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.

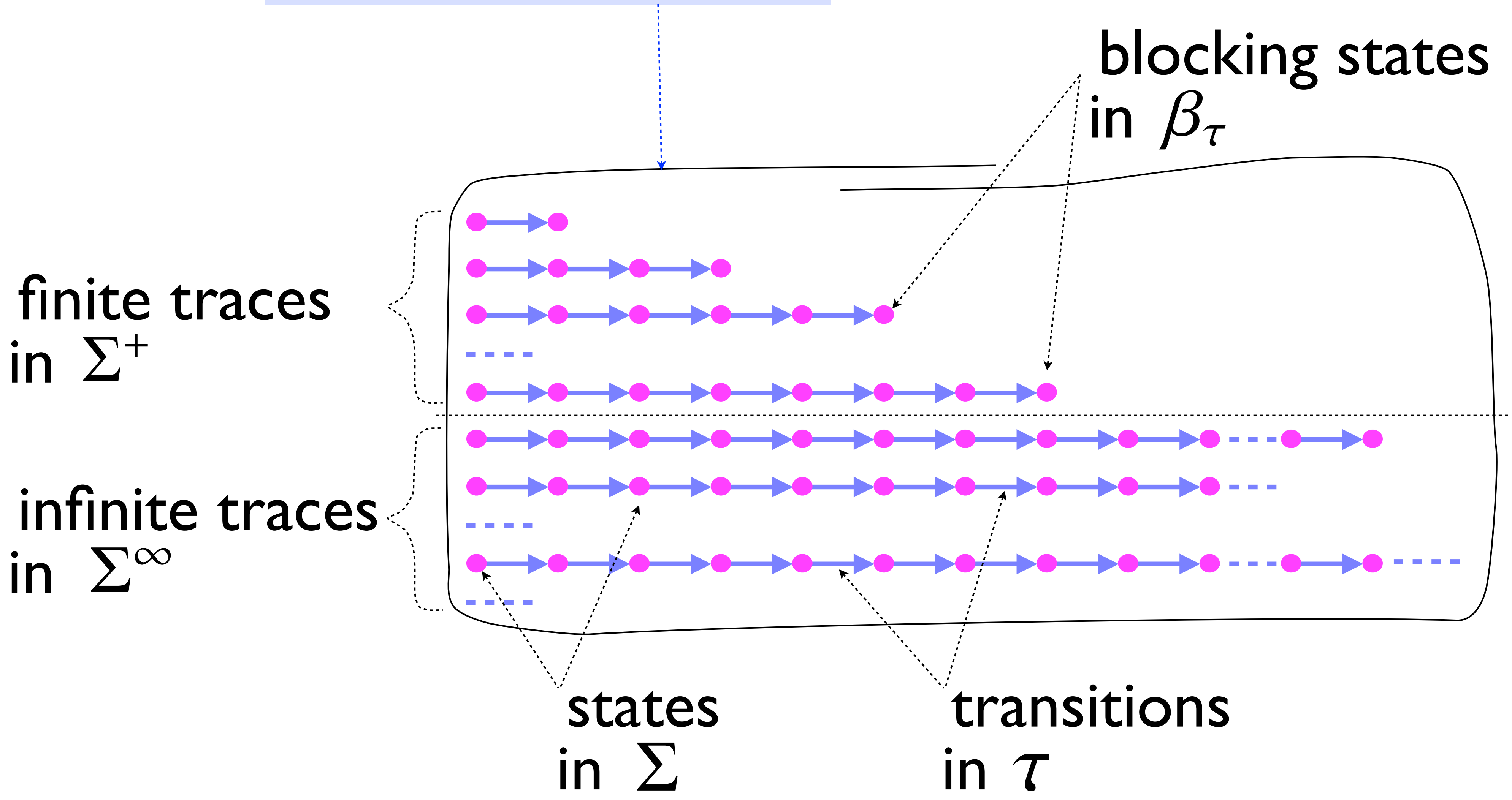
R. Burstall. Program proving as hand simulation with a little induction. *Information Processing*, 308–312. North-Holland, 1974.

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

Maximal trace semantics

Maximal trace semantics

- Program $\mathbb{P} \longmapsto \tau^{+\infty}[\![\mathbb{P}]\!] \in \wp(\Sigma^{+\infty})$



Fixpoint maximal trace semantics

- Complete lattice

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

- Computational ordering

$$(T_1 \sqsubseteq T_2) \triangleq (T_1^+ \subseteq T_2^+) \wedge (T_1^\infty \supseteq T_2^\infty) \qquad T^+ \triangleq T \cap \Sigma^+$$

$$(T_1 \sqcup T_2) \triangleq (T_1^+ \cup T_2^+) \cup (T_1^\infty \cap T_2^\infty) \qquad T^\infty \triangleq T \cap \Sigma^\infty$$

- Fixpoint semantics

$$\tau^{+\infty}[\![\mathsf{P}]\!] = \mathsf{lfp}_{\Sigma^\infty}^{\sqsubseteq} \overleftarrow{\phi}_{\tau}^{+\infty}[\![\mathsf{P}]\!]$$

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

$$\overleftarrow{\phi}_{\tau}^{+\infty}[\![\mathsf{P}]\!]T \triangleq \beta_\tau[\![\mathsf{P}]\!] \sqcup \tau[\![\mathsf{P}]\!] \mathbin{;} T$$

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

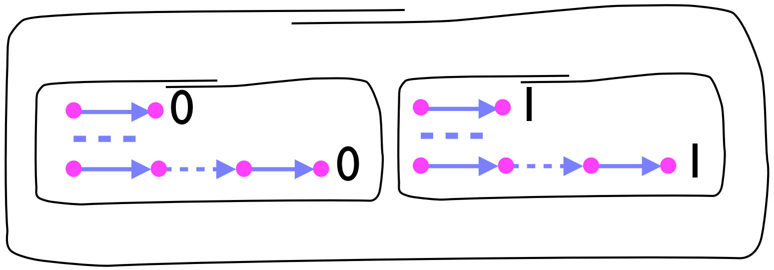
(Trace) properties

Program properties

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

$$P \in \wp(\wp(\Sigma^{+\infty}))$$

- Example:

$P =$ 

- Strongest property of program P :

$$\{\tau^{+\infty}[\![\mathrm{P}]\!]\}$$

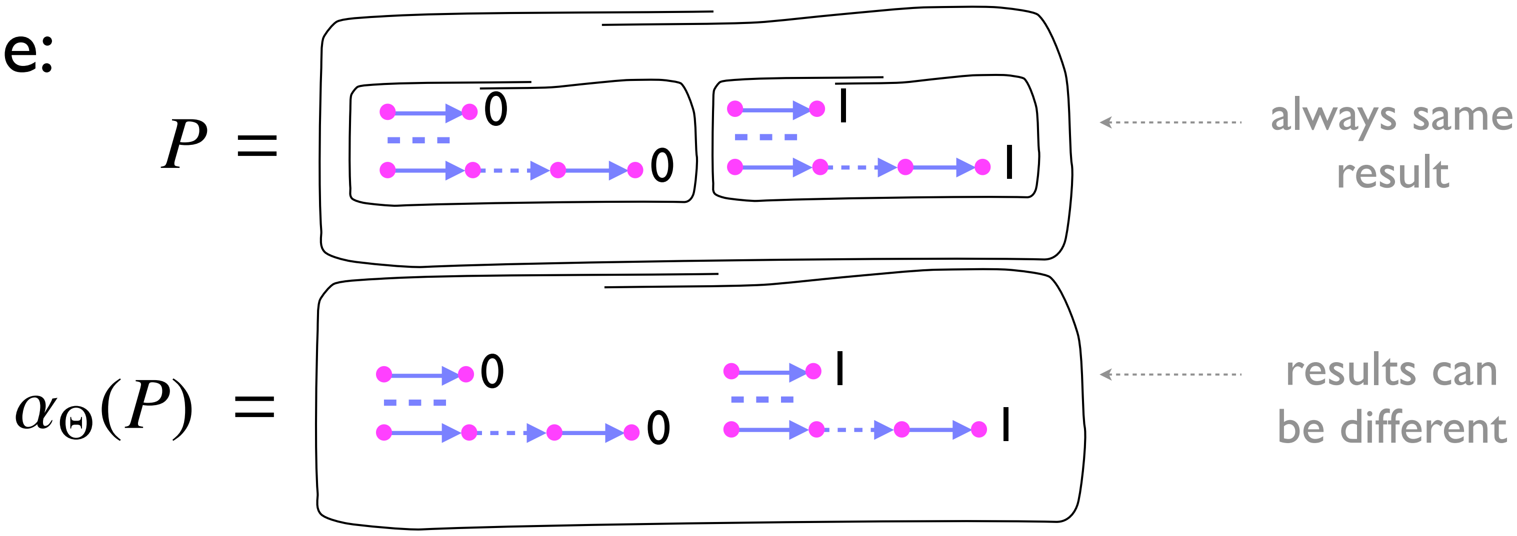
P. Cousot and R. Cousot. Systematic design of program analysis frameworks. *POPL*, 269–282, 1979.

Trace property abstraction

- Trace property abstraction:

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

- Example:



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

The Termination Problem

The termination proof problem

- Termination abstraction:

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

- Termination proof:

$$\alpha^t(\tau^{+\infty}[\![\mathrm{P}]\!]) = \tau^{+\infty}[\![\mathrm{P}]\!]$$

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

Example

- Arithmetic mean of integers `x` and `y`

```
while (x <> y) {
    x := x - 1;
    y := y + 1
}
```

- Does not *always* terminate e.g.

$$\langle x,y\rangle = \langle 1,0\rangle \rightarrow \langle 0,1\rangle \rightarrow \langle -1,2\rangle \rightarrow \langle -2,3\rangle \rightarrow \ldots$$

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:

  - (1) Under which *necessary* conditions

```
while (x <> y) {
    x := x - 1;
    y := y + 1
}
```

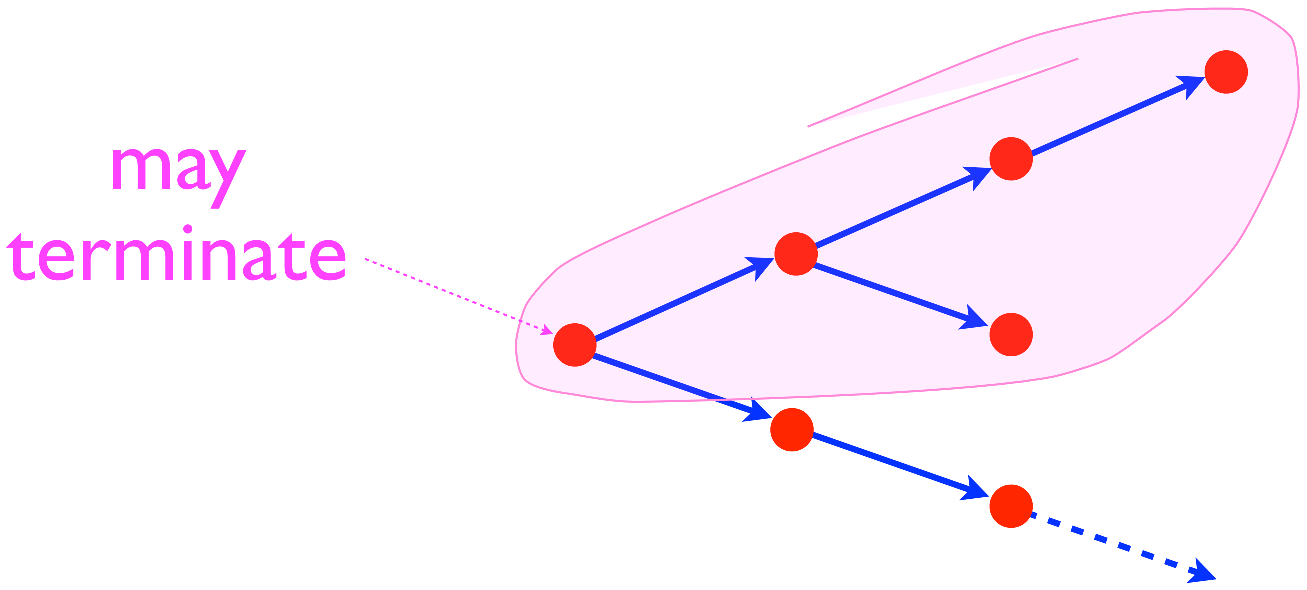
    does terminate?

  - (2) Prove these conditions to be *sufficient*

The Termination Inference Problem

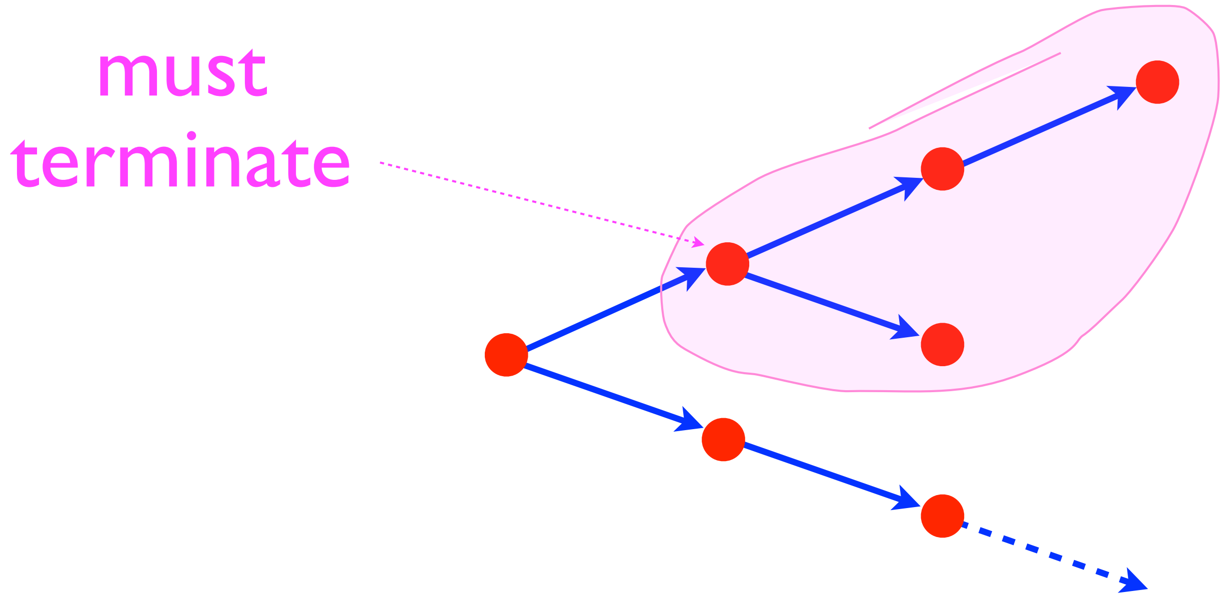
Potential termination

- 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.

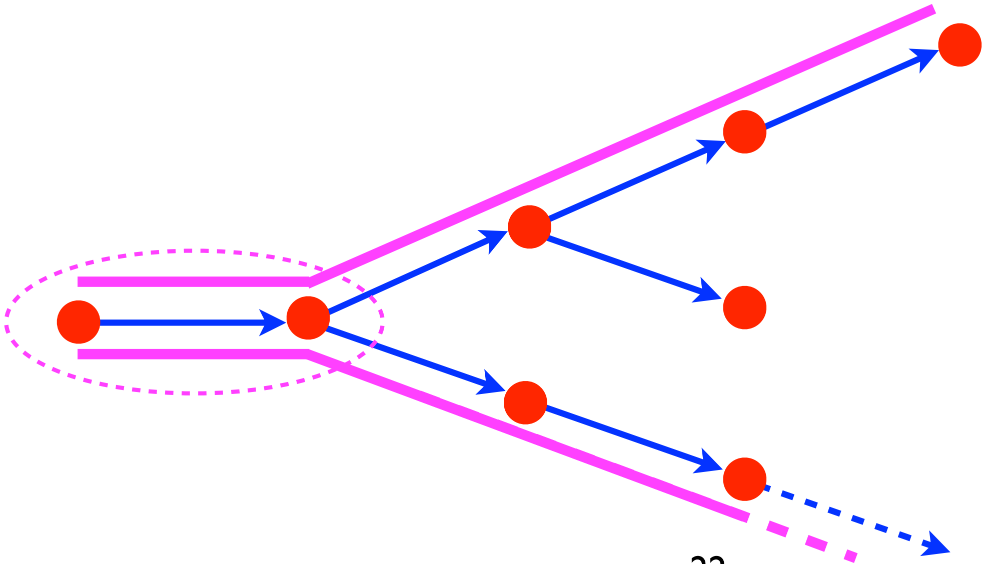
Definite termination trace abstraction

- Prefix Abstraction

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

- Definite termination abstraction

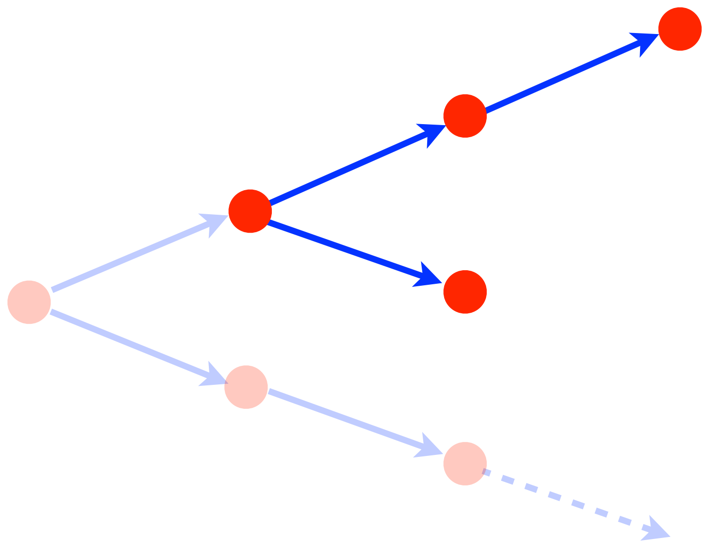
$$\alpha^{\mathsf{Mt}}(T) \quad \triangleq \quad \{\sigma \in T^{+} \mid \mathsf{pf}(\sigma) \cap \mathsf{pf}(T^{\infty}) = \emptyset\}$$



Definite termination

- The semantics/set of traces T *definitely terminates* if and only if

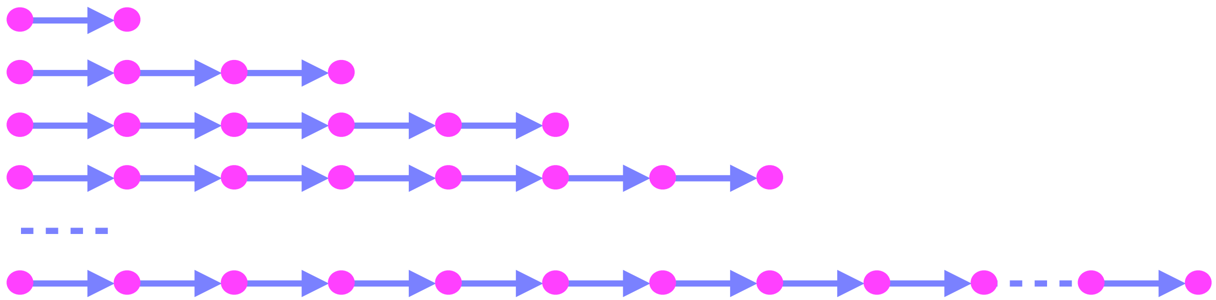
$$\alpha^{\mathrm{Mt}}(T) = T$$



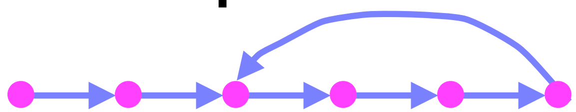
Finite abstractions do not work

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

  - Unbounded executions:



  - Finite homomorphic abstraction:



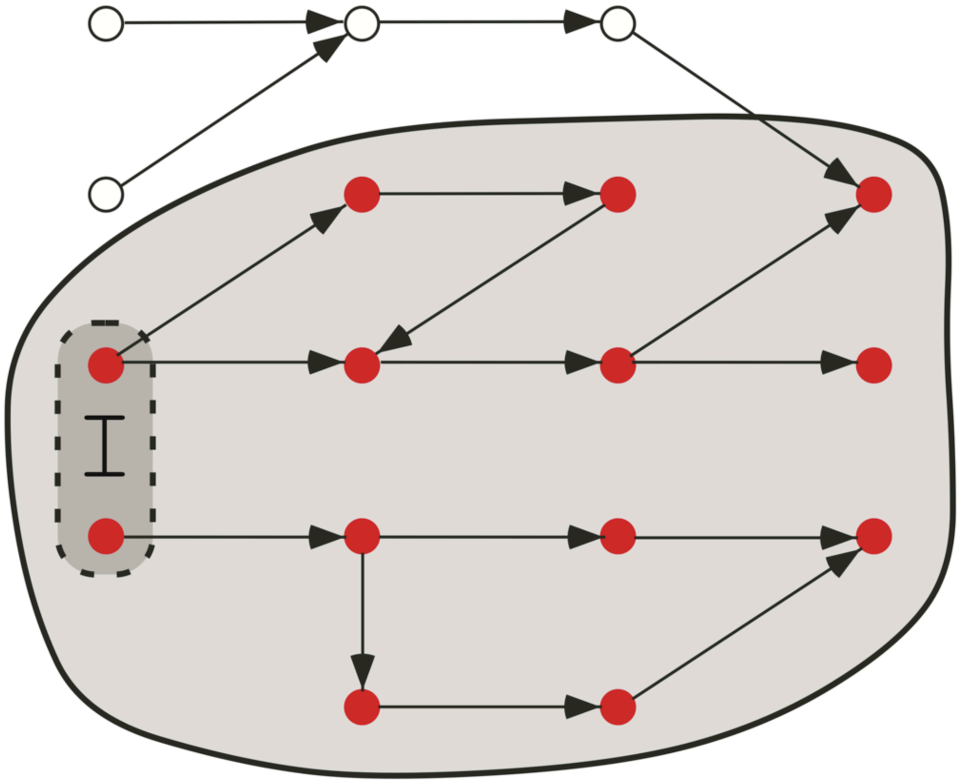
  - Termination: impossible (lasso)
  - Non-termination (lasso): unsound

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

Definite termination domain

Reachability analysis

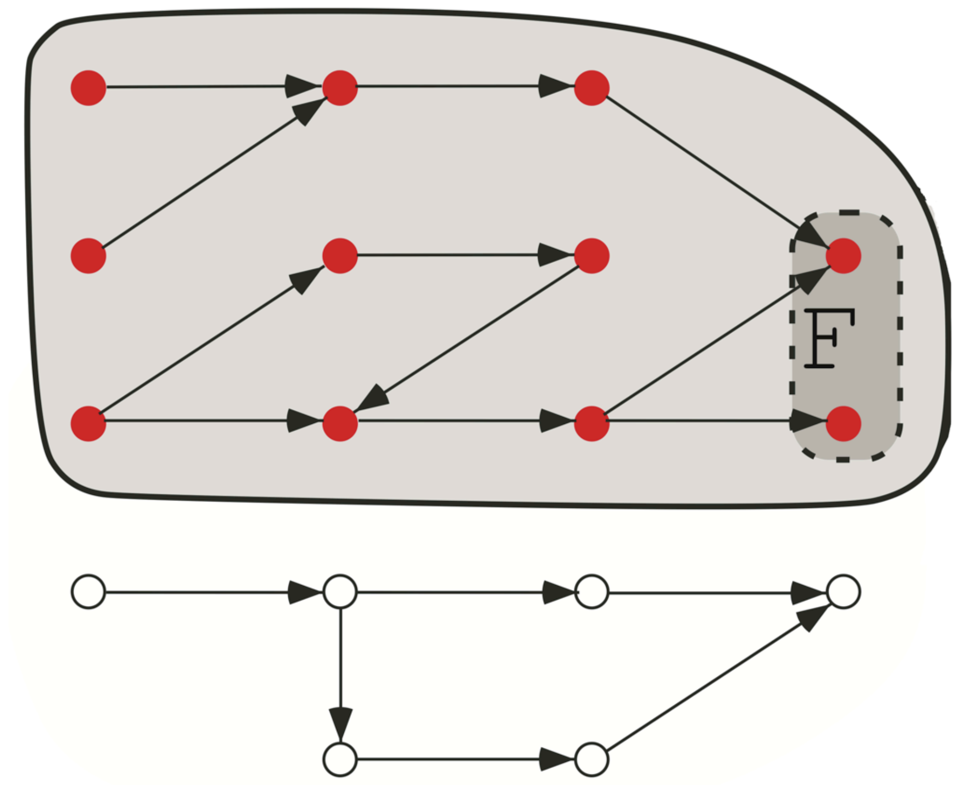
- A forward invariance analysis infers *states potentially reachable from initial states* (by over-approximating an abstract fixpoint $\operatorname{lfp} F$)(*)



(*) 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.

Accessibility analysis

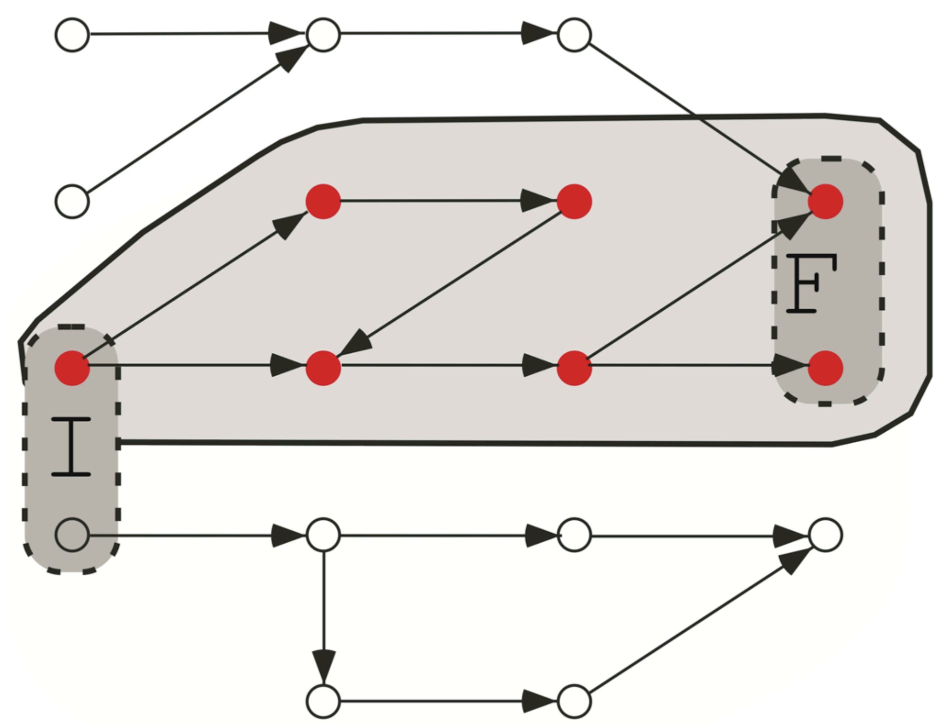
- A backward invariance analysis infers *states potentially / definitely accessing final states* (by over-approximating an abstract fixpoint $\mathrm{lfp}\ B$) [(*)]



(*) 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 $\text{lfp}\, F \sqcap \text{lfp}\, B$) (*)



$$
\begin{aligned}
X^0 &= \top \\
&\ldots \\
X^{2n+1} &= \text{lfp}\, \lambda Y \,.\, X^{2n} \sqcap F(Y) \\
X^{2n+2} &= \text{lfp}\, \lambda Y \,.\, X^{2n+1} \sqcap B(Y) \\
&\ldots
\end{aligned}
$$

(*) 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

- Arithmetic mean of two integers `x` and `y`

```
{x>=y}
  while (x <> y) {
    {x>=y+2}
       x := x - 1;
    {x>=y+1}
       y := y + 1
    {x>=y}
  }
{x=y}
```

- Necessarily $\mathtt{x} \geqslant \mathtt{y}$ for proper termination

Example (cont'd)

- Arithmetic mean of two integers $\mathtt{x}$ and $\mathtt{y}$ (cont'd)

```
while (x <> y) {
    k := k - 1;
    x := x - 1;
    y := y + 1
}
assume (k = 0)
```

auxiliary counter $\mathtt{k}$

Hint: imagine `k` is the number of remaining steps to be done in the loop

Example (cont'd)

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

```
{x=y+2k,x>=y}
  while (x <> y) {
    {x=y+2k,x>=y+2}
      k := k - 1;
    {x=y+2k+2,x>=y+2}
      x := x - 1;
    {x=y+2k+1,x>=y+1}
      y := y + 1
    {x=y+2k,x>=y}
  }
{x=y,k=0}
  assume (k = 0)
{x=y,k=0}
```

auxiliary counter `k`

- The difference `x - y` 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

$\Longrightarrow$ 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

$$\Big\downarrow \alpha^{\mathrm{Mt}}$$

Definite termination abstract backward trace semantics

$$\Big\downarrow \alpha^{\mathrm{w}}$$

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

$$\Big\downarrow \alpha^{\mathrm{rk}}$$

Variant function abstract ordinal backward semantics

The ranking abstraction

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

- $\alpha^{\mathsf{rk}}(r)$ 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:

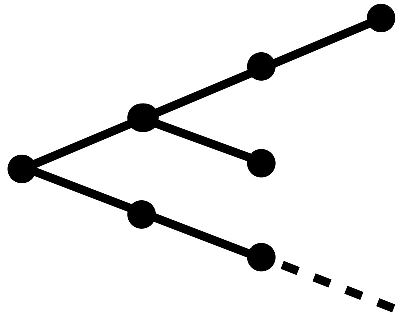
- The maximal trace semantics has a fixpoint definition
- The variant function is an abstraction of the maximal trace semantics
- With this abstraction, we construct a fixpoint definition of the abstract variant semantics

$\Longrightarrow$ *Fixpoint induction* provides a termination proof method

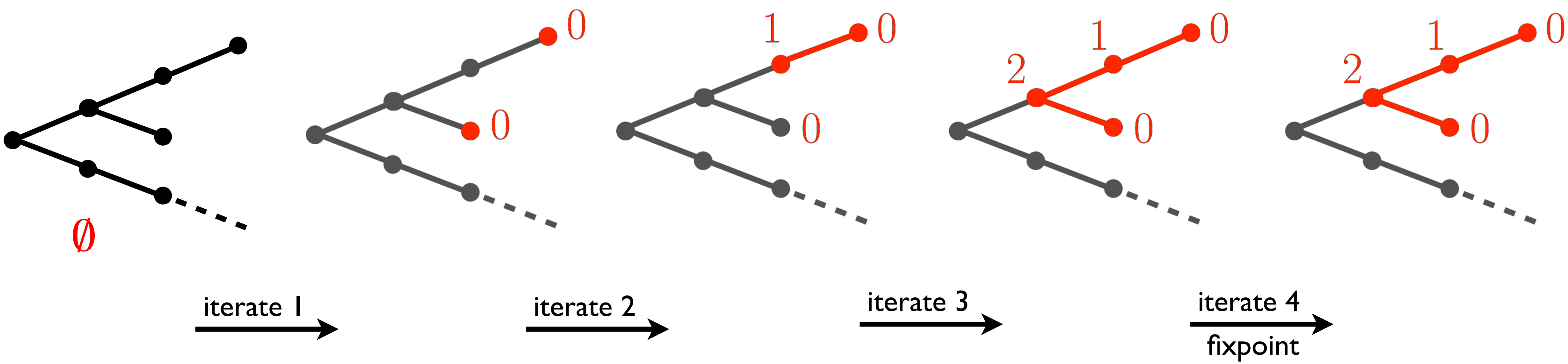
$\Longrightarrow$ Further *abstractions and widenings* provide a static analysis method

Example I

- Maximal trace semantics:



- Ranking fixpoint iterates:



Example II

- **Program**

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

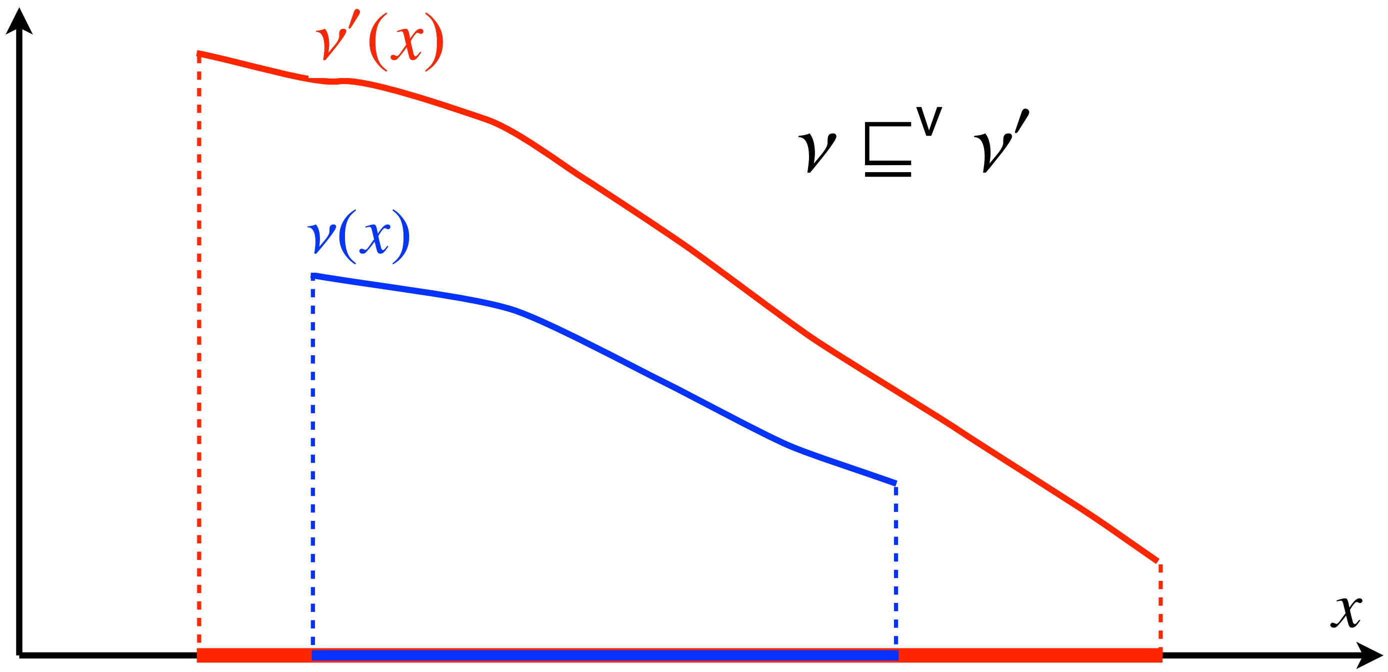
- **Fixpoint** $\nu \quad = \quad \mathsf{lfp}^{\sqsubseteq^{\mathsf{v}}}_{\dot{\emptyset}} \; \overleftarrow{\phi}^{\,\mathsf{Mv}}_{\tau}[\![\mathrm{P}]\!]$

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

- **Iterates**

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

Computational order on functions



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

© P. Cousot & R. Cousot

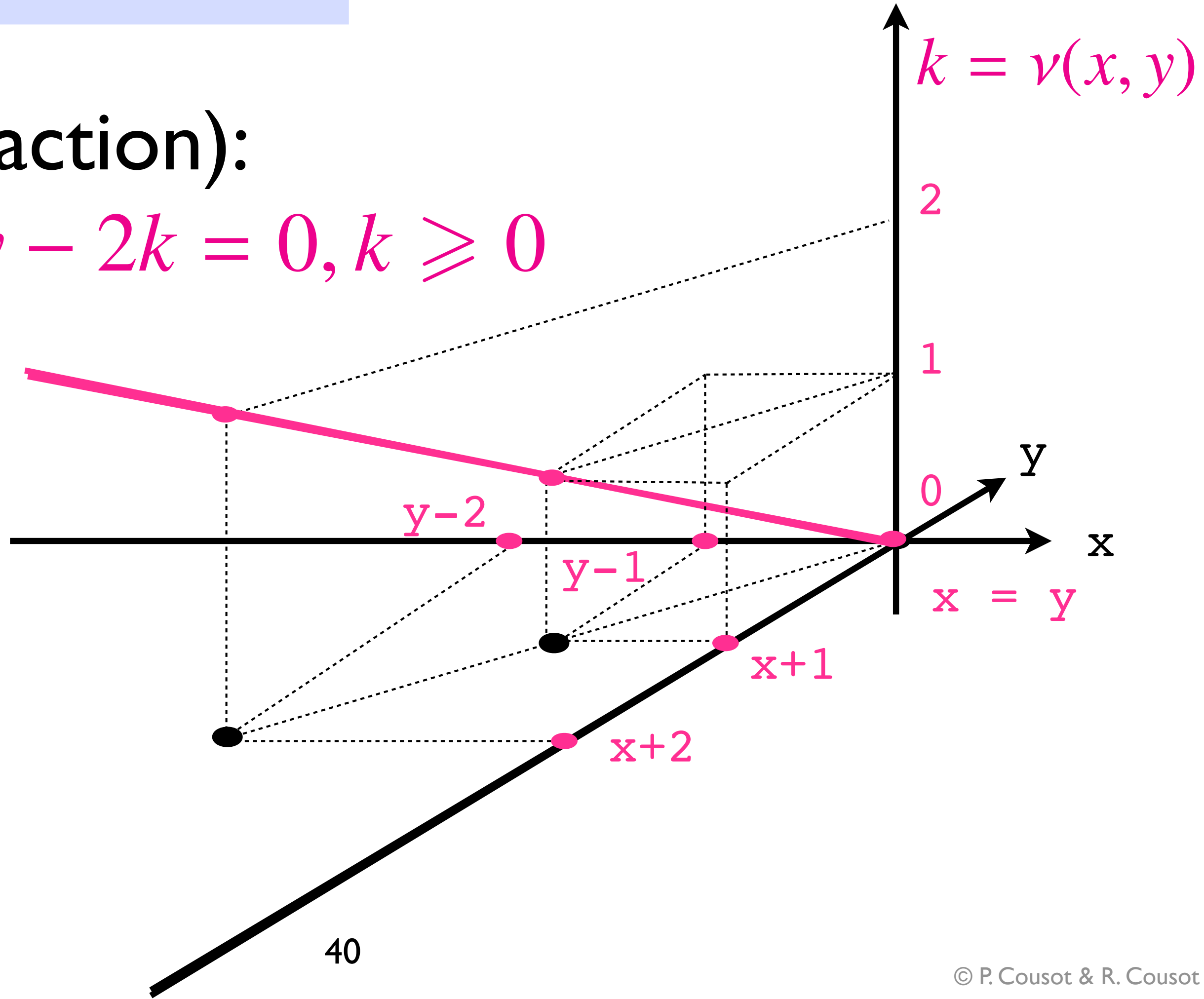
Example III

- Program:

```
{ even(x-y), x >= y }
while (x <> y) {
    x := x - 1;
    y := y + 1
}
{ x = y }
```

- Iterates (linear abstraction):

$$\exists k : v(x, y) = k, x - y - 2k = 0, k \geqslant 0$$



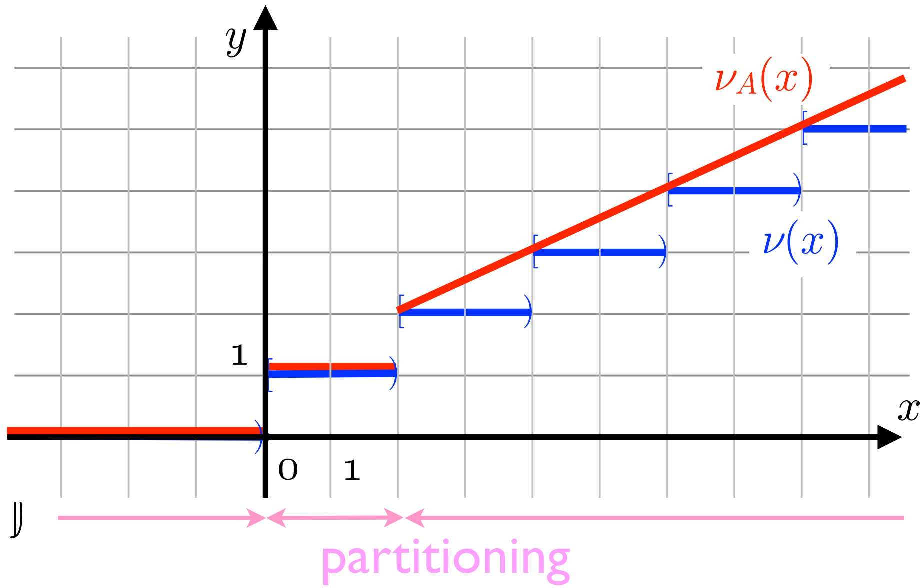
Example IV

- In general a widening is needed to enforce convergence

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

- Iterates with widening:

$$
\begin{aligned}
\nu_A^0 &= \lambda x \in [-\infty, +\infty] \bullet \bot \\
\nu_A^1 &= \lambda x \bullet \llparenthesis\, x \in [-\infty, 0] \mathrel{?} 0 \mathrel{\text{¿}} x \in [1, +\infty] \mathrel{?} \bot \,\rrparenthesis \\
\nu_A^2 &= \lambda x \in [-\infty, 0] \bullet 0 \mathbin{\dot{\cup}} \lambda x \in [1, 2] \bullet 1 \mathbin{\dot{\cup}} \lambda x \in [3, +\infty] \bullet \bot \\
\nu'^3_A &= \lambda x \bullet \llparenthesis\, x \in [-\infty, 0] \mathrel{?} 0 \mathrel{\text{¿}} x \in [1, 2] \mathrel{?} 1 \mathrel{\text{¿}} x \in [3, 4] \mathrel{?} 2 \\
&\qquad\qquad \mathrel{\text{¿}} x \in [5, +\infty] \mathrel{?} \bot \,\rrparenthesis \\
\nu_A^3 &= \nu_A^2 \mathbin{\dot{\triangledown}^{\vee}} \nu'^3_A \\
\nu'^4_A &= \lambda x \bullet \llparenthesis\, x \in [-\infty, 0] \mathrel{?} 0 \mathrel{\text{¿}} x \in [1, 2] \mathrel{?} 1 \mathrel{\text{¿}} x \in [3, +\infty] \mathrel{?} \frac{x}{2} + 1 \,\rrparenthesis \\
\nu_A^4 &= \nu_A^3 \;.
\end{aligned}
$$



Objection 1: Turing/Floyd's method goes forward not backward!

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

before

after

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

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

Backward variant ν:

$$\nu(x_{\text{before}}) = \nu(x_{\text{after}}) + 1$$

$$\Longleftrightarrow \nu(x_{\text{before}}) = \nu(f(x_{\text{before}})) + 1$$

Forward variant ν:

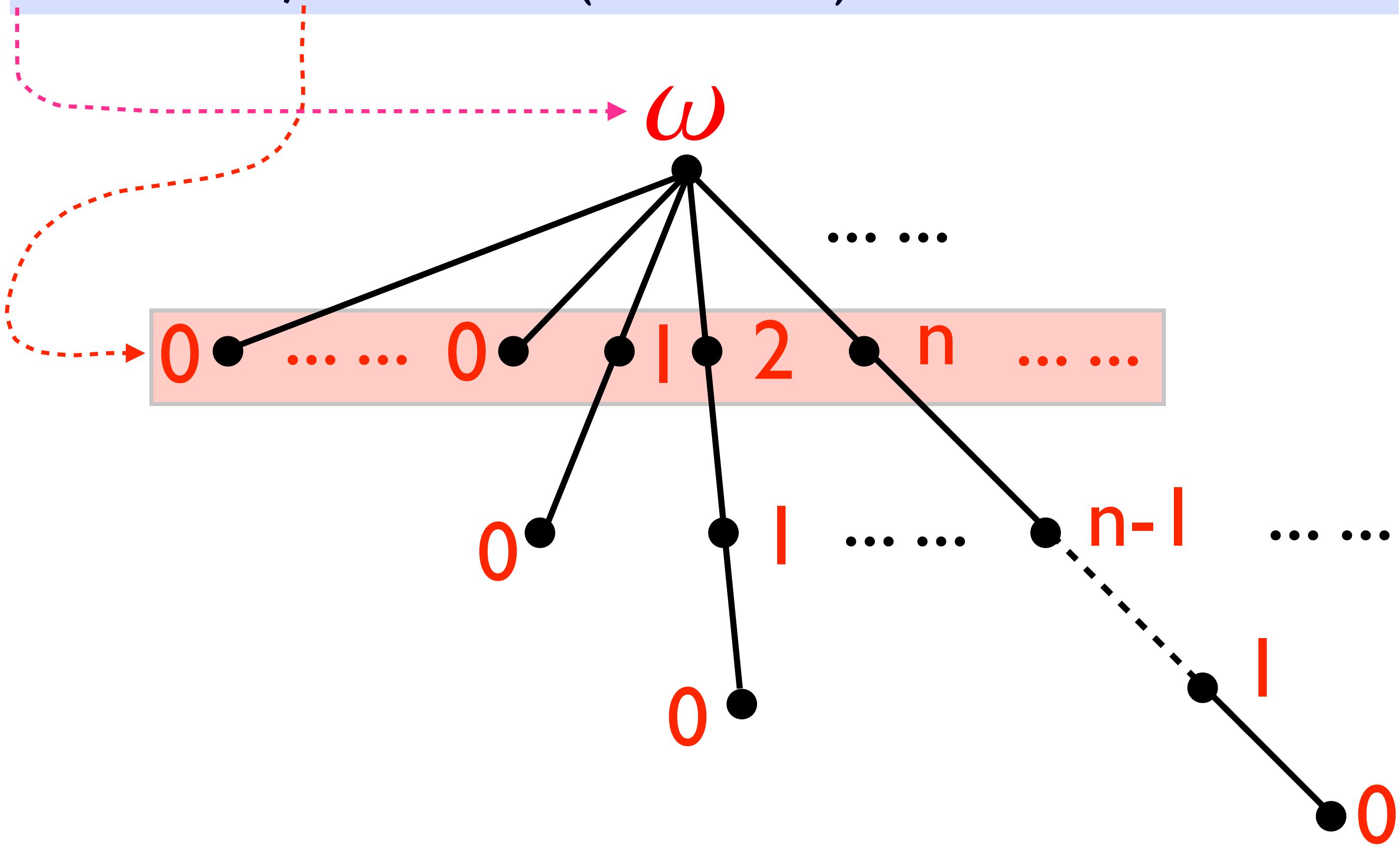
$$\nu(x_0) = \nu(x) + 1$$

$$\Longleftrightarrow \nu(x_0) = \nu(f(x_0)) + 1$$

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

Objection II: you need ordinals! (*)

- Example: `x := ?; while (x >= 0) do x := x - 1 od`
- Ranking:



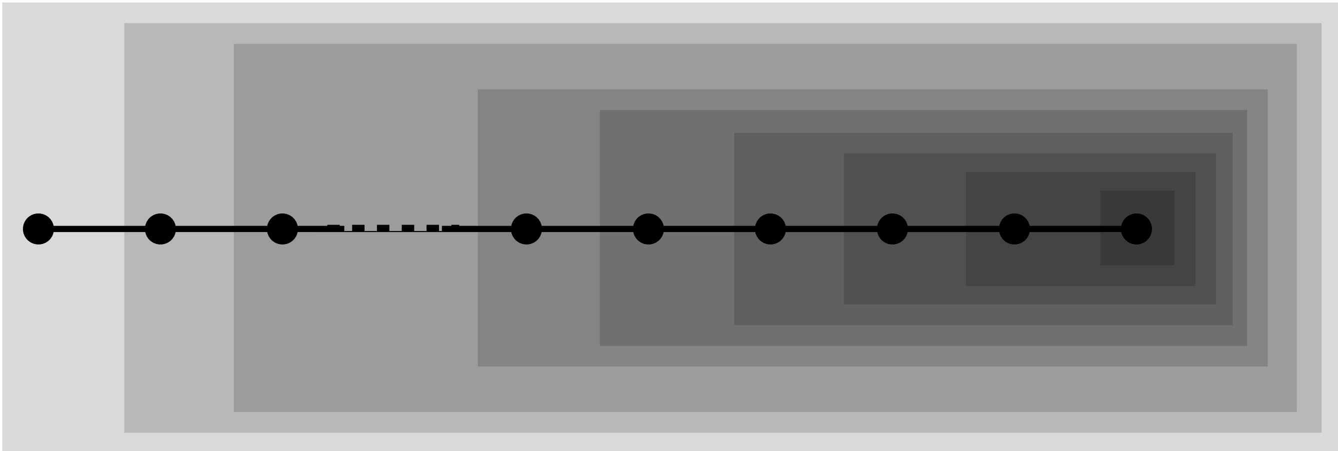
- To avoid transfinite ordinals/well-founded orders (*) for unbounded non-determinism, the computations need to be structured!

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

Structuring trace semantics with segments

Floyd/Turing termination proof method

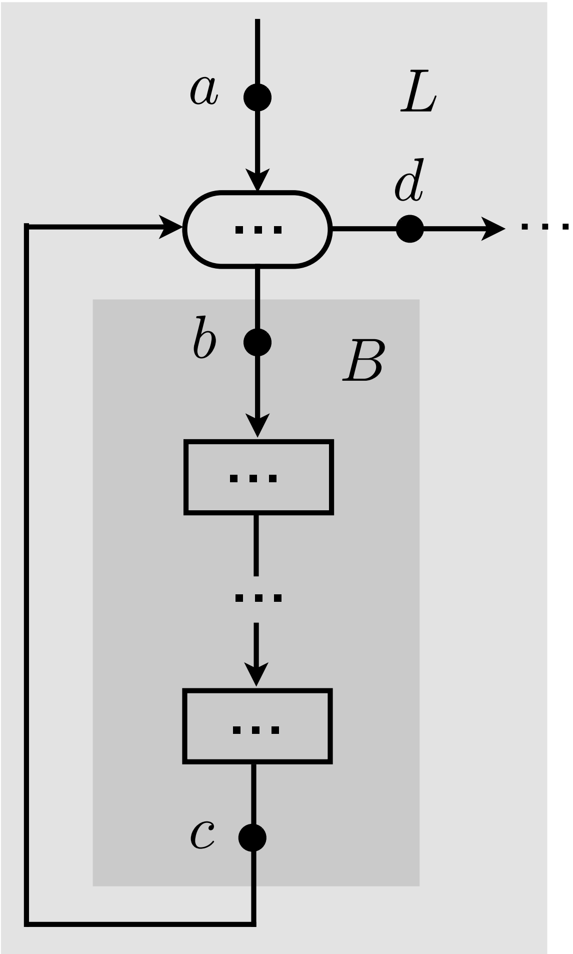
- Trivial postfix structuring of traces into segments



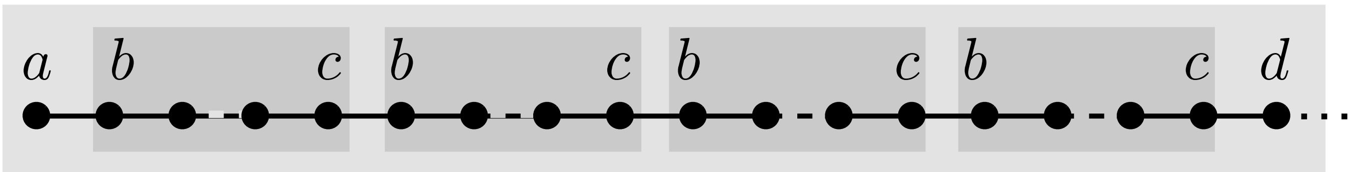
- Also used for termination of straight-line code (no need for variant functions)

Floyd with nested loops

- The trace semantics is recursively structured in segments according to loop nesting



Prove termination of outer loop assuming termination of body/ nested inner loops

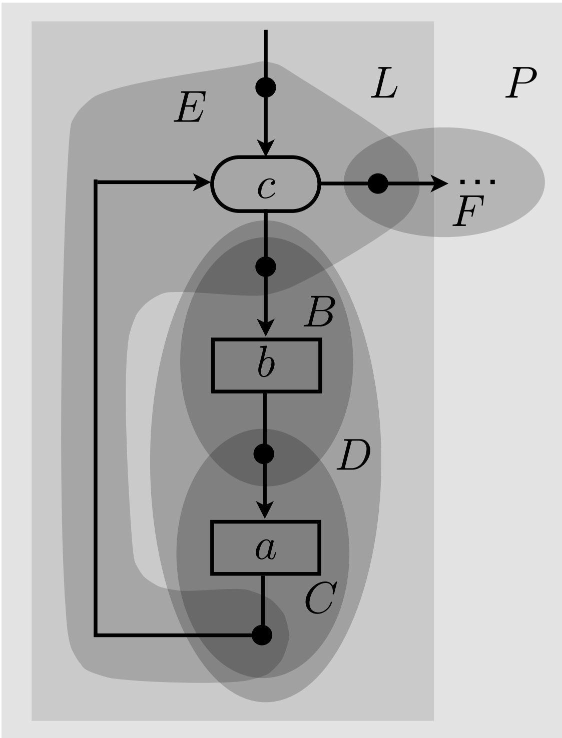


(equivalent to lexicographic orderings)

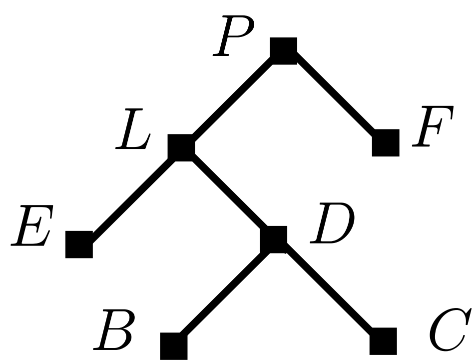
Hoare logic

- The trace semantics is recursively structured in segments according to the program syntax

- `while (c) { b; a }...`



tree structure
of the segmentation:



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

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

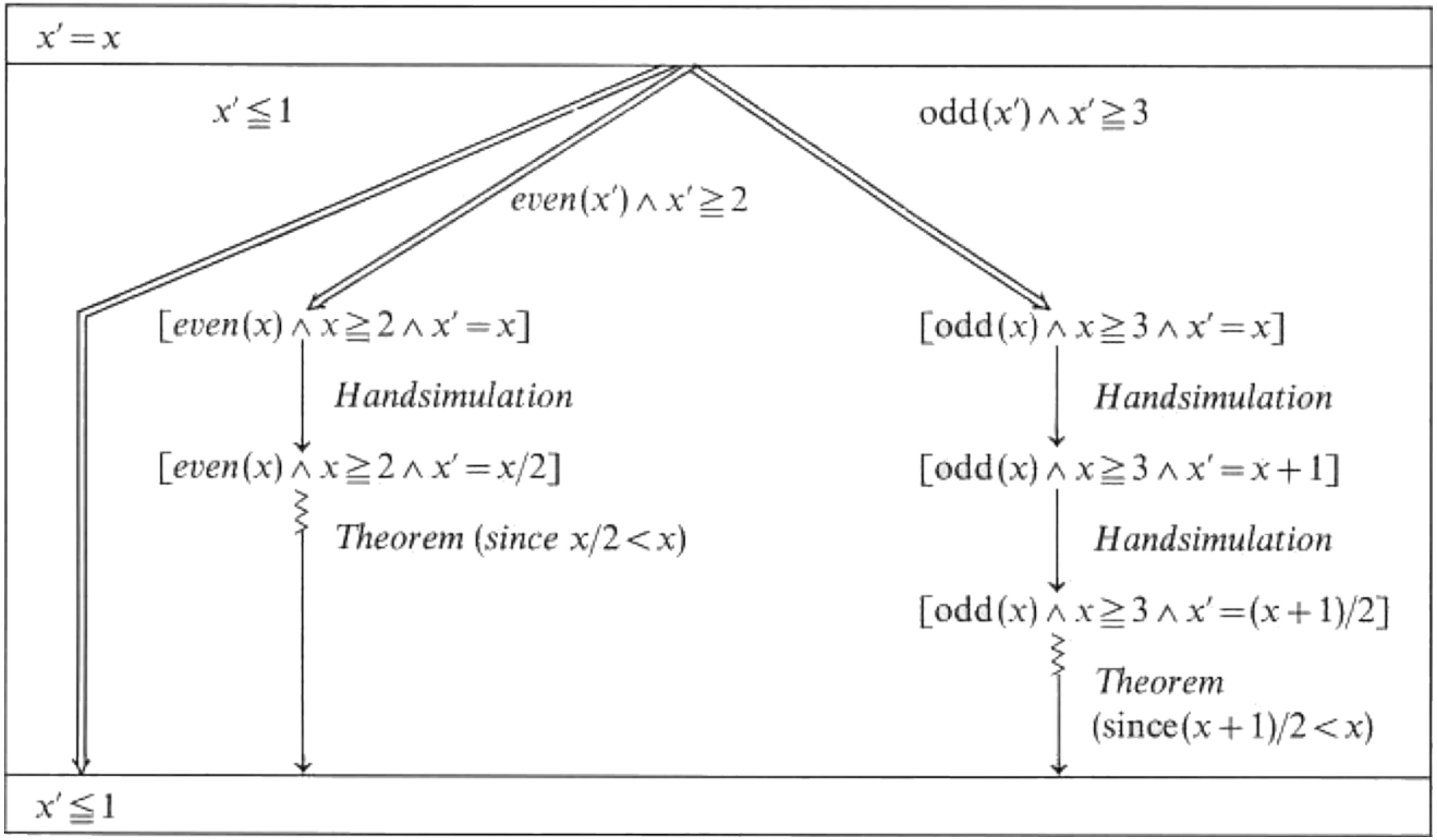
Z. Manna and A. Pnueli. Axiomatic approach to total correctness of programs. *Acta Inf.*, 3:243–263, 1974.

Burstall's proof method by hand-simulation and a little induction

- Program

do odd(x) **and** $x \geq 3 \rightarrow x := x+1$
$\square$ even (x) **and** $x \geq 2 \rightarrow x := x/2$
od

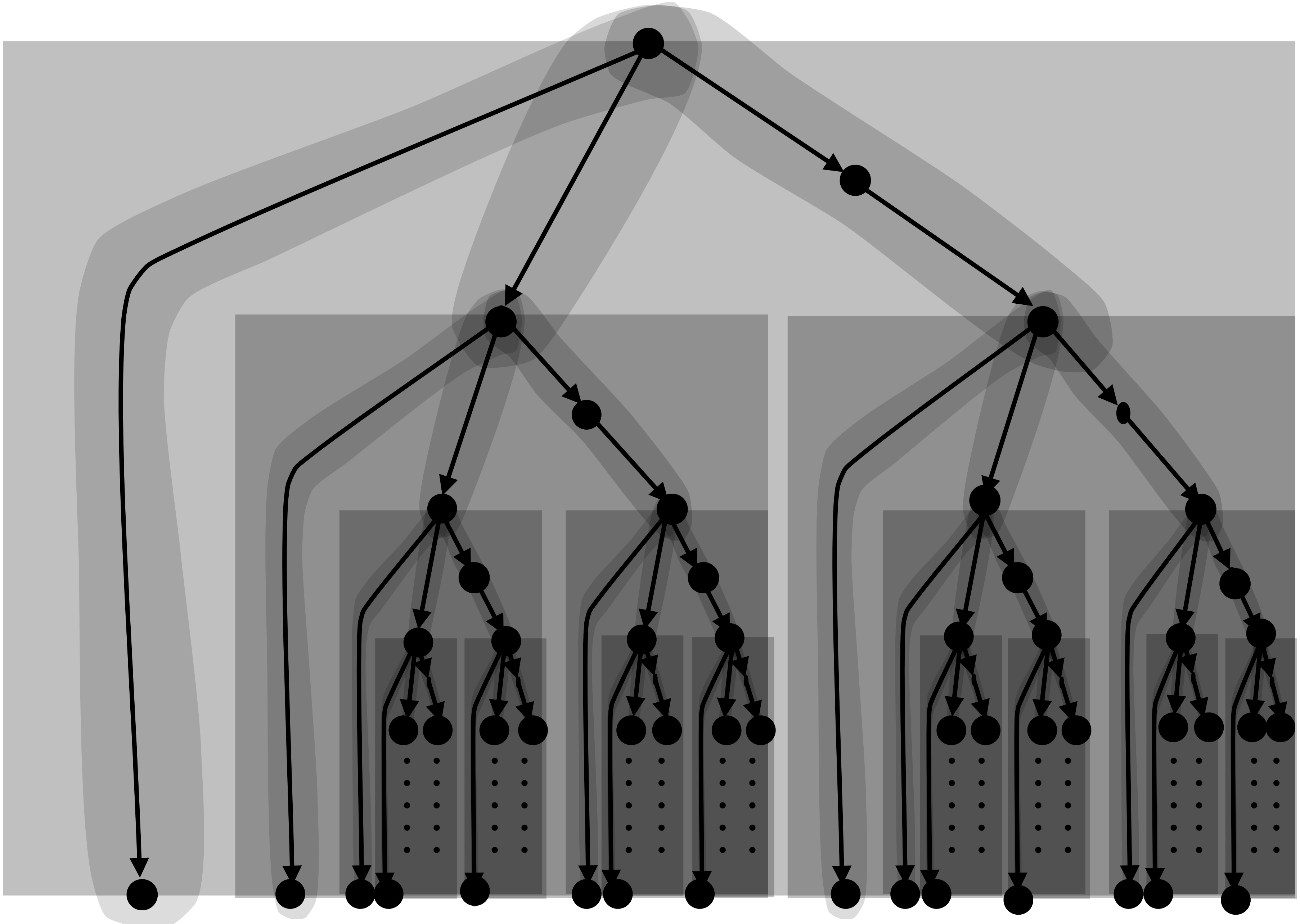
- Proof chart



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

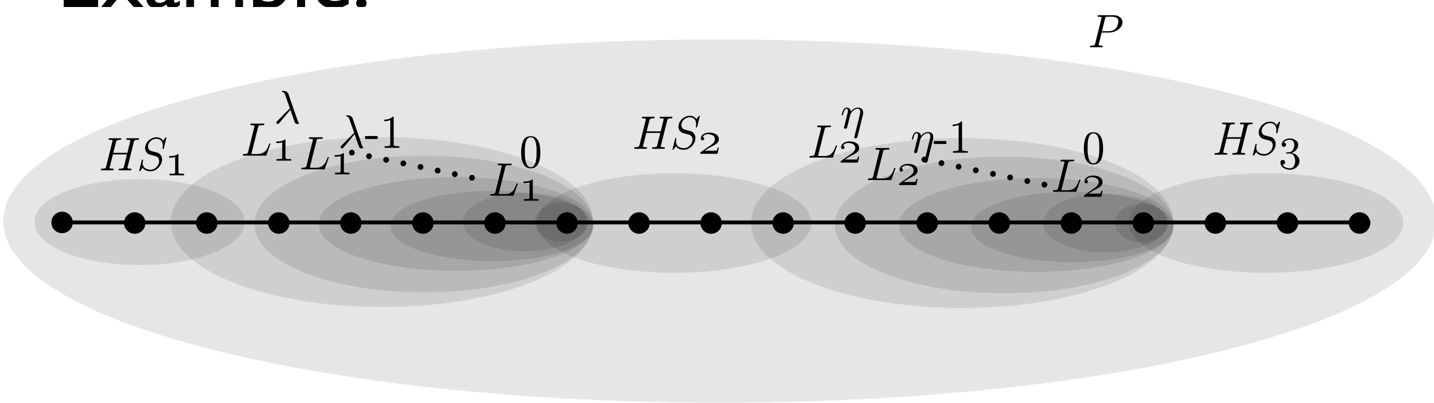
P. Cousot and R. Cousot. Sometime = always + recursion ≡ always, on the equivalence of the intermittent and invariant assertions methods for proving inevitability properties of programs. *Acta Informatica*, 24:1–31, 1987.

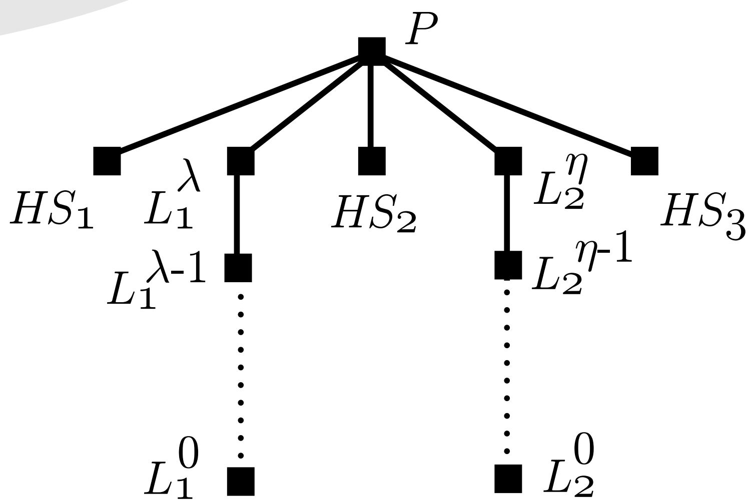
Well-founded tree structure of the trace segmentation



Burstall's proof method by hand-simulation and a little induction

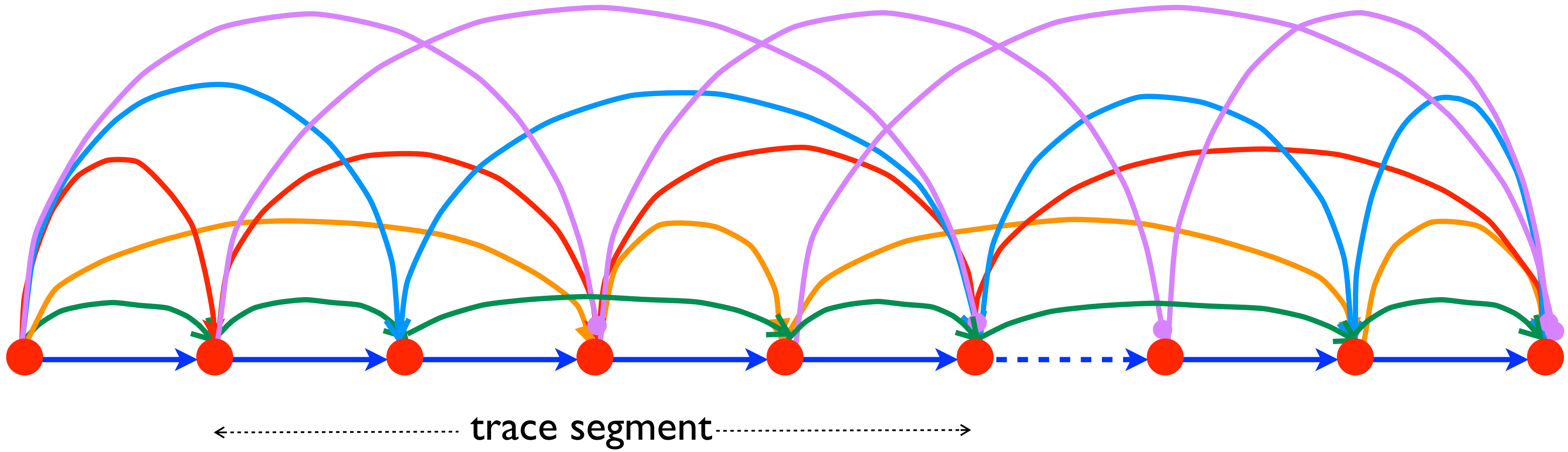
- Iterative program but recursive proof structur
- Inductive trace cover by segments
- Example:





Podelski-Rybalchenko

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

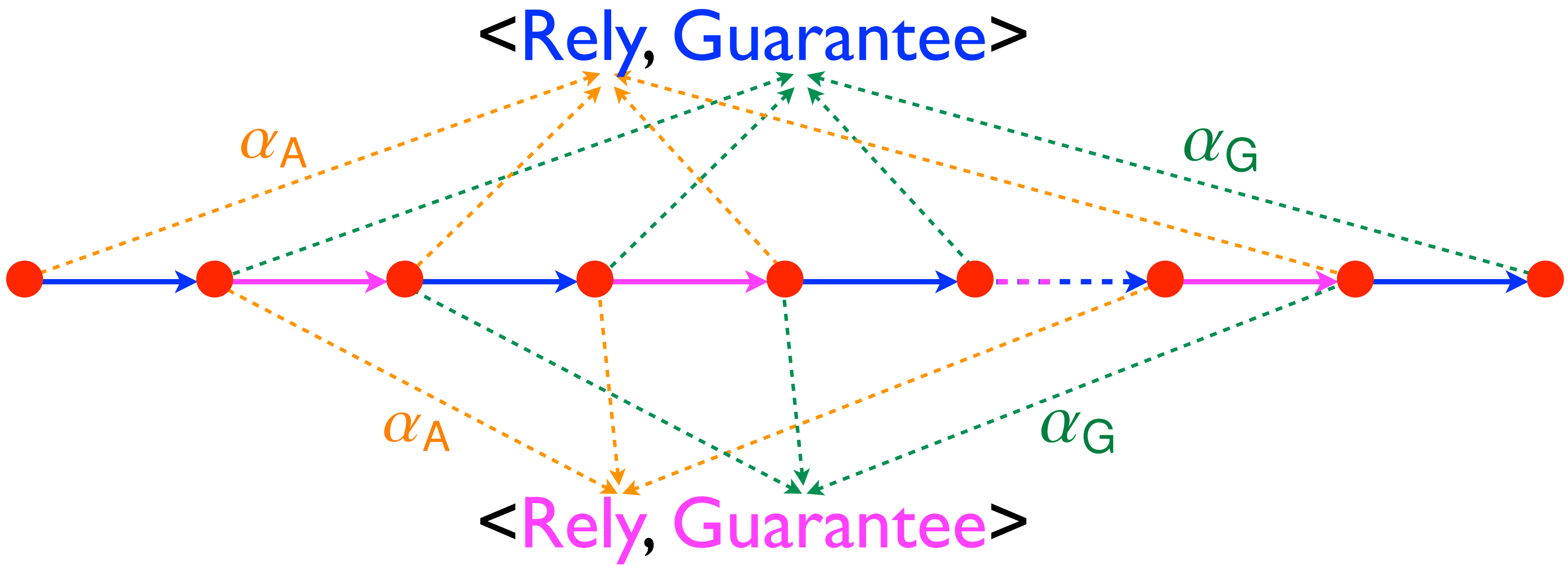


- Termination based on Ramsey theorem on colored edges of a complete graph, no recursive structure

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 rely-guarantee/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)

Trace semantics segmentation

- Recursive trace segmentation

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

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

- Proof by induction on the possibly infinite but well-founded trace segmentation tree
- *Orthogonal* to proofs on segment sets (using variant functions, Ramsey 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