gamma(\overline{F}^\delta)$ on one hand and $\overline{\bigwedge_{\delta<\lambda}} \overline{F}^\delta$ on the other hand do exist respectively in the co-cpos $\mathsf{ccpo}\langle \mathfrak{L}; \leq, \top, \wedge \rangle$ and $\mathsf{ccpo}\langle \overline{\mathfrak{L}}; \overline{\leq}, \overline{\top}, \overline{\wedge} \rangle$. By definition of glbs and γ preserving existing glbs, we have $\bigwedge_{\delta<\lambda} F^\delta \leq \bigwedge_{\delta<\lambda} \gamma(\overline{F}^\delta) = \gamma(\overline{\bigwedge_{\delta<\lambda}} \overline{F}^\delta)$. By definition of Galois connections, it follows that $\alpha(\bigwedge_{\delta<\lambda} F^\delta) \overline{\leq} \overline{\bigwedge_{\delta<\lambda}} \overline{F}^\delta$. By definition of the iterates, we conclude that $\alpha(F^\lambda) = \alpha(\bigwedge_{\delta<\lambda} F^\delta) \overline{\leq} \overline{\bigwedge_{\delta<\lambda}} \overline{F}^\delta = \overline{F}^\lambda$. □

3 The Syntax and Semantics of Programs

Let us consider imperative iterative programs acting on global variables. Programs are assumed to be compiled in an intermediate form as shown by the following example:

`X := ?;`	$\mathtt{a} : \mathtt{X} := ? \rightarrow \mathtt{b};$
`while X > 0 do`	$\mathtt{b} : (\mathtt{X} > 0) \rightarrow \mathtt{c};$
	$\mathtt{b} : \neg(\mathtt{X} > 0) \rightarrow \mathtt{d};$
`  X := X + 1;`	$\mathtt{c} : \mathtt{X} + 1 \rightarrow \mathtt{b};$
`od;`	$\mathtt{d} : \mathtt{stop};$

Programs are nondeterministic. The intuition is that if execution is at some label `L` then one of the transitions $\mathtt{L} : \mathtt{A} \rightarrow \mathtt{L}'\mathtt{;}$ labeled with `L` is executed, provided the action `A` is not blocking and the execution can go on by branching to the next label $\mathtt{L}'$. Otherwise the execution is blocked at `L`, which is the case for the stop command $\mathtt{L} : \mathtt{stop;}$ intended to correspond to normally expected termination while other blocking commands are supposed to be erroneous.

Nondeterminism is modeled by having several actions be referenced by the same label. For example, the random assignment $\{\mathtt{L}_1 : \mathtt{X} := ? \rightarrow \mathtt{L}_2\mathtt{;}\}$ which is a shorthand for $\{\mathtt{L}_1 : \mathtt{X} := z \rightarrow \mathtt{L}_2\mathtt{;} \mid z \in \mathbb{Z}\}$, where $\mathbb{Z}$ is the set of integers, can be used to model interactive integer inputs.

3.1 Abstract Syntax of Programs

$\mathtt{X} : \mathbb{X}$	Program variables	$\mathtt{A} ::= \mathtt{X := E}$	Assignment
$\mathtt{E} : \mathbb{E}$	Arithmetic expressions	$\mid \mathtt{X} := ?$	Random assignment
$\mathtt{B} : \mathbb{B}$	Boolean expressions	$\mid \mathtt{B}$	Test
$\mathtt{A} : \mathbb{A}$	Program actions	$\mid \neg\mathtt{B}$	Negated test
		$\mid \mathtt{skip}$	Null action
		$\mid \mathtt{stop}$	Stop action

Programs are collections of labelled nondeterministic commands:

$\mathtt{L} : \mathbb{L}$		Program labels
$\mathtt{C} : \mathbb{C}$		Commands
$\mathtt{C} ::= \mathtt{L}_1 : \mathtt{A} \rightarrow \mathtt{L}_2\mathtt{;}$	$\mathsf{label}[\![\mathtt{C}]\!] \triangleq \mathtt{L}_1,\ \mathsf{action}[\![\mathtt{C}]\!] \triangleq \mathtt{A},\ \mathsf{succ}[\![\mathtt{C}]\!] \triangleq \{\mathtt{L}_2\}$	Transition command
$\mid \mathtt{L}_1 : \mathtt{stop;}$	$\mathsf{label}[\![\mathtt{C}]\!] \triangleq \mathtt{L}_1,\ \mathsf{action}[\![\mathtt{C}]\!] \triangleq \mathtt{stop},$	Stop command

$$\mathsf{succ}[\![\mathtt{C}]\!] \triangleq \emptyset$$

$$\mathtt{P} \ : \ \mathbb{P} \triangleq \wp(\mathbb{C}) \qquad \mathsf{labels}[\![\mathtt{P}]\!] \triangleq \{\mathsf{label}[\![\mathtt{C}]\!] \mid \mathtt{C} \in \mathtt{P}\} \qquad \text{Programs}$$

3.2 Semantics of Program Actions

The commands of a program act on global variables $\mathtt{X} \in \mathbb{X}$ which take their values in the semantic domain $\mathfrak{V}$.

An *environment* $\rho \in \mathfrak{E}$ maps variables $\mathtt{X}$ to their value $\rho(\mathtt{X})$ so $\mathfrak{E} \triangleq \mathbb{X} \longmapsto \mathfrak{V}$. $\rho[\mathtt{X} := d]$ is the environment ρ where the variable $\mathtt{X}$ is assigned the value d: $\rho[\mathtt{X} := d](\mathtt{X}) \triangleq d$ and $\rho[\mathtt{X} := d](\mathtt{Y}) \triangleq \rho(\mathtt{Y})$ when $\mathtt{X} \neq \mathtt{Y}$.

The semantics of expressions is assumed to be given by $\mathsf{A}[\![\mathtt{E}]\!] \in \mathfrak{E} \longmapsto \mathfrak{V}$ for arithmetic expressions $\mathtt{E}$ and by $\mathsf{B}[\![\mathtt{B}]\!] \in \mathfrak{E} \longmapsto \mathfrak{B}$ where $\mathfrak{B} \triangleq \{\mathrm{tt}, \mathrm{ff}\}$ for boolean expressions $\mathtt{B}$.

The *semantics* $\mathsf{S}[\![\mathtt{A}]\!]\rho$ of an action $\mathtt{A}$ defines the effect of executing this action on the environment ρ. Nondeterministic statements such as the random assignment $\mathtt{X} := \mathtt{?}$ have more than one possible successor environment so we define $\mathsf{S} \in \mathbb{A} \longmapsto (\mathfrak{E} \longmapsto \wp(\mathfrak{E}))$ as follows:

$$\begin{array}{rlrl}
\mathsf{S}[\![\mathtt{B}]\!] &\triangleq \boldsymbol{\lambda}\, \rho \bullet \{\rho \mid \mathsf{B}[\![\mathtt{B}]\!]\rho = \mathrm{tt}\} & \mathsf{S}[\![\mathtt{X} := \mathtt{?}]\!] &\triangleq \boldsymbol{\lambda}\, \rho \bullet \{\rho[\mathtt{X} := z] \mid z \in \mathbb{Z}\} \\
\mathsf{S}[\![\neg\, \mathtt{B}]\!] &\triangleq \boldsymbol{\lambda}\, \rho \bullet \{\rho \mid \mathsf{B}[\![\mathtt{B}]\!]\rho = \mathrm{ff}\} & \mathsf{S}[\![\mathtt{skip}]\!] &\triangleq \boldsymbol{\lambda}\, \rho \bullet \{\rho\} \\
\mathsf{S}[\![\mathtt{X := E}]\!] &\triangleq \boldsymbol{\lambda}\, \rho \bullet \{\rho[\mathtt{X} := \mathsf{A}[\![\mathtt{E}]\!]\rho]\} & \mathsf{S}[\![\mathtt{stop}]\!] &\triangleq \boldsymbol{\lambda}\, \rho \bullet \emptyset
\end{array}$$

3.3 States

A *state* $s \in \mathfrak{S}$ is a pair $s = \langle \rho,\ \mathtt{C} \rangle$ where the environment ρ records the values of variables while $\mathtt{C}$ is the next command to be executed:

$$\mathfrak{S} \triangleq \mathfrak{E} \times \mathbb{C}$$

The set of states $\mathfrak{S}[\![\mathtt{P}]\!]$ of a program $\mathtt{P} \in \mathbb{P}$ is defined as:

$$\mathfrak{S}[\![\mathtt{P}]\!] \triangleq \mathfrak{E} \times \mathtt{P} \tag{3}$$

3.4 Transitional Semantics

The *transitional semantics* $\mathsf{S}[\![\mathtt{P}]\!]s$ of a program $\mathtt{P} \in \mathbb{P}$ specifies which successor states s' can follow state s during execution of program $\mathtt{P}$:

$$\mathsf{S}[\![\mathtt{P}]\!] \in \mathfrak{S}[\![\mathtt{P}]\!] \longmapsto \wp(\mathfrak{S}[\![\mathtt{P}]\!])$$

$$\mathsf{S}[\![\mathtt{P}]\!]\langle \rho,\ \mathtt{C} \rangle \triangleq \{\langle \rho',\ \mathtt{C}' \rangle \mid \rho' \in \mathsf{S}[\![\mathsf{action}[\![\mathtt{C}]\!]]\!]\rho \wedge \mathsf{label}[\![\mathtt{C}']\!] \in \mathsf{succ}[\![\mathtt{C}]\!]\} \tag{4}$$

Observe that by Def. (3) of $\mathfrak{S}[\![\mathtt{P}]\!]$, we have $\mathtt{C} \in \mathtt{P}$ and $\mathtt{C}' \in \mathtt{P}$ in (4). In particular $\forall \rho \in \mathfrak{E} : \mathsf{S}[\![\mathtt{P}]\!]\langle \rho,\ \mathtt{L : stop;} \rangle = \emptyset$.

Example 3.1 The program:

$$\mathtt{P} = \{\underline{a}, \underline{a}', \underline{b}, \underline{c}\} \tag{5}$$

which commands are defined as follows:

$$\underline{a} \triangleq \mathtt{a : Y} > 0 \to \mathtt{b;} \qquad \underline{a}' \triangleq \mathtt{a :} \neg (\mathtt{Y} > 0) \to \mathtt{c;}$$
$$\underline{b} \triangleq \mathtt{b : Y := Y} - 1 \to \mathtt{a;} \qquad \underline{c} \triangleq \mathtt{c : stop;}$$

has the following transitional semantics:

$$\mathsf{S}[\![\mathtt{P}]\!]\langle\rho, \underline{a}\rangle = \{\langle\rho, \underline{b}\rangle \mid \rho(\mathtt{Y}) > 0\}, \qquad \mathsf{S}[\![\mathtt{P}]\!]\langle\rho, \underline{a}'\rangle = \{\langle\rho, \underline{c}\rangle \mid \rho(\mathtt{Y}) \leq 0\},$$
$$\mathsf{S}[\![\mathtt{P}]\!]\langle\rho, \underline{b}\rangle = \{\langle\rho[\mathtt{Y} := \rho(\mathtt{Y}) - 1], \underline{a}\rangle\}, \qquad \mathsf{S}[\![\mathtt{P}]\!]\langle\rho, \underline{c}\rangle = \emptyset\ .$$

□

3.5 Sequences of States

Program executions are recorded in finite or infinite sequences of states over a given set $\mathcal{C}$ of commands. Formally, we define ($\vec{\epsilon} \in \emptyset \longmapsto \mathfrak{S}[\![\mathcal{C}]\!]$ is the empty sequence of states, $[n, m] = \{k \in \mathbb{Z} \mid n \leq k \leq m\}$ so $[n, m] = \emptyset$ when $m < n$):

$$\Sigma^n[\![\mathcal{C}]\!] \triangleq [0, n-1] \longmapsto \mathfrak{S}[\![\mathcal{C}]\!], \qquad \Sigma^+[\![\mathcal{C}]\!] \triangleq \bigcup_{n>0} \Sigma^n[\![\mathcal{C}]\!],$$
$$\Sigma^*[\![\mathcal{C}]\!] \triangleq \Sigma^+[\![\mathcal{C}]\!] \cup \{\vec{\epsilon}\}, \qquad \Sigma^\omega[\![\mathcal{C}]\!] \triangleq \mathbb{N} \longmapsto \mathfrak{S}[\![\mathcal{C}]\!],$$
$$\Sigma^\infty[\![\mathcal{C}]\!] \triangleq \Sigma^+[\![\mathcal{C}]\!] \cup \Sigma^\omega[\![\mathcal{C}]\!], \qquad \Sigma^\infty[\![\mathcal{C}]\!] \triangleq \Sigma^\infty[\![\mathcal{C}]\!] \cup \{\vec{\epsilon}\}\ .$$

We define the length $\#\sigma$ of a sequence $\sigma \in \Sigma^\infty[\![\mathcal{C}]\!]$ as 0 when $\sigma = \vec{\epsilon}$, $n > 0$ when $\sigma \in \Sigma^n[\![\mathcal{C}]\!]$ and the first infinite limit ordinal ω when $\sigma \in \Sigma^\omega[\![\mathcal{C}]\!]$.

For short, we define ($\mathbb{C}$ is the set of commands defined in Sec. 3.1):

$$\Sigma^n \triangleq \Sigma^n[\![\mathbb{C}]\!], \qquad \Sigma^+ \triangleq \Sigma^+[\![\mathbb{C}]\!], \qquad \Sigma^* \triangleq \Sigma^*[\![\mathbb{C}]\!],$$
$$\Sigma^\omega \triangleq \Sigma^\omega[\![\mathbb{C}]\!], \qquad \Sigma^\infty \triangleq \Sigma^\infty[\![\mathbb{C}]\!], \qquad \Sigma^\infty \triangleq \Sigma^\infty[\![\mathbb{C}]\!]\ .$$

3.6 Complete Trace Semantics of Programs

A *finite complete execution trace* $\sigma \in \mathsf{S}^n[\![\mathtt{P}]\!]$ of a program $\mathtt{P} \in \mathbb{P}$ is a finite sequence $\sigma_0 \ldots \sigma_{n-1} \in \Sigma^n[\![\mathtt{P}]\!]$ of states of length $\#\sigma = n$ such that:

- each state $\sigma_i, i = 1, \ldots, n-1$ is the successor of the previous state σ_{i-1} so $\sigma_i \in \mathsf{S}[\![\mathtt{P}]\!]\sigma_{i-1}$, and
- the last state σ_{n-1} is a blocking state so $\mathsf{S}[\![\mathtt{P}]\!]\sigma_{n-1} = \emptyset$.

The finite complete traces are not empty so $\mathbf{S}^+[\![\mathtt{P}]\!] \triangleq \bigcup_{n>0} \mathbf{S}^n[\![\mathtt{P}]\!]$. We define $\mathbf{S}^*[\![\mathtt{P}]\!] \triangleq \mathbf{S}^+[\![\mathtt{P}]\!] \cup \{\vec{\epsilon}\}$ where $\vec{\epsilon}$ is the empty trace.

An *infinite execution trace* $\sigma \in \mathbf{S}^\omega[\![\mathtt{P}]\!]$ of a program $\mathtt{P} \in \mathbb{P}$ is an infinite sequence $\sigma_0 \dots \sigma_i \dots \in \Sigma^\omega[\![\mathtt{P}]\!]$ of states of infinite length $\#\sigma = \omega$ such that each state $\sigma_{i+1} \in \mathbf{S}[\![\mathtt{P}]\!]\sigma_i$ is the successor of the previous state σ_i, $0 \leq i < \omega$.

The *complete execution traces* of a program $\mathtt{P} \in \mathbb{P}$ are $\mathbf{S}^\infty[\![\mathtt{P}]\!] \triangleq \mathbf{S}^+[\![\mathtt{P}]\!] \cup \mathbf{S}^\omega[\![\mathtt{P}]\!]$ and $\mathbf{S}^\propto[\![\mathtt{P}]\!] \triangleq \mathbf{S}^\infty[\![\mathtt{P}]\!] \cup \{\vec{\epsilon}\} = \mathbf{S}^*[\![\mathtt{P}]\!] \cup \mathbf{S}^\omega[\![\mathtt{P}]\!]$.

Formally, the *trace semantics* of a program $\mathtt{P} \in \mathbb{P}$ is defined as follows:

$$
\begin{aligned}
(6) \qquad \mathbf{S}^\infty[\![\mathtt{P}]\!] &\triangleq \mathbf{S}^+[\![\mathtt{P}]\!] \cup \mathbf{S}^\omega[\![\mathtt{P}]\!],\\
\mathbf{S}^+[\![\mathtt{P}]\!] &\triangleq \bigcup_{n>0} \mathbf{S}^n[\![\mathtt{P}]\!],\\
\mathbf{S}^n[\![\mathtt{P}]\!] &\triangleq \{\sigma \in \Sigma^n[\![\mathtt{P}]\!] \mid \forall i \in [0, n-2] : \sigma_{i+1} \in \mathbf{S}[\![\mathtt{P}]\!]\sigma_i \wedge\\
&\qquad\qquad \mathbf{S}[\![\mathtt{P}]\!]\sigma_{n-1} = \emptyset\} \qquad \text{when } n > 0,\\
\mathbf{S}^\omega[\![\mathtt{P}]\!] &\triangleq \{\sigma \in \Sigma^\omega[\![\mathtt{P}]\!] \mid \forall i \geq 0 : \sigma_{i+1} \in \mathbf{S}[\![\mathtt{P}]\!]\sigma_i\}\ .
\end{aligned}
$$

Example 3.2 The trace semantics of program P defined by (5) is the following:

$$
\begin{aligned}
\mathbf{S}^\infty[\![\mathtt{P}]\!] = \quad & \{\langle \rho[\mathtt{Y} := n],\, \underline{a}\rangle \langle \rho[\mathtt{Y} := n],\, \underline{b}\rangle \langle \rho[\mathtt{Y} := n-1],\, \underline{a}\rangle \dots\\
& \qquad \dots \langle \rho[\mathtt{Y} := 0],\, \underline{a}'\rangle \langle \rho[\mathtt{Y} := 0],\, \underline{c}\rangle \mid \rho \in \mathfrak{E} \wedge n > 0\}\\
& \cup \{\langle \rho[\mathtt{Y} := n],\, \underline{a}'\rangle \langle \rho[\mathtt{Y} := n],\, \underline{c}\rangle \mid \rho \in \mathfrak{E} \wedge n \leq 0\}\\
& \cup \{\langle \rho[\mathtt{Y} := n+1],\, \underline{b}\rangle \langle \rho[\mathtt{Y} := n],\, \underline{a}\rangle \langle \rho[\mathtt{Y} := n],\, \underline{b}\rangle \langle \rho[\mathtt{Y} := n-1],\, \underline{a}\rangle \dots\\
& \qquad \dots \langle \rho[\mathtt{Y} := 0],\, \underline{a}'\rangle \langle \rho[\mathtt{Y} := 0],\, \underline{c}\rangle \mid \rho \in \mathfrak{E} \wedge n > 0\}\\
& \cup \{\langle \rho[\mathtt{Y} := n+1],\, \underline{b}\rangle \langle \rho[\mathtt{Y} := n],\, \underline{a}'\rangle \langle \rho[\mathtt{Y} := n],\, \underline{c}\rangle \mid \rho \in \mathfrak{E} \wedge n \leq 0\}\\
& \cup \{\langle \rho,\, \underline{c}\rangle \mid \rho \in \mathfrak{E}\}\ .
\end{aligned}
$$

□

3.7 Suffix-Closure

The suffix σ^+ of a trace $\sigma \in \Sigma^\infty$ is defined by $s^+ = s$ for traces of length 1 and $s\sigma^+ = \sigma$. Intuitively, σ^+ describes an execution starting one step after σ, if possible. When necessary we let $\vec{\epsilon}^+ = \vec{\epsilon}$.

The suffix of a set $\mathcal{T}$ of traces is $\mathcal{T}^+ \triangleq \{\sigma^+ \mid \sigma \in \mathcal{T}\}$.

A set $\mathcal{T}$ of traces is *suffix-closed* whenever $\mathcal{T}^+ \subseteq \mathcal{T}$. The *suffix-closure* of a set $\mathcal{T}$ of traces is the least suffix-closed superset $\mathcal{T}^\oplus = \mathsf{lfp}^{\subseteq}\, \boldsymbol{\lambda}\, \mathcal{X} \bullet \mathcal{T} \cup \mathcal{X}^+$ of $\mathcal{T}$.

Lemma 3.3 (Suffix-closed trace semantics) *The trace semantics (8) is suffix-closed.*

Proof. For finite traces $s \in \mathsf{S}^\infty[\![\mathtt{P}]\!]$ of length 1, we have $s^+ = s \in \mathsf{S}^\infty[\![\mathtt{P}]\!]$.

For finite traces $s\sigma \in \mathsf{S}^\infty[\![\mathtt{P}]\!]$, we have $s\sigma^+ = \sigma$ which belongs to $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ since each state $\sigma_i, i = 1, \ldots, n-1$ is the successor of the previous state σ_{i-1} and the last state σ_{n-1} is a blocking state, by definition of $s\sigma \in \mathsf{S}^\infty[\![\mathtt{P}]\!]$.

The same way for infinite traces $s\sigma \in \mathsf{S}^\infty[\![\mathtt{P}]\!]$, we have $s\sigma^+ = \sigma$ which belongs to $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ since each state $\sigma_{i+1} \in \mathsf{S}[\![\mathtt{P}]\!]\sigma_i$ is the successor of the previous state σ_i, $0 \leq i < \omega$, by definition of $s\sigma \in \mathsf{S}^\infty[\![\mathtt{P}]\!]$. □

3.8 Complete Trace Semantics of Programs in Fixpoint Form (1)

The trace transformer $\mathsf{F}^\infty[\![\mathtt{P}]\!]$ of a program $\mathtt{P} \in \mathbb{P}$ is defined as follows:

$$\mathsf{F}^\infty[\![\mathtt{P}]\!] \in \wp(\Sigma^\infty[\![\mathtt{P}]\!]) \longmapsto \wp(\Sigma^\infty[\![\mathtt{P}]\!])$$
$$\mathsf{F}^\infty[\![\mathtt{P}]\!]\mathcal{T} \stackrel{\triangle}{=} \{s \mid \mathsf{S}[\![\mathtt{P}]\!]s = \emptyset\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T}\} \tag{7}$$

Example 3.4 The trace transformer of the program P defined by (5) is the following:

$$\begin{aligned}\mathsf{F}^\infty[\![\mathtt{P}]\!]\mathcal{T} = \quad & \{\langle \rho, \underline{c}\rangle \mid \rho \in \mathfrak{E}\} \\ & \cup \{\langle \rho, \underline{a}\rangle\langle \rho, \underline{b}\rangle\sigma \mid \rho(\mathtt{Y}) > 0 \wedge \langle \rho, \underline{b}\rangle\sigma \in \mathcal{T}\} \\ & \cup \{\langle \rho, \underline{a}'\rangle\langle \rho, \underline{c}\rangle\sigma \mid \rho(\mathtt{Y}) \leq 0 \wedge \langle \rho, \underline{c}\rangle\sigma \in \mathcal{T}\} \\ & \cup \{\langle \rho[\mathtt{Y} := \rho(\mathtt{Y}) + 1], \underline{b}\rangle\langle \rho, \underline{a}\rangle\sigma \mid \langle \rho, \underline{a}\rangle\sigma \in \mathcal{T}\} \\ & \cup \{\langle \rho[\mathtt{Y} := \rho(\mathtt{Y}) + 1], \underline{b}\rangle\langle \rho, \underline{a}'\rangle\sigma \mid \langle \rho, \underline{a}'\rangle\sigma \in \mathcal{T}\}\end{aligned}$$

□

We have the following fixpoint characterizations of the program trace semantics [5]:

$$\mathsf{S}^\infty[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\Sigma^\infty} \mathsf{F}^\infty[\![\mathtt{P}]\!] \tag{8}$$

$\mathsf{F}^\infty[\![\mathtt{P}]\!]$ is $\subseteq$-monotone which ensures the existence of the fixpoints [20].

3.9 Feasible Traces

Some finite or infinite sequences of states such as $\langle \rho, \mathtt{L : stop;}\rangle^\omega$ do not correspond to any execution of any program. In order to eliminate such infeasible sequences of states, we restrict *traces* to the finite or infinite sequences of states corresponding to potential program executions:

$$\mathfrak{T}^n \triangleq \mathsf{S}^n[\![\mathbb{C}]\!], \qquad \mathfrak{T}^+ \triangleq \mathsf{S}^+[\![\mathbb{C}]\!], \qquad \mathfrak{T}^* \triangleq \mathsf{S}^*[\![\mathbb{C}]\!],$$
$$\mathfrak{T}^\omega \triangleq \mathsf{S}^\omega[\![\mathbb{C}]\!], \qquad \mathfrak{T}^\infty \triangleq \mathsf{S}^\infty[\![\mathbb{C}]\!], \qquad \mathfrak{T}^\varkappa \triangleq \mathsf{S}^\varkappa[\![\mathbb{C}]\!] \ .$$

3.10 Complete Trace Semantics of Programs in Fixpoint Form (2)

The trace transformer $\mathsf{F}^\infty[\![\mathtt{P}]\!]$ of a program $\mathtt{P} \in \mathbb{P}$ can also be defined using feasible traces only or arbitrary state sequences containing only commands of $\mathtt{P}$ only, as follows:

Theorem 3.5 (Fixpoint complete trace semantics of programs) *For all $\mathcal{T} \in \wp(\Sigma^\infty)$ such that $\mathsf{S}^\infty[\![\mathtt{P}]\!] \subseteq \mathcal{T}$, we have*

$$\mathsf{S}^\infty[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\mathcal{T}} \mathsf{F}^\infty[\![\mathtt{P}]\!] \ . \tag{9}$$

Proof. By (8) and the dual of Th. 2.2 since $\Sigma^\infty \supseteq \mathcal{T} \supseteq \mathsf{S}^\infty[\![\mathtt{P}]\!]$. □

Corollary 3.6

$$\mathsf{S}^\infty[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\Sigma^\infty[\![\mathtt{P}]\!]} \mathsf{F}^\infty[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\mathfrak{T}^\infty} \mathsf{F}^\infty[\![\mathtt{P}]\!] \tag{10}$$

Proof. Obviously $\Sigma^\infty[\![\mathtt{P}]\!] \subseteq \mathsf{gfp}^{\subseteq}_{\Sigma^\infty} \mathsf{F}^\infty[\![\mathtt{P}]\!]$ by (8) and Def. (6) of $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ which implies that all commands appearing along a trace of $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ belongs to $\mathtt{P}$ proving that this trace belongs to $\Sigma^\infty[\![\mathtt{P}]\!]$.

By (8), $\mathsf{gfp}^{\subseteq}_{\Sigma^\infty} \mathsf{F}^\infty[\![\mathtt{P}]\!]$ contains only feasible traces so $\mathsf{gfp}^{\subseteq}_{\Sigma^\infty} \mathsf{F}^\infty[\![\mathtt{P}]\!] \subseteq \mathfrak{T}^\infty$.

Applying Th. 3.5, we conclude that $\mathsf{S}^\infty[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\Sigma^\infty[\![\mathtt{P}]\!]} \mathsf{F}^\infty[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\mathfrak{T}^\infty} \mathsf{F}^\infty[\![\mathtt{P}]\!]$.□

4 Correspondence between Syntax and Trace Semantics of Programs

The trace semantics maps programs to sets of traces. Inversely, we map sets of traces to programs by collecting commands appearing along traces.

4.1 Trace-wide Command Collection

The abstraction $\mathbb{p}^\infty$ collects all commands on all traces, as follows:

$$\begin{array}{lll} \mathbb{p}^\infty \in \wp(\mathfrak{T}^\infty) \longmapsto \mathbb{P} & \mathbb{p} \in \mathfrak{T}^\infty \longmapsto \mathbb{P} & \mathbb{p} \in \mathfrak{S} \longmapsto \mathbb{C} \\ \mathbb{p}^\infty[\mathcal{T}] \triangleq \bigcup\limits_{\sigma \in \mathcal{T}} \mathbb{p}[\sigma] & \mathbb{p}[\sigma] \triangleq \{\mathbb{p}[\sigma_i] \mid 0 \leq i < \#\sigma\} & \mathbb{p}[\langle \rho, \mathtt{C} \rangle] \triangleq \mathtt{C} \end{array} \tag{11}$$

This correspondence is formalized by the following Galois connection:

Lemma 4.1

$$\mathsf{po}\langle \wp(\mathfrak{T}^\infty); \subseteq \rangle \underset{\mathbb{p}^\infty}{\overset{\mathsf{S}^\infty}{\leftrightarrows}} \mathsf{po}\langle \mathbb{P}; \subseteq \rangle$$

Proof. $\mathbb{p}^\infty$ and $\mathbf{S}^\infty$ are obviously $\subseteq$-monotone.

For all programs $\mathtt{P} \in \mathbb{P}$, we have $\mathbf{S}^\infty[\![\mathtt{P}]\!] \in (\mathfrak{E}\times\mathtt{P})^+ \cup (\mathfrak{E}\times\mathtt{P})^\omega$ so $\mathbb{p}^\infty[\mathbf{S}^\infty[\![\mathtt{P}]\!]] \subseteq \mathtt{P}$.

Inversely, for all $\mathtt{C} \in \mathtt{P}$, there may exist an environment $\rho \in \mathfrak{E}$ such that $\mathbf{S}[\![\mathtt{P}]\!]\langle \rho,\ \mathtt{C}\rangle = \emptyset$ in which case the trace $\langle \rho,\ \mathtt{C}\rangle$ belongs to $\mathbf{S}^\infty[\![\mathtt{P}]\!]$ et so $\mathtt{C}$ belongs to $\mathbb{p}^\infty[\mathbf{S}^\infty[\![\mathtt{P}]\!]]$. Otherwise, $\forall \rho \in \mathfrak{E} : \mathbf{S}[\![\mathtt{P}]\!]\langle \rho,\ \mathtt{C}\rangle \neq \emptyset$. Let $\langle \rho_0,\ \mathtt{C}_0\rangle = \langle \rho,\ \mathtt{C}\rangle$ and $\langle \rho_1,\ \mathtt{C}_1\rangle \in \mathbf{S}[\![\mathtt{P}]\!]\langle \rho_0,\ \mathtt{C}_0\rangle$. We have built a sequence $\sigma_n = \langle \rho_0,\ \mathtt{C}_0\rangle \ldots \langle \rho_n,\ \mathtt{C}_n\rangle$ of states, up to $n = 1$, such that $\forall i < n : \langle \rho_{i+1},\ \mathtt{C}_{i+1}\rangle \in \mathbf{S}[\![\mathtt{P}]\!]\langle \rho_i,\ \mathtt{C}_i\rangle$. Having built σ_n, we may have $\mathbf{S}[\![\mathtt{P}]\!]\langle \rho_n,\ \mathtt{C}_n\rangle = \emptyset$ in which case $\sigma_n \in \mathbf{S}^\infty[\![\mathtt{P}]\!]$ and consequently $\mathtt{C} \in \mathbb{p}^\infty[\mathbf{S}^\infty[\![\mathtt{P}]\!]]$ by definition of $\mathbb{p}^\infty$. Otherwise, we have $\exists \langle \rho_{n+1},\ \mathtt{C}_{n+1}\rangle \in \mathbf{S}[\![\mathtt{P}]\!]\langle \rho_n,\ \mathtt{C}_n\rangle$, so that σ_n can be extended to σ_{n+1}. If we can go on in this way for ever, we obtain a limit trace σ which nonempty prefixes are the $\sigma_n, n \geq 0$. We have $\sigma \in \mathbf{S}^\infty[\![\mathtt{P}]\!]$ and σ starts with $\langle \rho,\ \mathtt{C}\rangle$ so that $\mathtt{C} \in \mathbb{p}^\infty[\mathbf{S}^\infty[\![\mathtt{P}]\!]]$ by definition of $\mathbb{p}^\infty$. We conclude that $\mathtt{P} \subseteq \mathbb{p}^\infty[\mathbf{S}^\infty[\![\mathtt{P}]\!]]$.

By antisymmetry, we conclude that $\mathbb{p}^\infty[\mathbf{S}^\infty[\![\mathtt{P}]\!]] = \mathtt{P}$.

Let $\mathcal{T} \subseteq \mathfrak{T}^\infty$ and $\sigma \in \mathcal{T}$. For all $0 \leq i < \#\sigma$, let $\sigma_i = \langle \rho_i,\ \mathtt{C}_i\rangle$. By definition of $\mathbb{p}^\infty$, we have $\{\mathtt{C}_i \mid 0 \leq i < \#\sigma\} \subseteq \mathbb{p}^\infty[\mathcal{T}]$. Moreover, if σ is finite so that $n = \#\sigma > 0$, we have $\mathbf{S}[\![\mathbb{C}]\!]\sigma_{n-1} = \emptyset = \mathbf{S}[\![\mathbb{p}^\infty[\mathcal{T}]]\!]\sigma_{n-1}$ since $\mathtt{C}_{n-1} \in \mathbb{p}^\infty[\mathcal{T}]$. Whether σ is finite or not, we have $\sigma_i \in \mathbf{S}[\![\mathbb{C}]\!]\sigma_{i-1}$ for all $0 \leq i < \#\sigma$. But $\mathtt{C}_{i-1} \in \mathbb{p}^\infty[\mathcal{T}]$ so $\sigma_i \in \mathbf{S}[\![\mathbb{p}^\infty[\mathcal{T}]]\!]\sigma_{i-1}$. It follows that $\sigma \in \mathbf{S}^\infty[\![\mathbb{p}^\infty[\mathcal{T}]]\!]$ proving that $\mathcal{T} \subseteq \mathbf{S}^\infty[\![\mathbb{p}^\infty[\mathcal{T}]]\!]$. □

4.2 Trace First Command Collection

Let us define the *first command of a trace* as:

$$\begin{array}{ll} \mathbb{p}_0^\infty \in \wp(\mathfrak{T}^\infty) \longmapsto \mathbb{P} & \mathbb{p}_0 \in \mathfrak{T}^\infty \longmapsto \mathbb{C} \\ \mathbb{p}_0^\infty[\mathcal{T}] \triangleq \{\mathbb{p}_0[\sigma] \mid \sigma \in \mathcal{T}\} & \mathbb{p}_0[\sigma] \triangleq \mathbb{p}[\sigma_0] \end{array} \tag{12}$$

Observe that if $\mathcal{T}$ is suffix-closed then $\mathbb{p}^\infty[\mathcal{T}] = \mathbb{p}_0^\infty[\mathcal{T}]$. It immediately follows from (12) and (1) that:

$$\mathsf{po}\langle \wp(\mathfrak{T}^\infty);\ \subseteq\rangle \underset{\mathbb{p}_0^\infty}{\overset{\gamma_0^\infty}{\leftrightarrows}} \mathsf{po}\langle \mathbb{P};\ \subseteq\rangle \tag{13}$$

where $\gamma_0^\infty[\mathtt{Q}] \triangleq \{\sigma \in \mathfrak{T}^\infty \mid \mathbb{p}_0[\sigma] \in \mathtt{Q}\}$.

Moreover, for transformations which eliminate commands from the subject programs, we can use the following correspondence between suffix-closed sets of traces and programs:

Corollary 4.2 *For all programs* $\mathtt{P} \in \mathbb{P}$*, we have:*

$$\mathsf{po}\langle \{\mathcal{T} \subseteq \Sigma^\infty[\![\mathtt{P}]\!] \mid \mathcal{T}^+ = \mathcal{T}\};\ \subseteq\rangle \underset{\mathbb{p}_0^\infty}{\overset{\mathbf{S}^\infty}{\leftrightarrows\!\!\!\!\!\!\rightarrow}} \mathsf{po}\langle \wp(\mathtt{P});\ \subseteq\rangle$$

Proof. For all $\mathcal{T} \subseteq \Sigma^\infty[\![\mathtt{P}]\!]$ such that $\mathcal{T}^+ = \mathcal{T}$ and $\mathtt{Q} \subseteq \mathtt{P}$, we have:

$$
\begin{array}{ll}
& \mathbb{p}_0^\infty[\mathcal{T}] \subseteq \mathtt{Q} \\
\Leftrightarrow & \quad \wr \mathbb{p}_0^\infty[\mathcal{T}] = \mathbb{p}^\infty[\mathcal{T}] \text{ since } \mathcal{T} \text{ is suffix-closed} \int \\
& \mathbb{p}^\infty[\mathcal{T}] \subseteq \mathtt{Q} \\
\Leftrightarrow & \quad \wr \text{by Lem. 4.1} \int \\
& \mathcal{T} \subseteq \mathtt{S}^\infty[\![\mathtt{Q}]\!] \ .
\end{array}
$$

□

5 Blocking Command Elimination

In the following, we consider the *blocking code elimination*, which consists in eliminating blocking commands other than stop commands. The final iterative algorithm is trivial but this case study is simple enough to exemplify the design of correct program transformations by abstract interpretation. In particular the iterative nature of the blocking code elimination algorithm follows from the fixpoint definition (10) of the trace semantics.

5.1 Introduction to Blocking Command Elimination

A command `C` of the form $\mathtt{L}_1 : \mathtt{A} \rightarrow \mathtt{L}_2;$ of a program `P` is *semantically blocking* if and only if it has no possible successor for all evaluation environments (formally $\mathtt{S}[\![\mathtt{P}]\!]\langle \rho, \mathtt{C} \rangle = \emptyset$ for all environments $\rho \in \mathfrak{E}$ that can be encountered when executing command `C` in program `P`). We have singled out a `stop` command `L : stop;` corresponding to a normally expected termination. Other blocking commands are considered as undesirable (for example they might correspond to some abnormal termination such as e.g. a runtime error freezing the computer screen). The use of such undesirable semantically blocking commands may be considered as bad program design, and a removal function (preferably an algorithm) $\mathtt{t}_\flat[\mathtt{P}]$ would be useful to eliminate blocking commands or to check that a program $\mathtt{P} = \mathtt{t}_\flat[\mathtt{P}]$ is well designed according to this criterion. Non-terminating program behaviors should be left unchanged. Because of tests and iteration, the problem is obviously undecidable so that any effective algorithm $\mathbb{t}_\flat$ is necessarily an approximation of function $\mathtt{t}_\flat$. For example:

$$
\mathbb{t}_\flat \left[\begin{array}{l} 1 : \mathtt{false} \rightarrow 1; \\ 2 : \mathtt{skip} \rightarrow 3; \\ 3 : \mathtt{skip} \rightarrow 5; \\ 4 : \mathtt{stop}; \end{array} \right] = \begin{array}{l} 1 : \mathtt{false} \rightarrow 1; \\ 4 : \mathtt{stop}; \end{array}
$$

since the command $3 : \mathtt{skip} \rightarrow 5;$ and therefore $2 : \mathtt{skip} \rightarrow 3;$ are blocking. The command $1 : \mathtt{false} \rightarrow 1;$ is also blocking but is not removed by the syntactic blocking command elimination algorithm $\mathbb{t}_\flat$. This is because it is in general not decidable whether `B` is false in the command $1 : \mathtt{B} \rightarrow 1;$. So the syntactic elimination algorithm $\mathbb{t}_\flat$ only gets rid of syntactically blocking

commands where a command $\mathtt{C}$ of the form $\mathtt{L}_1 : \mathtt{A} \rightarrow \mathtt{L}_2;$ of a program $\mathtt{P}$ is *syntactically blocking* if $\mathtt{L}_2 \notin \mathsf{labels}[\![\mathtt{P}]\!]$. The command $1 : \mathtt{false} \rightarrow 1;$ would have been eliminated by the incomputable semantic elimination function $\mathtt{t}_\flat$. In that sense, $\mathtt{t}_\flat$ is an abstraction of $\mathtt{t}_\flat$.

Obviously a preliminary static program analysis could also be used to determine a larger subset of the semantically blocking actions by taking values of variables into account (e.g. by using the constant propagation static analysis [15]). We do not consider this more refined offline transformation because infinitely many such variants of $\mathtt{t}_\flat$ can be designed and we choose the simplest one to illustrate our purpose.

5.2 Semantic Blocking Trace Elimination

The semantic blocking trace elimination is:

$$\mathtt{t}_\flat \in \wp(\Sigma^\infty) \longmapsto \wp(\Sigma^\infty)$$

$$\mathtt{t}_\flat[\mathcal{T}] \triangleq (\mathcal{T} \cap \Sigma^\omega) \cup \{\sigma \in \mathcal{T} \mid C_\flat[\sigma]\} \tag{14}$$

$$C_\flat[\sigma] \triangleq \sigma \in \Sigma^+ \wedge \mathsf{action}[\![\mathsf{last}[\sigma]]\!] = \mathtt{stop} \tag{15}$$

where $\mathsf{last}[\sigma]$ denotes the command $\mathtt{C} = \mathsf{last}[\sigma]$ in the last state $\langle \rho,\ \mathtt{C}\rangle = \sigma_{\#\sigma-1}$ of the finite trace $\sigma \in \Sigma^+$ of length $\#\sigma$.

We define:

$$\gamma_{\mathtt{t}_\flat} \in \wp(\Sigma^\infty) \longmapsto \wp(\Sigma^\infty)$$

$$\gamma_{\mathtt{t}_\flat}[\mathcal{Y}] \triangleq \mathcal{Y} \cup \{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \tag{16}$$

so that:

Lemma 5.1

$$\mathsf{po}\langle \wp(\Sigma^\infty);\ \subseteq\rangle \underset{\mathtt{t}_\flat}{\overset{\gamma_{\mathtt{t}_\flat}}{\leftrightarrows}} \mathsf{po}\langle \wp(\Sigma^\infty);\ \subseteq\rangle$$

Proof.

$\mathtt{t}_\flat[\mathcal{X}] \subseteq \mathcal{Y}$

$\Leftrightarrow$ $\wr$def. (14) of $\mathtt{t}_\flat\int$

$(\mathcal{X} \cap \Sigma^\omega) \cup \{\sigma \in \mathcal{X} \mid C_\flat[\sigma]\} \subseteq \mathcal{Y}$

$\Leftrightarrow$ $\wr$def. lubs, def. intersection and $\mathcal{X} \subseteq \Sigma^\infty\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq \mathcal{Y} \wedge (\mathcal{X} \cap \{\sigma \in \Sigma^\infty \mid C_\flat[\sigma]\}) \subseteq \mathcal{Y}$

$\Leftrightarrow$ $\wr C_\flat[\sigma] \Rightarrow \sigma \in \Sigma^+$ by def. (15) of $C_\flat\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq \mathcal{Y} \wedge ((\mathcal{X} \cap \Sigma^+) \cap \{\sigma \in \Sigma^\infty \mid C_\flat[\sigma]\}) \subseteq \mathcal{Y}$

$\Leftrightarrow$ $\wr A \cap B \subseteq C$ if and only if $A \subseteq (\neg B \cup C)\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq \mathcal{Y} \wedge (\mathcal{X} \cap \Sigma^+) \subseteq (\neg\{\sigma \in \Sigma^\infty \mid C_\flat[\sigma]\} \cup \mathcal{Y})$

$\Leftrightarrow$ $\wr(X \cap B) \subseteq Y$ if and only if $(X \cap B) \subseteq (Y \cap B)\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq (\mathcal{Y} \cap \Sigma^\omega) \wedge (\mathcal{X} \cap \Sigma^+) \subseteq ((\neg\{\sigma \in \Sigma^\infty \mid C_\flat[\sigma]\} \cup \mathcal{Y}) \cap \Sigma^+)$

$\Leftrightarrow$ $\wr$def. complement$\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq (\mathcal{Y} \cap \Sigma^\omega) \wedge (\mathcal{X} \cap \Sigma^+) \subseteq ((\{\sigma \mid \sigma \notin \Sigma^\infty \vee \neg C_\flat[\sigma]\} \cup \mathcal{Y}) \cap \Sigma^+)$

$\Leftrightarrow$ $\wr\Sigma^+ \subseteq \Sigma^\infty$ and def. intersection$\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq (\mathcal{Y} \cap \Sigma^\omega) \wedge (\mathcal{X} \cap \Sigma^+) \subseteq ((\{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \cup \mathcal{Y}) \cap \Sigma^+)$

$\Leftrightarrow$ $\wr\Sigma^+ \cap \Sigma^\omega = \emptyset\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq ((\{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \cup \mathcal{Y}) \cap \Sigma^\omega) \wedge (\mathcal{X} \cap \Sigma^+) \subseteq ((\{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \cup \mathcal{Y}) \cap \Sigma^+)$

$\Leftrightarrow$ $\wr(X \cap B) \subseteq Y$ if and only if $(X \cap B) \subseteq (Y \cap B)\int$

$(\mathcal{X} \cap \Sigma^\omega) \subseteq (\{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \cup \mathcal{Y}) \wedge (\mathcal{X} \cap \Sigma^+) \subseteq (\{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \cup \mathcal{Y})$

$\Leftrightarrow$ $\wr$def. lubs$\int$

$(\mathcal{X} \cap \Sigma^\omega) \cup (\mathcal{X} \cap \Sigma^+) \subseteq (\{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \cup \mathcal{Y})$

$\Leftrightarrow$ $\wr\mathcal{X} \subseteq \Sigma^\infty = \Sigma^+ \cup \Sigma^\omega\int$

$\mathcal{X} \subseteq (\{\sigma \in \Sigma^+ \mid \neg C_\flat[\sigma]\} \cup \mathcal{Y})$

$\Leftrightarrow$ $\wr$def. (16) of $\gamma_{\mathsf{t}_\flat}\int$

$\mathcal{X} \subseteq \gamma_{\mathsf{t}_\flat}[\mathcal{Y}]$

□

Intuitively Lem. 5.1 states that *the transformed semantics is an abstraction of the subject semantics*. This corresponds to the idea that the program transformation looses some information on the original program. For example the elimination of blocking commands looses all behavior about blocking program behaviors, constant propagation looses all information about how constants are computed, partial evaluation looses all information on program computations for input values other than the ones for which the program is specialized, etc.

Let $1_S \triangleq \boldsymbol{\lambda}\, x \in S \bullet x$ be the identity operator on a set S and $\mathsf{t}_\flat[\mathcal{T}] \triangleq \{\mathsf{t}_\flat[\sigma] \mid \sigma \in \mathcal{T}\}$ be the right image of $\mathcal{T}$ by $\mathsf{t}_\flat$. We have:

Lemma 5.2 *If $\mathcal{T} \subseteq \Sigma^\infty$ then:*

$$\mathsf{po}\langle \wp(\mathsf{t}_\flat[\mathcal{T}]);\, \subseteq\rangle \underset{1_{\wp(\mathsf{t}_\flat[\mathcal{T}])}}{\overset{\mathsf{t}_\flat}{\leftleftarrows}} \mathsf{po}\langle \wp(\mathcal{T});\, \subseteq\rangle$$

Proof. Observe that $\mathsf{t}_\flat$ is a lower closure operator that is reductive ($\forall \mathcal{X} \subseteq \mathcal{T} : \mathsf{t}_\flat[\mathcal{X}] \subseteq \mathcal{X}$), idempotent ($\mathsf{t}_\flat \circ \mathsf{t}_\flat = \mathsf{t}_\flat$) and monotone ($\forall \mathcal{X}, \mathcal{Y} \subseteq \mathcal{T} : (\mathcal{X} \subseteq \mathcal{Y}) \Rightarrow (\mathsf{t}_\flat[\mathcal{X}] \subseteq \mathsf{t}_\flat[\mathcal{Y}])$). It follows that for all $\mathcal{X} \subseteq \mathsf{t}_\flat[\mathcal{T}]$ and $\mathcal{Y} \subseteq \mathcal{T}$, we have:

$1_{\wp(\mathsf{t}_\flat[\mathcal{T}])}(\mathcal{X}) \subseteq \mathcal{Y}$

$\Rightarrow$ $\wr$def. identity$\int$

$\mathcal{X} \subseteq \mathcal{Y}$

$\Rightarrow$ $\wr \mathcal{X} \in \wp(\mathsf{t}_\flat[\mathcal{T}])$ so that there exists $\mathcal{Z} \in \wp(\mathcal{T})$ such that $\mathcal{X} = \mathsf{t}_\flat[\mathcal{Z}] \int$

$\mathsf{t}_\flat[\mathcal{Z}] \subseteq \mathcal{Y}$

$\Rightarrow$ $\wr \mathsf{t}_\flat$ is monotone$\int$

$\mathsf{t}_\flat[\mathsf{t}_\flat[\mathcal{Z}]] \subseteq \mathsf{t}_\flat[\mathcal{Y}]$

$\Rightarrow$ $\wr \mathsf{t}_\flat$ is idempotent$\int$

$\mathsf{t}_\flat[\mathcal{Z}] \subseteq \mathsf{t}_\flat[\mathcal{Y}]$

$\Rightarrow$ $\wr$def. $\mathcal{X} = \mathsf{t}_\flat[\mathcal{Z}]\int$

$\mathcal{X} \subseteq \mathsf{t}_\flat[\mathcal{Y}]$

$\Rightarrow$ $\wr \mathsf{t}_\flat$ is reductive and $\subseteq$ is transitive$\int$

$\mathcal{X} \subseteq \mathcal{Y}$

$\Rightarrow$ $\wr$def. identity$\int$

$1_{\wp(\mathsf{t}_\flat[\mathcal{T}])}(\mathcal{X}) \subseteq \mathcal{Y}$

proving that $1_{\wp(\mathsf{t}_\flat[\mathcal{T}])}(\mathcal{X}) \subseteq \mathcal{Y}$ if and only if $\mathcal{X} \subseteq \mathsf{t}_\flat[\mathcal{Y}]$. Moreover $1_{\wp(\mathsf{t}_\flat[\mathcal{T}])} \in \wp(\mathsf{t}_\flat[\mathcal{T}]) \longmapsto \wp(\mathcal{T})$ is injective. □

It immediately follows from Lem. 5.2 with $\mathcal{T} = \Sigma^\infty[\![\mathtt{P}]\!]$ that:

$$\mathsf{po}\langle \wp(\mathsf{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]]);\ \subseteq\rangle \underset{1_{\wp(\mathsf{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!])}}{\overset{\mathsf{t}_\flat}{\twoheadleftarrow\!\!\!\!\!\!\rightarrow}} \mathsf{po}\langle \wp(\Sigma^\infty[\![\mathtt{P}]\!]);\ \subseteq\rangle,$$

so that by duality:

$$\mathsf{po}\langle \wp(\Sigma^\infty[\![\mathtt{P}]\!]);\ \supseteq\rangle \underset{\mathsf{t}_\flat}{\overset{1_{\wp(\mathsf{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!])}}{\twoheadleftarrow\!\!\!\!\!\!\rightarrow}} \mathsf{po}\langle \wp(\mathsf{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]]);\ \supseteq\rangle\ . \tag{17}$$

The intuition is that $\mathsf{t}_\flat$ is a dual abstraction which can be used to approximate greatest fixpoints from above.

5.3 Observational Abstraction

For a program transformation to be correct, the semantics of the subject and transformed programs should be equivalent at some level of observation. This observational equivalence can be formalized in the abstract interpretation framework by requiring that the abstraction of the semantics of the subject and of the transformed programs into an abstract observation domain should to be identical:

$$\forall \mathtt{P} \in \mathbb{P} : \alpha_{\mathcal{O}}(\mathsf{S}^\infty[\![\mathtt{P}]\!]) = \alpha_{\mathcal{O}}(\mathsf{S}^\infty[\![\mathtt{t}[\![\mathtt{P}]\!]]\!])\ .$$

The specification of the observational abstraction $\alpha_{\mathcal{O}}$ must be considered as part of the problematics (in that it explicitly defines the chosen correctness criterion).

5.4 Observational Abstraction for Blocking Code Elimination

In the particular case of blocking code elimination, the observational abstraction $\alpha_{\mathcal{O}}(\mathcal{T})$ of traces $\mathcal{T}$ is $\mathtt{t}_\flat[\mathcal{T}]$, in that:

- all infinite behaviors of $\mathcal{T}$ are observed in $\mathtt{t}_\flat[\mathcal{T}]$;
- all complete finite behaviors of $\mathcal{T}$ terminating with a `stop` command are observed in $\mathtt{t}_\flat[\mathcal{T}]$;
- no other trace of $\mathcal{T}$ is observed in $\mathtt{t}_\flat[\mathcal{T}]$ so none of the complete finite behaviors terminating of $\mathcal{T}$ with a non-`stop` blocking command is observed in $\mathtt{t}_\flat[\mathcal{T}]$.

5.5 Transformation Design Strategy

Our objective is to constructively derive a blocking code elimination algorithm $\mathtt{t}_\flat$ transforming a subject program P into a transformed program $\mathtt{t}_\flat[\![\mathtt{P}]\!]$ such that P and $\mathtt{t}_\flat[\![\mathtt{P}]\!]$ have equivalent semantics for the $\mathtt{t}_\flat$ observational abstraction:

$$\alpha_{\mathcal{O}}(\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]) = \alpha_{\mathcal{O}}(\mathsf{S}^\infty[\![\mathtt{t}_\flat[\![\mathtt{P}]\!]]\!])$$

this is

$$\mathtt{t}_\flat[\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]] = \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{t}_\flat[\![\mathtt{P}]\!]]\!]]$$

since $\alpha_{\mathcal{O}} = \mathtt{t}_\flat$ for blocking command elimination, hence

$$\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] = \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{t}_\flat[\![\mathtt{P}]\!]]\!]] \ .$$

since $\mathtt{t}_\flat$ is idempotent.

Our design strategy is to first derive the non-blocking trace semantics of programs $\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]$ by abstraction of the trace semantics $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ and then to design the blocking command elimination algorithm $\mathtt{t}_\flat[\![\mathtt{P}]\!]$ as an abstraction of $\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]$.

5.6 Non-Blocking Trace Semantics of Programs

We define the non-blocking trace semantics of a program P as:

$$\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!] \stackrel{\Delta}{=} \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] \ . \tag{18}$$

We observe that $\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]$ is suffix-closed since, by (18) and (14), it contains all infinite execution traces of $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ (which suffix is also an infinite execution trace of $\mathsf{S}^\infty[\![\mathtt{P}]\!]$), the traces s of length 1 reduced to a `stop` command (such that $s^+ = s$) and finite traces of the form $s\sigma$ which are execution traces of $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ which, by (15), end with a `stop` command so that their suffix $s\sigma^+ = \sigma$ is also a finite execution trace of $\mathsf{S}^\infty[\![\mathtt{P}]\!]$ ending with a `stop` command.

In order to express $\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]$ algorithmically as a fixpoint iteration, we can start from the fixpoint form (10) of the program execution trace semantics, such that $\mathsf{S}^\infty_\flat = \mathsf{t}_\flat[\mathsf{gfp}^{\subseteq}_{\Sigma^\infty[\![\mathtt{P}]\!]} \mathsf{F}^\infty[\![\mathtt{P}]\!]]$ where $\mathsf{F}^\infty[\![\mathtt{P}]\!]\mathcal{T} \triangleq \{s \mid \mathsf{S}[\![\mathtt{P}]\!]s = \emptyset\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T}\}$. Then (17) leads to the idea of using the dual of Cor. 2.4 to express $\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]$ in greatest fixpoint form $\mathsf{gfp}^{\subseteq}\mathsf{F}^\infty_\flat[\![\mathtt{P}]\!]$. We have:

— $\top$-strictness

$\mathsf{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]]$ is the $\subseteq$-supremum of $\wp(\mathsf{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]])$;

— Scott co-continuity

By (17), $\mathsf{t}_\flat$ is a complete $\cap$-morphism hence Scott co-continuous;

— For the commutation condition, we have:

$\mathsf{t}_\flat[\mathsf{F}^\infty[\![\mathtt{P}]\!]\mathcal{T}]$

$=$ ⟨By def. (14) of $\mathsf{t}_\flat$⟩

$(\mathsf{F}^\infty[\![\mathtt{P}]\!]\mathcal{T} \cap \Sigma^\omega) \cup \{\sigma \in \mathsf{F}^\infty[\![\mathtt{P}]\!]\mathcal{T} \mid C_\flat[\sigma]\}$

$=$ ⟨By def. (7) of $\mathsf{F}^\infty[\![\mathtt{P}]\!]$⟩

$(\{s \mid \mathsf{S}[\![\mathtt{P}]\!]s = \emptyset\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T}\} \cap \Sigma^\omega) \cup \{\sigma \in \{s \mid \mathsf{S}[\![\mathtt{P}]\!]s = \emptyset\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T}\} \mid C_\flat[\sigma]\}$

$=$ ⟨def. Σ^ω and $\cup$⟩

$\{s \mid \mathsf{S}[\![\mathtt{P}]\!]s = \emptyset \wedge C_\flat[s]\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T} \cap \Sigma^\omega\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T} \wedge C_\flat[\sigma]\}$

$=$ ⟨def. (15) of $C_\flat[\sigma]$⟩

$\{s \mid \mathsf{S}[\![\mathtt{P}]\!]s = \emptyset \wedge \exists \rho, \mathtt{L} : s = \langle \rho, \mathtt{L} : \mathtt{stop;}\rangle\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in (\mathcal{T} \cap \Sigma^\omega) \cup \{\sigma' \in \mathcal{T} \mid C_\flat[\sigma']\}\}$

$=$ ⟨def. (4) of $\mathsf{S}[\![\mathtt{P}]\!]$ and (14) of $\mathsf{t}_\flat$⟩

$\{\langle \rho, \mathtt{L} : \mathtt{stop;}\rangle \mid \mathtt{L} : \mathtt{stop;} \in \mathtt{P} \wedge \rho \in \mathfrak{E}\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathsf{t}_\flat[\mathcal{T}]\}$

$= \mathsf{F}^\infty_\flat[\![\mathtt{P}]\!] \circ \mathsf{t}_\flat[\mathcal{T}]$

by defining:

$$\mathsf{F}^\infty_\flat[\![\mathtt{P}]\!] \triangleq \boldsymbol{\lambda}\mathcal{T} \bullet \{\langle \rho, \mathtt{L} : \mathtt{stop;}\rangle \mid \mathtt{L} : \mathtt{stop;} \in \mathtt{P} \wedge \rho \in \mathfrak{E}\} \cup \{s\sigma \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T}\} \quad (19)$$

We conclude, by the dual of Cor. 2.4, that:

$$\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!] \triangleq \mathsf{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] = \mathsf{t}_\flat[\mathsf{gfp}^{\subseteq}_{\Sigma^\infty[\![\mathtt{P}]\!]} \mathsf{F}^\infty[\![\mathtt{P}]\!]] = \mathsf{gfp}^{\subseteq}_{\mathsf{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]]} \mathsf{F}^\infty_\flat[\![\mathtt{P}]\!] \quad (20)$$

5.7 Blocking Command Elimination Algorithm

We can now design the syntactic blocking command elimination algorithm $\mathtt{t}_\flat[\![\mathtt{P}]\!]$ as an upper approximation of the non-blocking trace semantics of programs:

$$\mathtt{t}_\flat[\![\mathtt{P}]\!] \supseteq \mathtt{p}^\infty[\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]] = \mathtt{p}^\infty_0[\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]] = \mathtt{p}^\infty_0[\mathsf{gfp}^{\subseteq}_{\mathtt{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]]}\mathsf{F}^\infty_\flat[\![\mathtt{P}]\!]]$$

since $\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]$ is suffix-closed and by (20). Then (13) leads to the idea of using Th. 2.7 to constructively derive the algorithm $\mathtt{t}_\flat[\![\mathtt{P}]\!]$. For all $\mathcal{T} \subseteq \Sigma^\infty[\![\mathtt{P}]\!]$, we have:

$$\begin{array}{ll}
& \mathtt{p}^\infty_0[\mathsf{F}^\infty_\flat[\![\mathtt{P}]\!]\mathcal{T}] \\
= & \quad \wr\text{def. (12) of } \mathtt{p}^\infty_0\wr \\
& \{\mathtt{p}_0[\sigma] \mid \sigma \in \mathsf{F}^\infty_\flat[\![\mathtt{P}]\!]\mathcal{T}\} \\
= & \quad \wr\text{def. (19) of } \mathsf{F}^\infty_\flat[\![\mathtt{P}]\!]\wr \\
& \{\mathtt{p}_0[\langle \rho,\, \mathtt{L : stop;}\rangle] \mid \mathtt{L : stop;} \in \mathtt{P} \wedge \rho \in \mathfrak{E}\} \cup \{\mathtt{p}_0[s\sigma] \mid \sigma_0 \in \mathsf{S}[\![\mathtt{P}]\!]s \wedge \sigma \in \mathcal{T}\}, \\
= & \quad \wr\text{def. (12) of } \mathtt{p}_0 \text{ and (11) of } \mathtt{p},\ s, \sigma_0 \in \mathfrak{S}[\![\mathtt{P}]\!] \text{ and def. (3) of } \mathfrak{S}[\![\mathtt{P}]\!] \text{ so} \\
& \quad\ \text{that } s = \langle \rho,\, \mathtt{C}\rangle,\ \sigma_0 = \langle \rho',\, \mathtt{C}'\rangle \text{ and } \sigma = \sigma_0\sigma'\wr \\
& \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathtt{P}\} \cup \{\mathtt{p}[\langle \rho,\, \mathtt{C}\rangle] \mid \exists \rho' \in \mathfrak{E} : \exists \sigma' \in \Sigma^\infty[\![\mathtt{P}]\!] : \exists \mathtt{C}' \in \mathbb{C} : \\
& \quad \langle \rho',\, \mathtt{C}'\rangle \in \mathsf{S}[\![\mathtt{P}]\!]\langle \rho,\, \mathtt{C}\rangle \wedge \langle \rho',\, \mathtt{C}'\rangle\sigma' \in \mathcal{T}\} \\
= & \quad \wr\text{def. (11) of } \mathtt{p} \text{ and (4) of } \mathsf{S}[\![\mathtt{P}]\!]\wr \\
& \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathtt{P}\} \cup \{\mathtt{C} \in \mathtt{P} \mid \exists \rho' \in \mathfrak{E} : \exists \sigma' \in \Sigma^\infty[\![\mathtt{P}]\!] : \exists \mathtt{C}' \in \mathtt{P} : \\
& \quad \rho' \in \mathsf{S}[\![\mathsf{action}[\![\mathtt{C}]\!]]\!]\rho \wedge \mathsf{label}[\![\mathtt{C}']\!] \in \mathsf{succ}[\![\mathtt{C}]\!] \wedge \langle \rho',\, \mathtt{C}'\rangle\sigma' \in \mathcal{T}\} \\
\subseteq & \quad \wr\text{Ignoring the (maybe undecidable) condition } \rho' \in \mathsf{S}[\![\mathsf{action}[\![\mathtt{C}]\!]]\!]\rho\wr \\
& \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathtt{P}\} \cup \{\mathtt{C} \in \mathtt{P} \mid \exists \rho' \in \mathfrak{E} : \exists \sigma' \in \Sigma^\infty[\![\mathtt{P}]\!] : \exists \mathtt{C}' \in \mathtt{P} : \\
& \quad \mathsf{label}[\![\mathtt{C}']\!] \in \mathsf{succ}[\![\mathtt{C}]\!] \wedge \langle \rho',\, \mathtt{C}'\rangle\sigma' \in \mathcal{T}\} \\
= & \quad \wr\text{def. (12) of } \mathtt{p}_0\wr \\
& \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathtt{P}\} \cup \\
& \quad \{\mathtt{C} \in \mathtt{P} \mid \exists \mathtt{C}' \in \mathtt{P} : \mathsf{label}[\![\mathtt{C}']\!] \in \mathsf{succ}[\![\mathtt{C}]\!] \wedge \mathtt{C}' \in \mathtt{p}_0[\mathcal{T}]\} \\
= & \quad \wr\text{def. } \cap\wr \\
& \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathtt{P}\} \cup \\
& \quad \{\mathtt{C} \in \mathtt{P} \mid \{\mathsf{label}[\![\mathtt{C}']\!] \mid \mathtt{C}' \in \mathtt{p}_0[\mathcal{T}] \cap \mathtt{P}\} \cap \mathsf{succ}[\![\mathtt{C}]\!] \neq \emptyset\} \\
= & \quad \wr\text{by def. of labels in Sec. 3.1}\wr \\
& \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathtt{P}\} \cup \{\mathtt{C} \in \mathtt{P} \mid \mathsf{succ}[\![\mathtt{C}]\!] \cap \mathsf{labels}[\![\mathtt{p}_0[\mathcal{T}] \cap \mathtt{P}]\!] \neq \emptyset\} \\
= & \ \mathbb{F}_\flat[\![\mathtt{P}]\!] \circ \mathtt{p}_0[\mathcal{T}]
\end{array}$$

by defining:

$$\mathbb{F}_\flat[\![\mathtt{P}]\!] \stackrel{\Delta}{=} \boldsymbol{\lambda}\, \mathcal{Q} \bullet \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathtt{P}\} \cup \{\mathtt{C} \in \mathtt{P} \mid \mathsf{succ}[\![\mathtt{C}]\!] \cap \mathsf{labels}[\![\mathcal{Q} \cap \mathtt{P}]\!] \neq \emptyset\}, \tag{21}$$

Moreover $\mathbb{p}_0[\mathtt{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]]] = \mathtt{P}$ so by (13) and Th. 2.7, we conclude that:

$$\mathtt{t}_\flat[\![\mathtt{P}]\!] \stackrel{\Delta}{=} \mathsf{gfp}^{\subseteq}_{\mathtt{P}} \mathbb{F}_\flat[\![\mathtt{P}]\!] \supseteq \mathbb{p}_0^\infty[\mathsf{gfp}^{\subseteq}_{\mathtt{t}_\flat[\Sigma^\infty[\![\mathtt{P}]\!]]} \mathsf{F}_\flat^\infty[\![\mathtt{P}]\!]] = \mathbb{p}^\infty[\mathsf{S}_\flat^\infty[\![\mathtt{P}]\!]] \ . \tag{22}$$

All iterates of $\mathsf{gfp}^{\subseteq}_{\mathtt{P}} \mathbb{F}_\flat[\![\mathtt{P}]\!]$ are included in $\mathtt{P}$ so that we have

$$\mathtt{t}_\flat[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\mathtt{P}} \mathbb{F}'_\flat[\![\mathtt{P}]\!]$$

with

$$\mathbb{F}_\flat[\![\mathtt{P}]\!] \stackrel{\Delta}{=} \boldsymbol{\lambda}\, \mathcal{Q} \bullet \{\mathtt{L : stop;} \mid \mathtt{L : stop;} \in \mathcal{Q}\} \cup \{\mathtt{C} \in \mathcal{Q} \mid \mathsf{succ}[\![\mathtt{C}]\!] \cap \mathsf{labels}[\![\mathcal{Q}]\!] \neq \emptyset\},$$

Observe that $\mathsf{po}\langle \mathtt{P}; \subseteq \rangle$ satisfies the descending chain condition so that the above fixpoint form of $\mathtt{t}_\flat[\![\mathtt{P}]\!]$ immediately leads to an effective iteration algorithm, that we can describe informally as follows:

- Start from $\mathtt{Q} := \mathtt{P}$;
- Repeat
  Suppress the commands $\mathtt{C}$ from $\mathtt{Q}$ such that $\mathtt{C} \neq \mathtt{L : stop;}$ and $\mathsf{succ}[\![\mathtt{C}]\!] \cap \mathsf{labels}[\![\mathtt{Q}]\!] = \emptyset$;
  Until $\mathtt{Q}$ is left unchanged;
- Return $\mathtt{Q}$.

5.8 Correctness of the Blocking Command Elimination Algorithm

The correctness of the transformation is stated by the fact that the observation of the semantics of the subject and transformed programs by the observational abstraction $\alpha_{\mathcal{O}} = \mathtt{t}_\flat$ is the same. Formally, $\alpha_{\mathcal{O}}(\mathsf{S}^\infty[\![\mathtt{P}]\!]) = \alpha_{\mathcal{O}}(\mathsf{S}^\infty[\![\mathtt{t}_\flat[\![\mathtt{P}]\!]]\!])$, that is

$$\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] = \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{t}_\flat[\![\mathtt{P}]\!]]\!]] \ .$$

Proof. By Lem. 4.1, S^∞ is monotone. By (17), $\mathtt{t}_\flat$ is monotone. By (22), we have $\mathtt{t}_\flat[\![\mathtt{P}]\!] = \mathsf{gfp}^{\subseteq}_{\mathtt{P}} \mathbb{F}_\flat[\![\mathtt{P}]\!]$ so $\mathtt{t}_\flat[\![\mathtt{P}]\!] \subseteq \mathtt{P}$. By monotony we conclude that $\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{t}_\flat[\![\mathtt{P}]\!]]\!]] \subseteq \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]$.

By Lem. 4.1, $\mathsf{S}^\infty \circ \mathbb{p}^\infty$ is extensive so that $\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] \subseteq \mathsf{S}^\infty[\![\mathbb{p}^\infty[\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]]]\!]$.

By (22), $\mathbb{p}^\infty[\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]] \subseteq \mathbb{t}_\flat[\![\mathtt{P}]\!]$. By Lem. 4.1, S^∞ is monotone. So by monotony, $\mathsf{S}^\infty[\![\mathbb{p}^\infty[\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!]]]\!] \subseteq \mathsf{S}^\infty[\![\mathbb{t}_\flat[\![\mathtt{P}]\!]]\!]$. By (18), $\mathsf{S}^\infty_\flat[\![\mathtt{P}]\!] \triangleq \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]$ so that we have $\mathsf{S}^\infty[\![\mathbb{p}^\infty[\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]]]\!] \subseteq \mathsf{S}^\infty[\![\mathbb{t}_\flat[\![\mathtt{P}]\!]]\!]$.

By transitivity, $\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] \subseteq \mathsf{S}^\infty[\![\mathbb{t}_\flat[\![\mathtt{P}]\!]]\!]$. By Lem. 5.2, $\mathtt{t}_\flat$ is monotone and idempotent so $\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] = \mathtt{t}_\flat[\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]]] \subseteq \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathbb{t}_\flat[\![\mathtt{P}]\!]]\!]]$.

By antisymmetry, we conclude that $\mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathtt{P}]\!]] = \mathtt{t}_\flat[\mathsf{S}^\infty[\![\mathbb{t}_\flat[\![\mathtt{P}]\!]]\!]]$. □

6 Conclusion

The general idea to formalize program transformation by abstract interpretation is to define a semantic transformation as an abstraction of the subject program semantics. This transformation is an abstraction in that the transformed semantics has lost some information on the subject semantics (e.g. the existence of blocking traces). The correctness of the semantic transformation is proved using an observational abstraction specifying which details about the subject and transformed semantics should be abstracted away to consider them as equivalent. Then the syntactic – source to source – program transformation is constructively derived by abstraction of transformed semantics into a transformed program. This new approach has been illustrated on the simple case of blocking command elimination.

Many more complex examples such as transition compression, constant propagation, partial evaluation, slicing, etc. have to be treated similarly in order to convince that this point of view is quite general. This will probably require the generalization of the present program transformation framework, for example using weaker hypotheses on abstraction in absence of a best approximation [9].

References

[1] S. Abramsky and A. Jung. Domain theory. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Semantic Structures*, volume 3 of *Handbook of Logic in Computer Science*, chapter 1, pages 1–168. Clarendon Press, 1994.

[2] K.R. Apt and G.D. Plotkin. Countable nondeterminism and random assignment. *J. ACM*, 33(4):724–767, Oct. 1986.

[3] R.M. Burstall and J. Darlington. A transformation system for developing recursive programs. *J. ACM*, 24(1):44–67, Jan. 1977.

[4] 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 mathématiques, Université scientifique et médicale de Grenoble, Grenoble, France, 21 Mar. 1978.

[5] P. Cousot. Constructive design of a hierarchy of semantics of a transition system by abstract interpretation. *ENTCS*, 6, 1997.

`http://www.elsevier.nl/locate/entcs/volume6.html`, 25 pages.

[6] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *4th POPL*, pages 238–252, Los Angeles, CA, 1977. ACM Press.

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

[8] P. Cousot and R. Cousot. Systematic design of program analysis frameworks. In *6th POPL*, pages 269–282, San Antonio, TX, 1979. ACM Press.

[9] P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Logic and Comp.*, 2(4):511–547, Aug. 1992.

[10] J.W. de Bakker, J.-J.Ch. Meyer, and J.I. Zucker. On infinite computations in denotational semantics. *Theoret. Comput. Sci.*, 26:53–82, 1983. (Corrigendum: Theoret. Comput. Sci. 29:229–230, 1984).

[11] N. Jones, C.K. Gomard, and P. Sestoft. *Partial Evaluation and Automatic Program Generation*. Int. Series in Computer Science. Prentice-Hall, June 1993.

[12] N.D. Jones. Abstract interpretation and partial evaluation in functional and logic programming. In M. Bruynooghe, editor, *Proc. Int. Symp. ILPS '1994*, pages 17–22. MIT Press, 13–17 Nov. 1994.

[13] N.D. Jones. An introduction to partial evaluation. *ACM Comput. Surv.*, 28(3):480–504, Sep. 1996.

[14] N.D. Jones. Combining abstract interpretation and partial evaluation (brief overview). In P. Van Hentenryck, editor, *Proc. 4th Int. Symp. SAS '97*, Paris, France, 8–10 Sep. 1997, LNCS 1302, pages 396–405. Springer-Verlag, 1997.

[15] G. Kildall. A unified approach to global program optimization. In *1st POPL*, pages 194–206, Boston, MA, Oct. 1973. ACMpress.

[16] S.C. Kleene. Representation of events in nerve nets and finite automata. *Automata Studies*, pages 3–42, 1956.

[17] A. Pnueli, 0. Shtrichman, and M. Siegel. The code validation tool CVT: Automatic verification of a compilation process. *STTT*, 2(2):192–201, 1998.

[18] J.C. Reynolds. The discoveries of continuations. *Lisp and Symbolic Computation*, 6(3/4):233–248, Nov. 1993.

[19] P. Steckler and M. Wand. Selective thunkification. In B. Le Charlier, editor, *Proc. 1st Int. Symp. SAS '94*, Namur, Belgium, 20–22 Sep. 1994, LNCS 864, pages 162–178. Springer-Verlag, 1994.

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

[21] P. Wadler. Deforestation: Transforming programs to eliminate trees. *Theoret. Comput. Sci.*, 73(2):231–248, 28 Mar. 1990.

[22] M. Weiser. Program slicing. *IEEE Trans. Software Engrg.*, SE-10(4):352–357, Jul. 1984.