Computer Science PhD Day, Università Ca' Foscari di Venezia Aula Magna Silvio Trentin, Dorsoduro 3825/e, Venezia, Italy

Termination Proof Inference by Abstract Interpretation

Patrick Cousot

cims.nyu.edu/~pcousot/ www.di.ens.fr/~cousot/

Radhia Cousot

www.di.ens.fr/~rcousot/

Abstract

The existing approaches to termination proof are scattered and largely not comparable with each other.

We introduce a unifying design principle for termination based on an abstract interpretation of a complete infinitary trace semantics. We show that 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.

For (1), we show that the abstract interrpetation-based 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.

Three principles

Principle I

Program verification methods (formal proof or static analysis methods) are abstract interpretations of a semantics of the programming language (***)

^(*) 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.

^(**) P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *POPL*, 269–282, 1979.

Refinement to principle II

Safety as well as termination verification methods are abstract interpretations of a maximal trace semantics of the programming language

Comments on principle II

- This is well-known for instances of safety (like invariance) using prefix trace semantics^(*)
- This is true for full safety
- New for termination

^(*) P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *POPL*, 269–282, 1979.

New principle III

More expressive and powerful verification methods are derived by structuring the trace semantics (into a hierarchy of segments)

Comments on principle III

- Syntactic instances have been known for long (different variant functions for nested loops, Hoare logic for total correctness,...)
- Semantic instances have been ignored for long (Burstall's total correctness proof method using intermittent assertions) and very successful recently (Podelski-Rybalchenko)

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

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

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

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

Maximal trace semantics

9

Maximal trace semantics



Fixpoint maximal trace semantics

Complete lattice

$$\langle \wp(\Sigma^{*\infty}), \sqsubseteq, \Sigma^{\infty}, \Sigma^{*}, \sqcup, \sqcap \rangle$$

- Computational ordering $(T_1 \sqsubseteq T_2) \triangleq (T_1^+ \subseteq T_2^+) \land (T_1^\infty \supseteq T_2^\infty) \quad T^+ \triangleq T \cap \Sigma^+$ $(T_1 \sqcup T_2) \triangleq (T_1^+ \cup T_2^+) \cup (T_1^\infty \cap T_2^\infty) \quad T^\infty \triangleq T \cap \Sigma^\infty$
- Fixpoint semantics

$$\tau^{+\infty} \llbracket \mathbf{P} \rrbracket = \mathsf{lfp}_{\Sigma^{\infty}}^{\sqsubseteq} \overleftarrow{\phi}_{\tau}^{+\infty} \llbracket \mathbf{P} \rrbracket$$
$$= \mathsf{lfp}_{\emptyset}^{\subseteq} \overleftarrow{\phi}_{\tau}^{+} \llbracket \mathbf{P} \rrbracket \cup \mathsf{gfp}_{\Sigma^{\infty}}^{\subseteq} \overleftarrow{\phi}_{\tau}^{\infty} \llbracket \mathbf{P} \rrbracket$$
$$\overleftarrow{\phi}_{\tau}^{+\infty} \llbracket \mathbf{P} \rrbracket T \triangleq \beta_{\tau} \llbracket \mathbf{P} \rrbracket \sqcup \tau \llbracket \mathbf{P} \rrbracket \operatorname{}^{\ominus} T$$

Patrick Cousot, Radhia Cousot: Inductive Definitions, Semantics and Abstract Interpretation. POPL 1992: 83-94

(Trace) properties

Program properties

• A program property P is the set of semantics which have this property:

$$P \in \mathcal{O}(\mathcal{O}(\Sigma^{+\infty}|))$$

- Example: $P = \begin{bmatrix} \bullet \bullet 0 \\ \bullet \bullet \bullet \bullet 0 \end{bmatrix} \begin{bmatrix} \bullet \bullet \bullet I \\ \bullet \bullet \bullet \bullet \bullet 0 \end{bmatrix}$
- Strongest property of program P :

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

 $\{ au^{+\infty}\}$

Trace property abstraction

• Trace property abstraction:

$$\alpha_{\Theta}(P) \triangleq \bigcup P \qquad \langle \wp(\wp(\Sigma^{+\infty})), \subseteq \rangle \xleftarrow{\gamma_{\Theta}}{\alpha_{\Theta}} \langle \wp(\Sigma^{+\infty}), \subseteq \rangle$$



- The strongest trace property of a trace semantics is this trace semantics $\alpha_{\Theta}(\{\tau^{+\infty} [\![P]\!]\}) = \tau^{+\infty} [\![P]\!]$
- Safety/liveness (termination) are trace properties, not general program properties

The Termination Problem

The termination proof problem

• Termination abstraction:

$$\alpha^t(T) \triangleq T \cap \Sigma^+$$

• Termination proof:

$$\alpha^{t}(\tau^{+\infty}\llbracket \mathbf{P} \rrbracket) = \tau^{+\infty}\llbracket \mathbf{P} \rrbracket$$

 Termination proofs are not very useful since programs do not *always* terminate

Example

• Arithmetic mean of integers x and y

• Does not *always* terminate e.g.

$$\langle x,y \rangle = \langle |,0 \rangle \rightarrow \langle 0,| \rangle \rightarrow \langle -|,2 \rangle \rightarrow \langle -2,3 \rangle \rightarrow ...$$

Patrick Cousot: Proving Program Invariance and Termination by Parametric Abstraction, Lagrangian Relaxation and Semidefinite Programming. VMCAI 2005: 1-24

The termination inference problem

- Determine a necessary condition for program termination and prove it sufficient
- Example:
 - (I) Under which necessary conditions

does terminate?

• (2) Prove these conditions to be sufficient

The Termination Inference Problem

Potential termination

• For non-deterministic programs, we may be interested in potential termination



Definite termination abstraction

• or in definite termination



 Potential and definite termination coincide for deterministic programs. Only definite termination in this presentation.

Definite termination trace abstraction

• Prefix Abstraction

$$\mathsf{pf}(\sigma) \triangleq \{ \sigma' \in \Sigma^{+\infty} \mid \exists \sigma'' \in \Sigma^{*\infty} : \sigma = \sigma' \sigma'' \}$$
$$\mathsf{pf}(T) \triangleq \bigcup \{ \mathsf{pf}(\sigma) \mid \sigma \in T \} .$$

• Definite termination abstraction

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



Definite termination

 \bullet The semantics/set of traces T definitely terminates if and only if

$$\alpha^{\mathsf{Mt}}(T) = T$$



Finite abstractions do not work

- « Abstract and model-check » is impossible^(*) for termination and unsound for non-termination of unbounded programs
 - Unbounded executions:





• Non-termination (lasso): unsound

^(*) Excluding trivial solutions, see: Patrick Cousot: Partial Completeness of Abstract Fixpoint Checking. SARA 2000: 1-25

Definite termination domain

Reachability analysis

• A forward invariance analysis infers states potentially reachable from initial states (by over-approximating an abstract fixpoint lfp F)^(*)



(*) 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. 26

Venezia, 2012/03/12, Termination proof inference by abstract interpretation

Accessibility analysis

• A backward invariance analysis infers states potentially / definitely accessing final states (by over-approximating an abstract fixpoint lfp B)^(*)



(*) P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *POPL*, 269–282, 1979.

Combined reachability/accessibility analyses

• An iterated forward/backward invariance analysis infers reachable states potentially/definitely accessing final states (by over-approximating $\lim F \sqcap \lim B$) ^(*)



$$egin{array}{rll} X^0 &=& op \ && \dots \ && X^{2n+1} &=& \mathrm{lfp}\,\lambda Y\,.\,X^{2n}\sqcap F(Y) \ X^{2n+2} &=& \mathrm{lfp}\,\lambda Y\,.\,X^{2n+1}\sqcap B(Y) \ && \dots \end{array}$$

- (*) 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.
- (*) P. Cousot & R. Cousot. Abstract interpretation and application to logic programs. J. Log. Program. 13 (2 & 3): 103–179 (1992)

Example

 \bullet Arithmetic mean of two integers x and y

• Necessarily $x \ge y$ for proper termination

Example (cont'd)

• Arithmetic mean of two integers x and y (cont'd)

while $(x \leftrightarrow y)$ {



<u>Hint:</u> imagine \mathbf{k} is the number of remaining steps to be done in the loop

Venezia, 2012/03/12, Termination proof inference by abstract interpretation

Example (cont'd)

• Arithmetic mean of two integers x and y (cont'd)



 The difference <u>x</u> – <u>y</u> must initially be even for proper termination

Observations

- k provides the value of the variant function in the sense of Turing/Floyd
- The constraints on k (hence the variant function) are computed backwards
 - ⇒ a backward analysis should be able to infer the variant function

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

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

The Turing-Floyd termination proof method

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

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

The hierarchy of termination semantics

- Maximal trace concrete backward trace semantics $\alpha^{\rm Mt}$
 - Definite termination abstract backward trace semantics

$$lpha^{\sf W}$$

 $lpha^{\mathsf{rk}}$

Weakest pre-condition abstract backward state semantics (termination domain)

Variant function abstract ordinal backward semantics

The ranking abstraction

$$\begin{array}{rcl} \alpha^{\mathsf{rk}} & \in & \wp(\Sigma \times \Sigma) \mapsto (\Sigma \not \mapsto \mathbb{O}) \\ \alpha^{\mathsf{rk}}(r)s & \triangleq & 0 & \text{when} & \forall s' \in \Sigma : \langle s, \ s' \rangle \notin r \\ \alpha^{\mathsf{rk}}(r)s & \triangleq & \sup\left\{\alpha^{\mathsf{rk}}(r)s' + 1 \mid \exists s' \in \Sigma : \langle s, \ s' \rangle \in r \land \\ & \forall s' \in \Sigma : \langle s, \ s' \rangle \in r \implies s' \in \mathsf{dom}(\alpha^{\mathsf{rk}}(r))\right\} \end{array}$$

- $\alpha^{\rm rk}(r)$ extracts the well-founded part of relation r
- provides the rank of the elements s in its domain
- strictly decreasing with transitions of relation r
- \implies the most precise variant function

Fixpoint definition of the variant function

We now apply the abstract interpretation methodology:

- The maximal trace semantics has a fixpoint definition
- The variant function is an abstraction of the maximal trace semantics
- With this abstraction, we construct a fixpoint definition of the abstract variant semantics
 - \implies Fixpoint induction provides a termination proof method
 - Further abstractions and widenings provide a static analysis method



$$v^{\omega} = \lambda x \in [-\infty, 0] \bullet 0 \, \dot{\cup} \, \lambda x \in [1, +\infty] \bullet (x+1) \div 2 \, .$$

. . .

Computational order on functions



 $\nu \sqsubseteq^{\mathsf{v}} \nu' \triangleq \mathsf{dom}(\nu) \subseteq \mathsf{dom}(\nu') \land \forall x \in \mathsf{dom}(\nu) : \nu(x) \preccurlyeq \nu'(x)$

© P. Cousot & R. Cousot

Example III





Objection I: Turing/Floyd's method goes forward not backward!

An analysis can be inverted using auxiliary variables^(*)

int x, x0; while (c(x)) { x0 := x; x := f(x) }

Backward variant V:Forward $v(x_{before}) = v(x_{after}) + I$ $v(x_{0})$ $\leftrightarrow v(x_{before}) = v(f(x_{before})) + I$ $\leftrightarrow v(x_{0})$

Forward variant V: $v(x_0) = v(x) + 1$ $\iff v(x_0) = v(f(x_0)) + 1$

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

Venezia, 2012/03/12, Termination proof inference by abstract interpretation

Objection II: you need ordinals!"



 To avoid transfinite ordinals/well-founded orders of for unbounded non-determinism, the computations need to be structured!

^(*) R. Floyd. Assigning meaning to programs. *Proc. Symp. in Applied Math.*, Vol. 19, 19–32. Amer. Math. Soc., 1967.

Venezia, 2012/03/12, Termination proof inference by abstract interpretation

Structuring trace semantics with segments

Floyd/Turing termination proof method

• Trivial postfix structuring of traces into segments



 Also used for termination of straight-line code (no need for variant functions)



Hoare logic

• The trace semantics is recursively structured in



C. Houre. An axiomatic basis for computer programming. *Communications of the Association for Computing Machinery*, 12(10):576–580, 1969. Z. Mana and A. Pnueli. Axiomatic approach to total correctness of programs. *Acta Inf.*, 3:243–263, 1974.



{ *P*, *PF*, *PL*, *PLE*, *PLD*, *PLDB*, *PLDC* }

vith widening/narrowing, as considered in this paper, are definitely trictly more powerful than finite abstractions. The computation of variant functions by abstraction is new, and different from the ounter-example guided ways to find disjunctive ranking functions, sed in tools like Terminator [7] and derivatives.

8. Some price do odd(x) a Abstract interpretation has established constructive principles for

easoning about semantics. A semantics is a fix the proving and emantic property at some level of abstraction consists in verifying properties of abstract fixpoints which have to be checked (in hecking/verification methods), guessed (in proof methods), or utomatically inferred or approximated (in static analysis methods).

This principle was mainly applied in the past to invariance and ndirectly to terming only eductor to invariance. We have shown hat the abstract interpretation principle directly applies to both afety (generalizing invariance) and termination.

Moreover we have generalized the classical syntactic structural nduction into the language-independent semantic concept of semanic structural induction based on (abstractions of) inductive trace overs which includes induction on syntax, control states, membry states, and execution trace segments and thus generalizes all *Handsimulation* variables of a program. *POPL*, 84–97, 1978. [32] P. Cousot, R. Cousot, and L. Mauborgne. A scalable segmented decision tree erification and static analysis methods.

This methodology allowed us to establish new principles for $x \ge 2 \land x' = 6200272-95, 2010.$ roving termination by abstract interpretation of a termination emantics. It remains to design a suitable collection of abstract Theorem (sulvent and scalable array content analysis. POPH 105+118, 2011 ion lomains beyond the examples proposed in this paper and the with spicieting/impleinentations idered in this paper, are definitely strictly presenperverfault three preterabet rection at The computation as to efexatione functions by sales in a sine we generate the other it bility counter-example guided ways to find disjunctive ranking functions, used in tools like Terminator [7] and derivatives.

References

$x' \leq 1$

and

18.1. B ConcluBion. and L. Zuck. Modular ranking abstraction. Int. J. Found. Comput. Sciences of 1999 and proving as hand simulation with a little induction. Informa-Comput. Sciences of 1999 Worth Holland, 1974. Destrable of the terms of terms of the terms of t reasonithing bout usern antices pAcesern antities 149a 200 point so proving a saman Burstall n Brogram printing to hand a state by the state induction informa-

- Cousot and R. Couson 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. POPI 269–282, 1979.
- P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems P. X. 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**-1**D**. 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 mevitable 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. Kang 3 95-112, 1094. $\wedge X'$
- [31] P. Cousot and N. Halbwachs. Automatic discovery of linear restraints among
- abstract domain. Time for Verification, Essays in Memory of A. Pnueli, LNCS $\left[\text{odd}(x) \land x \ge 3 \land x' = x+1 \right]$
 - [33] P. Cousot, R. Cousot, and F. Logozzo. A parametric segmentation functor for
 - [34] P. Cousot, R. Cousot, and F. Logozzo. Precondition Inference from intermittent assertions and application to contracts on collections. VMCAI, LNCS 6538, 150-
- [20] P. Cousb68n 201 Cousot. Static determination of dynamic properties of reductive = (x + 1)/2prosponces constraint of the prosponse o
- Holland, 1977.
 Programmes parallèles. Thèse d'État ès sciences math. INPLe Naney, 1985.
 P. Cousot and R. Cousot. Systematic design of program analysis frameworks.
 POPL, B69Dasey19rol. H. Priestley. Introduction to Lattices and Order. 2nd Edition.
- [22] P. Cousof and R. Cousof. Constructive versions of Tarski's fixed point theorems. P[37] f Family Sena: 43 Granderdy commands, nondeterminacy and formal derivation of
- of J&gEanDijKotrk: & Missoiplfne BfcBragranmsing:tBrentice.Hahl:eU906urse
- P. CH39/24 Arene and Chine Bain Militie Pater State Predation States in Predation States and Contract Contract
- [2944] Providence of the set of t
- [25] PL Couset and Ran Courses and metime in always of recursion = always on the DVS Car

Well-founded tree structure of the trace segmentation







Podelski-Rybalchenko

• Transition invariants are abstractions of trace segments covering the trace semantics by their extremities



trace segment

• Termination based on Ramsey theorem on colored edges of a complete graph, no recursive structure

A. Podelski and A. Rybalchenko. Transition invariants. LICS, 32-41, 2004.

F. P. Ramsey. On a problem of formal logic. In Proc. London

Math. Soc., volume 30, pages 264–285, 1930.

Rely-guarantee

 Example of abstraction of segments into relyguarantee/contracts state properties:



Joey W. Coleman, Cliff B. Jones: A Structural Proof of the Soundness of Rely/guarantee Rules. J. Log. Comput. 17(4): 807-841 (2007)

nantics segmentation

egmentation

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 $SS' \in C$ then $S \in C$ (prefix-closure)2. if $S \in C$ then $\exists S' : S = \chi S'$ (root)3. if $SBB' \in C$ then $B \oplus B'$ (well-foundedness)4. if $SBB' \in C$ then $B \subseteq \bigcup_{SBB' \in C} B'$ (cover). \Box

on the possibly infinite but wellmentation tree

oofs on segment sets (using variant theorem, etc.)

Conclusion

Presentation based on our POPL'2012 paper

 Patrick Cousot, Radhia Cousot: An abstract interpretation framework for termination. POPL 2012: 245-258

More in the paper

- The paper provides
 - More topics (e.g. general safety by abstract interpretation, abstract trace covers/proofs)
 - More technical details (e.g. fixpoint definitions of the various abstract termination semantics)
 - More examples (e.g. a more detailed piecewise linear termination abstraction)

Contributions

- Formalization of existing termination proof methods as abstract interpretations
- Pave the way for new backward termination static analysis methods (going beyond reduction of termination to safety analyzes)
- The new concept of trace semantics segmentation is not specific to termination and applies to all specification/verification/analysis methods

Future work

- Abstract domains for termination
- Semantic techniques for segmentation inference
- Eventuality verification/static analysis
- (General) liveness^(*) verification/static analysis

^(*) Beyond LTL, as defined in

Bowen Alpern, Fred B. Schneider: Defining Liveness. Inf. Process. Lett. (IPL) 21(4):181-185 (1985)2EEBowen Alpern, Fred B. Schneider: Defining Liveness. Inf. Process. Lett. (IPL) 21(4):181-185 (1985)

The end, thank you