Symbolic Shape Analysis

Thomas Wies

University of Freiburg, Germany

```
class SortedList {
  private static Node first;
  /*: public static specvar content :: objset;
      vardefs "content == {v. v ≠ null ∧ next* first v}";
      invariant "tree [next]";
      invariant "∀ v. v ∈ content ∧ v.next ≠ null
                              ⟶ v..Node.data ≤ v.next.data"; */
  public static void insert(Node n)
    /*: requires "n ≠ null ∧ n ∉ content"
        modifies content
        ensures "content = old content ∪ {n}" */
  {
    Node prev = null;
    Node curr = first;
    while ((curr != null) && (curr.data < n.data)) {
      prev = curr;
      curr = curr.next;
    }
    n.next = curr;
    if (prev != null) prev.next = n;
    else first = n;
  }
}
```

Bohne, Symbolic Shape Analysis Implementation

Properties verified in previous example:

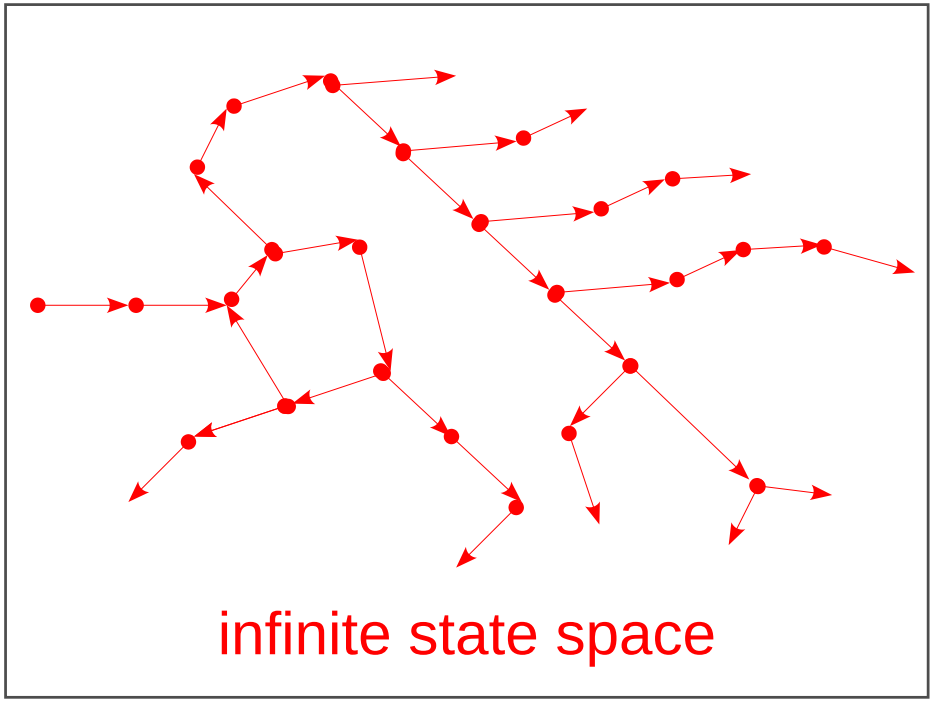
- correctly inserts the element into the list
  (relates pre- and post states of procedure)
- list remains sorted
- data structure remains acyclic list
- no null pointer dereferences

Bohne

- accepts annotated Java programs as input
- annotations are user-specied formulae:
  - data structure invariants
  - procedure contracts (pre- and post conditions)
- automatically computes quantified loop invariants
- proves desired properties and absence of errors

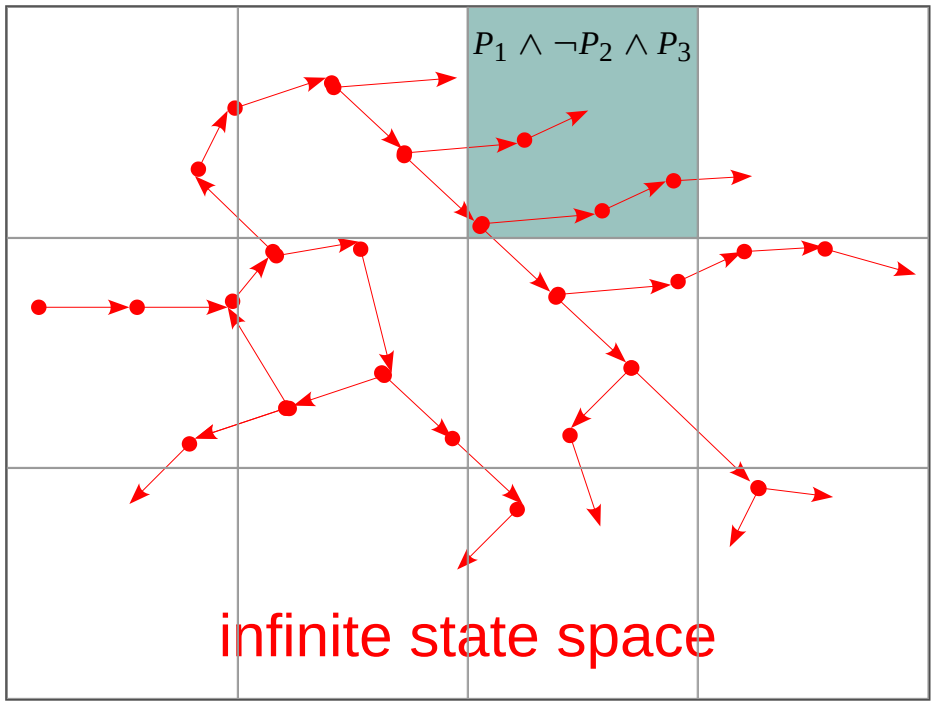
Predicate Abstraction

- take transition graph (nodes are states)
- define partitioning of nodes through state predicates
- abstract transition graph is graph of abstract nodes
- abstract nodes are equivalence classes of concrete nodes



Predicate Abstraction

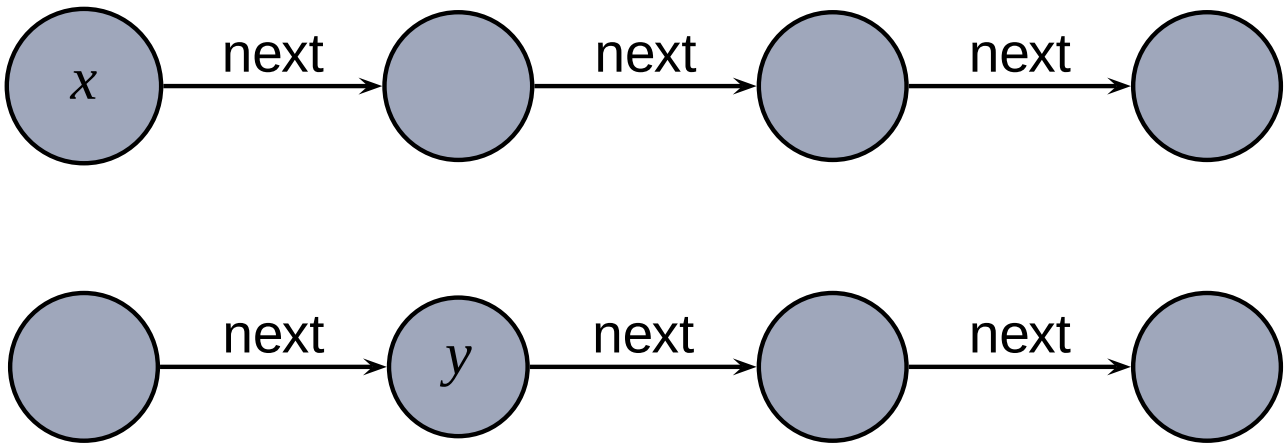
- take transition graph (nodes are states)
- define partitioning of nodes through state predicates
- abstract transition graph is graph of abstract nodes
- abstract nodes are equivalence classes of concrete nodes



state predicates: P_1, P_2, P_3

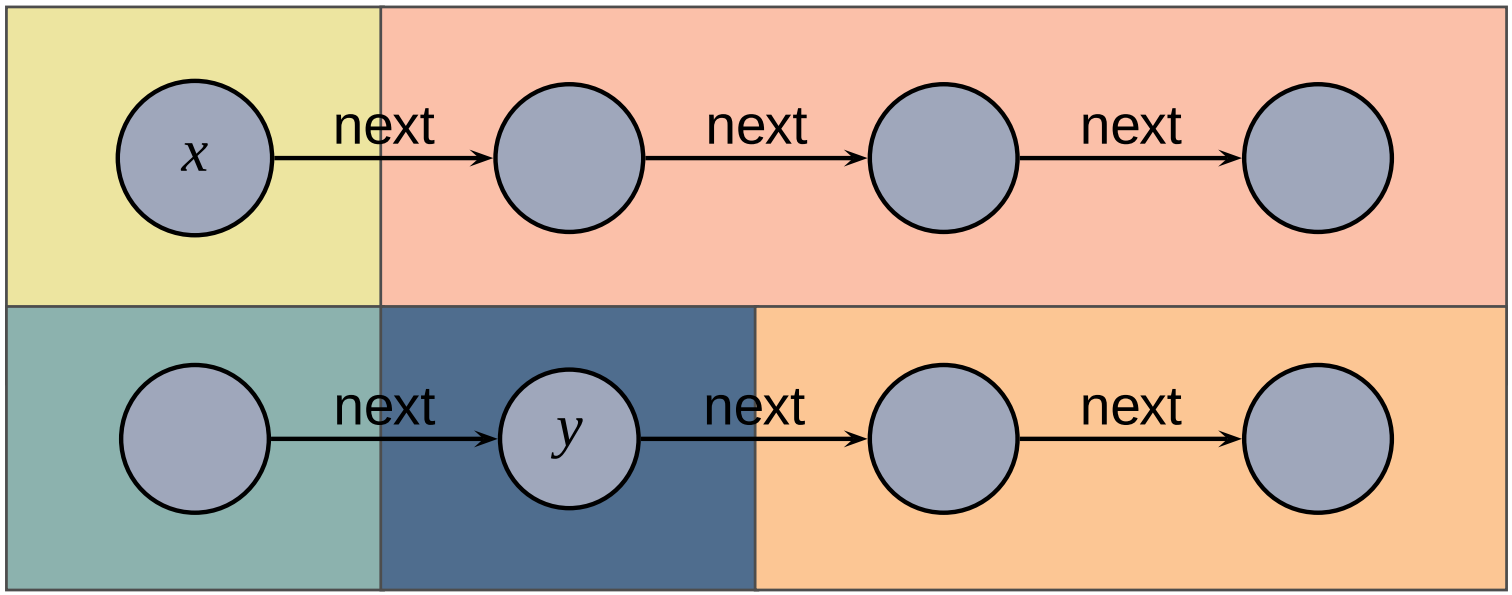
Shape Analysis à la Sagiv, Reps, and Wilhelm

- states are graphs
- define partitioning of nodes through predicates on nodes
- abstract states are graphs of abstract nodes
- abstract nodes are equivalence classes of concrete nodes



Shape Analysis à la Sagiv, Reps, and Wilhelm

- states are graphs
- define partitioning of nodes through predicates on nodes
- abstract states are graphs of abstract nodes
- abstract nodes are equivalence classes of concrete nodes



$$\text{shape analysis} = 2^{\text{predicate abstraction}}$$

Why go symbolic?

Apply not only idea, but also techniques of predicate abstraction.

Generic Benefits of Predicate Abstraction

- use formulae to represent infinite sets of states
  - no need to define meaning of abstract values
  - abstract domain $\subseteq$ concrete domain
  - abstraction $=$ entailment $\models$

Generic Benefits of Predicate Abstraction

- use formulae to represent infinite sets of states
  - no need to define meaning of abstract values
  - abstract domain $\subseteq$ concrete domain
  - abstraction $=$ entailment $\models$
- use reasoning procedures
  - automation
  - separation of concerns (black-boxing)
  - soundness by construction, loss of precision identifiable
  - get leverage from theorem proving community
  - abstraction $=$ provable entailments $\vdash$

Generic Benefits of Predicate Abstraction

- use formulae to represent infinite sets of states
  - no need to define meaning of abstract values
  - abstract domain $\subseteq$ concrete domain
  - abstraction $=$ entailment $\models$
- use reasoning procedures
  - automation
  - separation of concerns (black-boxing)
  - soundness by construction, loss of precision identifiable
  - get leverage from theorem proving community
  - abstraction $=$ provable entailments $\vdash$
- abstraction refinement
  - more automation
  - symbolic execution of counterexamples
  - abstract domain $\subset$ refined abstract domain

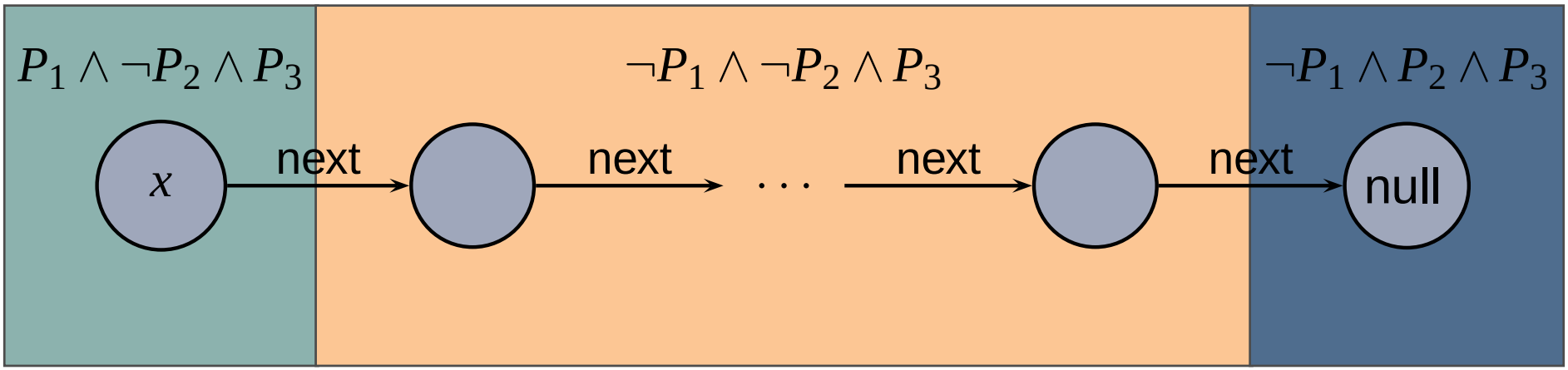
Outline

1. Boolean heaps (abstract domain)
2. Cartesian post (abstract transformer)
3. Abstraction refinement

Boolean Heaps

Partition heap according to finitely many predicates on heap objects.

$$P_1 = \{ v \mid v = x \} \quad P_2 = \{ v \mid v = \mathsf{null} \} \quad P_3 = \{ v \mid \mathsf{next}^*(x, v) \}$$



Describe partitioning as a universally quantified formula

$$\forall v \,.\, P_1 \wedge \neg P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge \neg P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge P_2 \wedge P_3$$

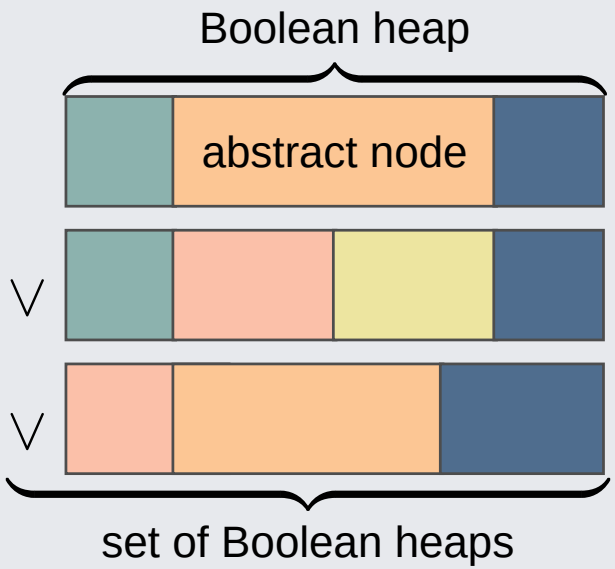
➜ Boolean heaps

Abstract domain = {sets of Boolean heaps}

Abstract element

$$\bigvee_i \forall v \,.\, \underbrace{\bigvee_j \underbrace{\bigwedge_k P_{i,j,k}(v)}_{\text{abstract node}}}_{\text{Boolean heap}}$$

set of Boolean heaps



Symbolic shape analysis

$$\underbrace{\underbrace{\bigvee_i \; \forall v \,.\, \bigvee_j \underbrace{\bigwedge_k P_{i,j,k}(v)}_{\text{abstract node}}}_{\text{Boolean heap} \cong \text{abstract state}}}_{\text{set of Boolean heaps}}$$

➜ sets of sets of bit-vectors (sets of BDDs)

Predicate abstraction

$$\underbrace{\underbrace{\bigvee_i \; \underbrace{\bigwedge_j P_{i,j}}_{\text{abstract state}}}_{\text{sets of abstract states}}}$$

➜ sets of bit-vectors (BDDs)

➜ Boolean heaps provide extra precision needed for shape analysis.

Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\text{post}^{\#}(H) = ?$$

Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\text{post}^{\#}(H) = \alpha \circ \text{post} \circ \gamma(H)$$

$\text{post}^{\#}$ is most precise abstract post, but it is also hard to compute.

Abstract Post on Boolean Heaps

How to compute abstract post on Boolean heaps?

$$\text{post}^{\#}(H) = \alpha \circ \text{post} \circ \gamma(H)$$

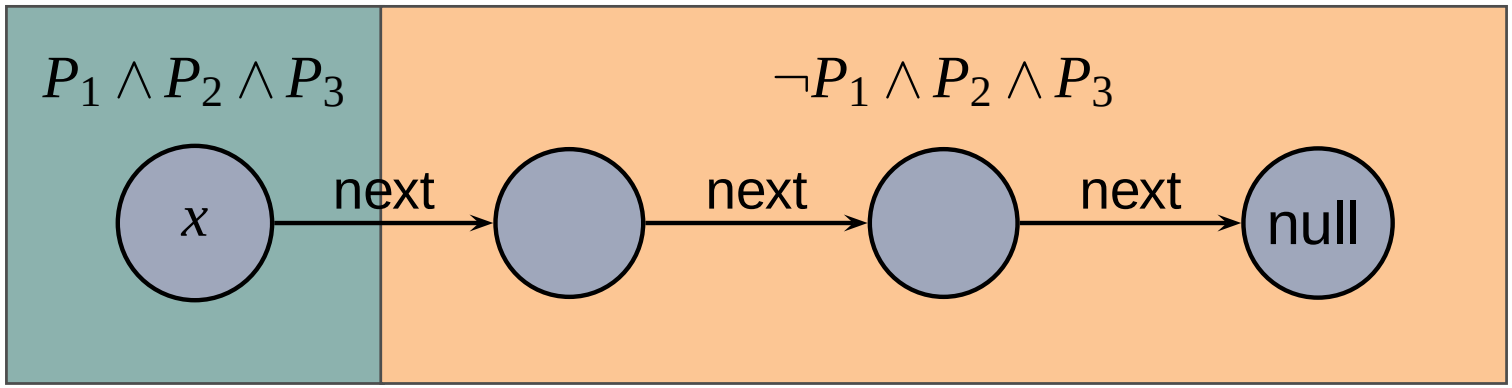
$\text{post}^{\#}$ is most precise abstract post, but it is also hard to compute.

Bohne implements an abstraction of $\text{post}^{\#}$ that can be computed efficiently.

Next slides: Cartesian post.

Abstract Post on Boolean Heaps

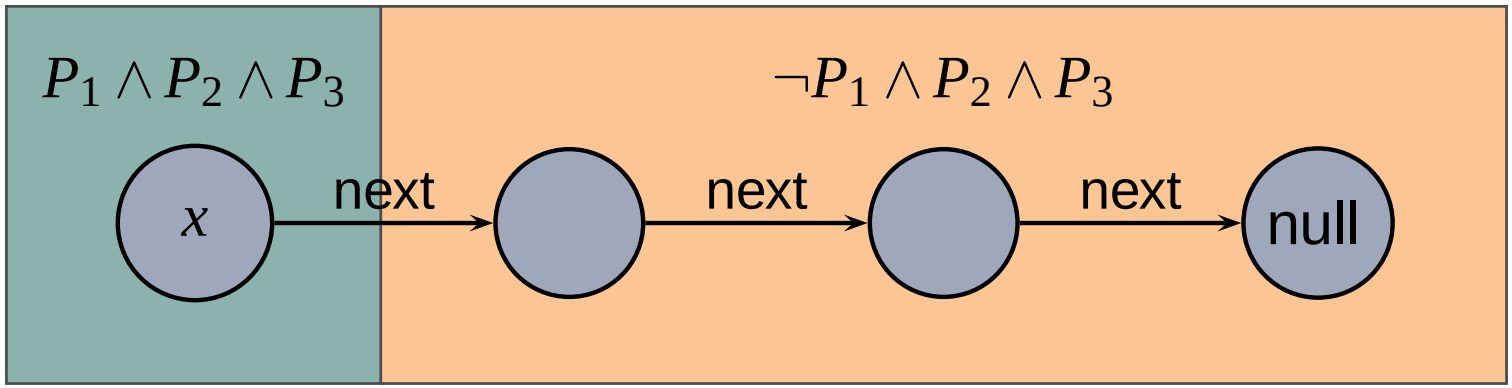
$$P_1 = \{ v \mid v = x \} \quad P_2 = \{ v \mid \mathsf{next}^*(v, \mathsf{null}) \} \quad P_3 = \{ v \mid \mathsf{next}^*(x, v) \}$$



$$\forall v \,.\, P_1 \wedge P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge P_2 \wedge P_3$$

Abstract Post on Boolean Heaps

$$P_1 = \{\, v \mid v = x \,\} \quad P_2 = \{\, v \mid \mathsf{next}^*(v, \mathsf{null}) \,\} \quad P_3 = \{\, v \mid \mathsf{next}^*(x, v) \,\}$$

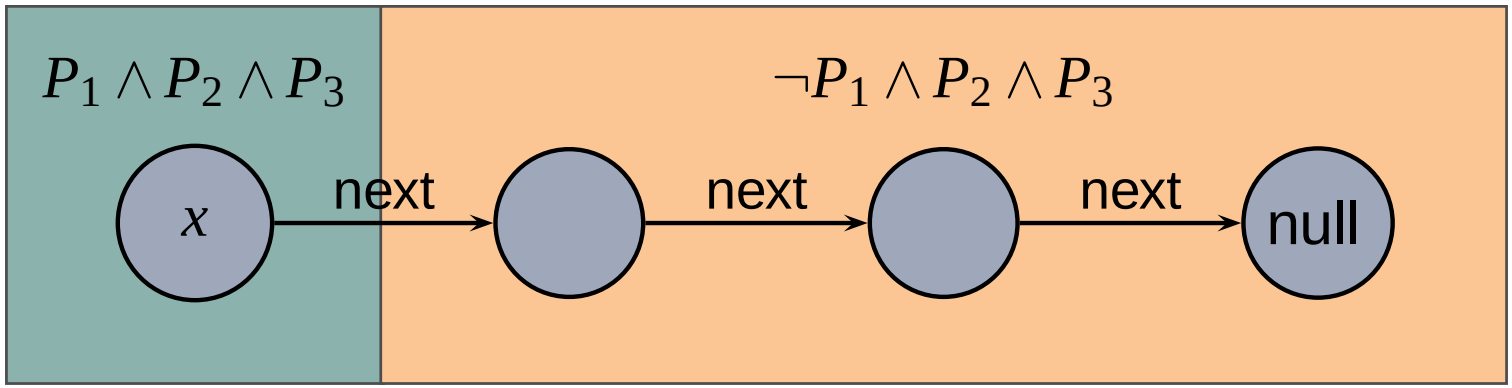


$$\alpha \circ \mathsf{post}_c \circ \gamma(\forall v \,.\, P_1 \wedge P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge P_2 \wedge P_3)$$

for command $c = (\texttt{x:=x.next})$

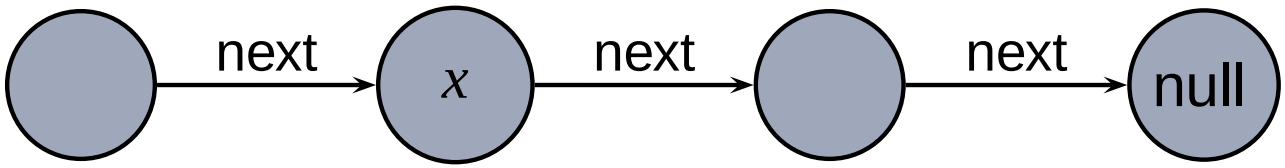
Abstract Post on Boolean Heaps

$$P_1 = \{\, v \mid v = x \,\} \quad P_2 = \{\, v \mid \mathsf{next}^*(v, \mathsf{null}) \,\} \quad P_3 = \{\, v \mid \mathsf{next}^*(x, v) \,\}$$



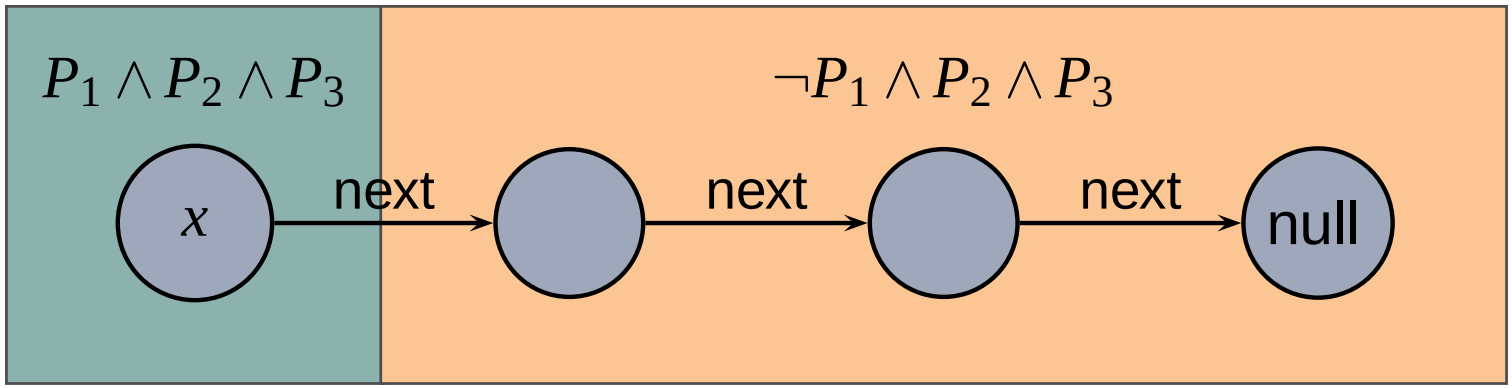
$$\alpha \circ \mathsf{post}_c \circ \gamma(\forall v \,.\, P_1 \wedge P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge P_2 \wedge P_3)$$

for command $c = (\texttt{x:=x.next})$



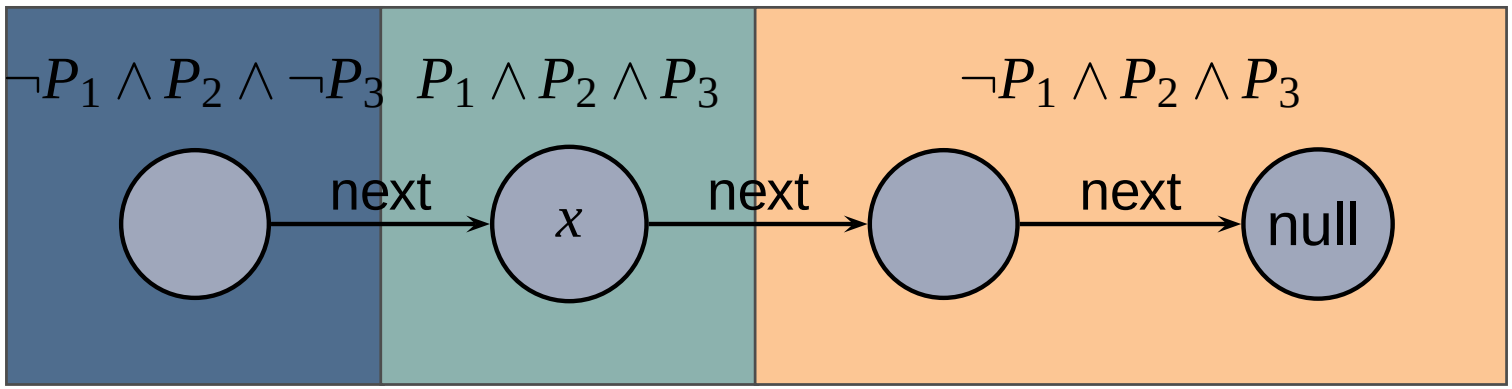
Abstract Post on Boolean Heaps

$P_1 = \{ v \mid v = x \}$ $P_2 = \{ v \mid \mathsf{next}^*(v, \mathsf{null}) \}$ $P_3 = \{ v \mid \mathsf{next}^*(x, v) \}$



$\alpha \circ \mathsf{post}_c \circ \gamma(\forall v \,.\, P_1 \wedge P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge P_2 \wedge P_3)$

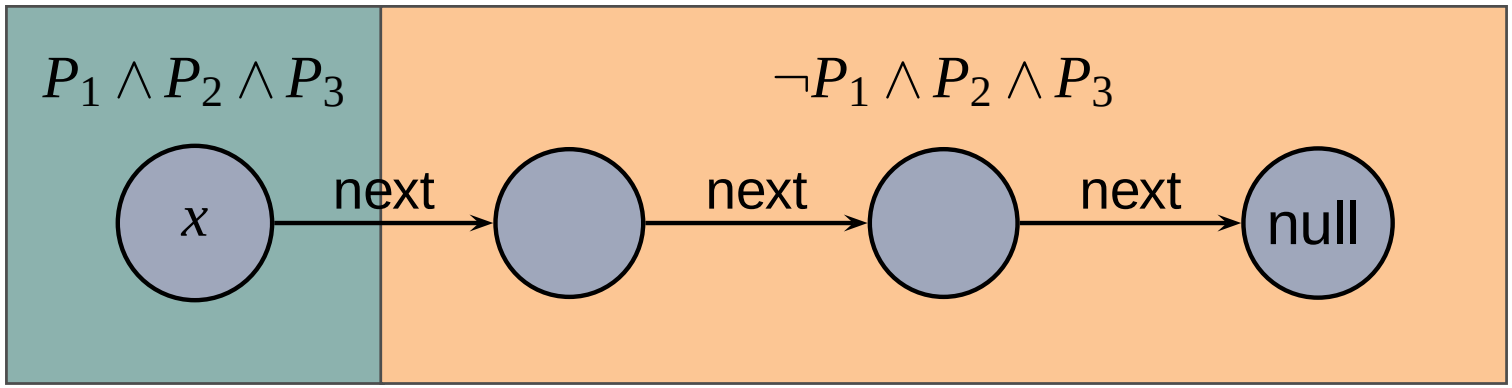
for command $c = (\texttt{x:=x.next})$



$\forall v \,.\, \neg P_1 \wedge P_2 \wedge \neg P_3 \;\vee\; P_1 \wedge P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge P_2 \wedge P_3$

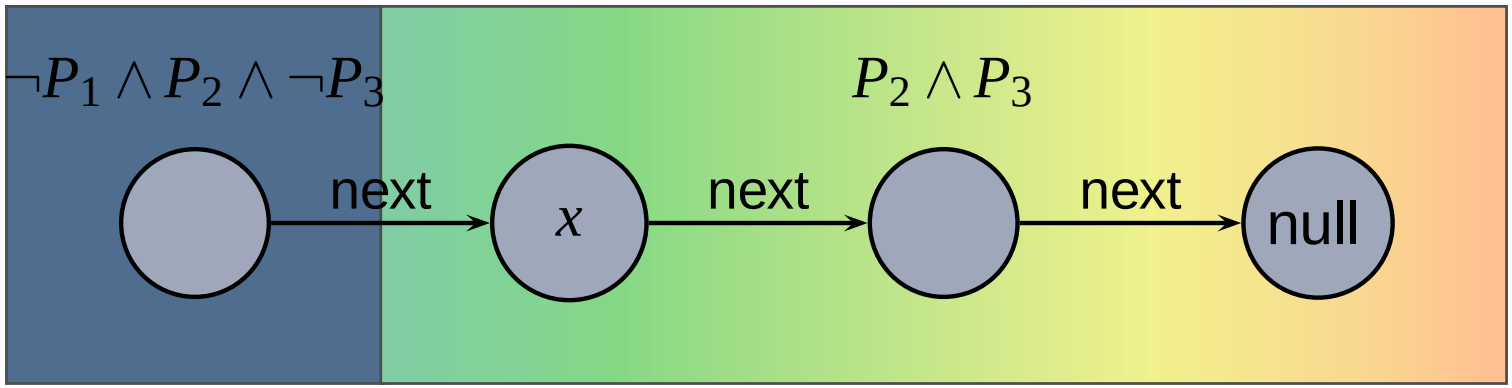
Abstract Post on Boolean Heaps

$$P_1 = \{ v \mid v = x \} \quad P_2 = \{ v \mid \mathsf{next}^*(v, \mathsf{null}) \} \quad P_3 = \{ v \mid \mathsf{next}^*(x, v) \}$$



$$\mathsf{CartesianPost}_c(\forall v \,.\, P_1 \wedge P_2 \wedge P_3 \;\vee\; \neg P_1 \wedge P_2 \wedge P_3)$$

for command $c = (\texttt{x:=x.next})$



$$\forall v \,.\, \neg P_1 \wedge P_2 \wedge \neg P_3 \;\vee\; P_2 \wedge P_3$$

Cartesian Post

$$\mathsf{CartesianPost}(\forall v \,.\, \bigvee_i C_i)$$

$$= \forall v \,.\, \bigvee_i \bigwedge \{\, P \mid C_i \models \mathsf{wlp}(P) \,\}$$

Cartesian Post

Compute effect of heap updates locally

- for each abstract object C_i
- and independently for each heap predicate P

Cartesian Post

$$\text{CartesianPost}(\forall v \,.\, \bigvee_i C_i)$$

$$= \forall v \,.\, \bigvee_i \bigwedge \{\, P \mid C_i \models \text{wlp}(P) \,\}$$

In practice: precompute abstract weakest preconditions

$$\text{wlp}^{\#}(P) = \bigvee \{\, \phi \in \text{BoolExp}(\text{Pred}) \mid \phi \models \text{wlp}(P) \,\}$$

Cartesian Post

Compute effect of heap updates locally

- for each abstract object C_i
- and independently for each heap predicate P

Cartesian Post

$$\mathsf{CartesianPost}(\forall v \,.\, \bigvee_i C_i)$$

$$= \forall v \,.\, \bigvee_i \bigwedge \{\, P \mid C_i \models \mathsf{wlp}(P) \,\}$$

In practice: precompute abstract weakest preconditions

$$\mathsf{wlp}^{\#}(P) = \bigvee \{\, \phi \in \mathsf{BoolExp}(\mathsf{Pred}) \mid \phi \models \mathsf{wlp}(P) \,\}$$

Cartesian Post

Same advantages as for predicate abstraction:

- ➜ abstraction reduced to checking verification conditions
- ➜ requires $\mathcal{O}(n^k)$ decision procedure calls (in practice)
- ➜ abstract transformer computed once for the whole analysis
- ➜ best abstract post computable from Cartesian post.

What is wlp(P)?

where P is not an assertion on states, but defined by a formula in a variable v ranging over heap objects, such as next$^*(x, v)$

Heap Predicates

Denotation of formulae with free variables v:

$$[\![\mathsf{next}(v) = x]\!] = \lambda s \in \mathsf{State} \,.\, \{\, o \in \mathsf{Obj} \mid \mathsf{next}_s\ o = x_s \,\}$$

$$\text{or } [\![\mathsf{next}(v) = x]\!] = \lambda o \in \mathsf{Obj} \,.\, \{\, s \in \mathsf{State} \mid \mathsf{next}_s\ o = x_s \,\}$$

Heap Predicates

Denotation of formulae with free variables v:

$$[\![\mathsf{next}(v) = x]\!] = \lambda s \in \mathsf{State} \,.\, \{\, o \in \mathsf{Obj} \mid \mathsf{next}_s\, o = x_s \,\}$$

$$\text{or } [\![\mathsf{next}(v) = x]\!] = \lambda o \in \mathsf{Obj} \,.\, \{\, s \in \mathsf{State} \mid \mathsf{next}_s\, o = x_s \,\}$$

Heap predicates

$$\mathsf{HeapPred} \stackrel{def}{=} \mathsf{Obj} \to 2^{\mathsf{State}}$$

$$[\![\phi(v)]\!] \stackrel{def}{=} \lambda o \,.\, \{\, s \in \mathsf{State} \mid s, [v \mapsto o] \models \phi(v) \,\}$$

Heap Predicate Transformers

Remember: $\mathsf{HeapPred} = \mathsf{Obj} \to 2^{\mathsf{State}}$.

Lift predicate transformers post and wlp to heap predicates.

$$\begin{aligned} \mathsf{lift} &\in (2^{\mathsf{State}} \to 2^{\mathsf{State}}) \to \mathsf{HeapPred} \to \mathsf{HeapPred} \\ \mathsf{lift}\ \tau\ p &= \lambda o \,.\, \tau\ (p\ o) \end{aligned}$$

Heap Predicate Transformers

Remember: $\mathsf{HeapPred} = \mathsf{Obj} \to 2^{\mathsf{State}}$.

Lift predicate transformers post and wlp to heap predicates.

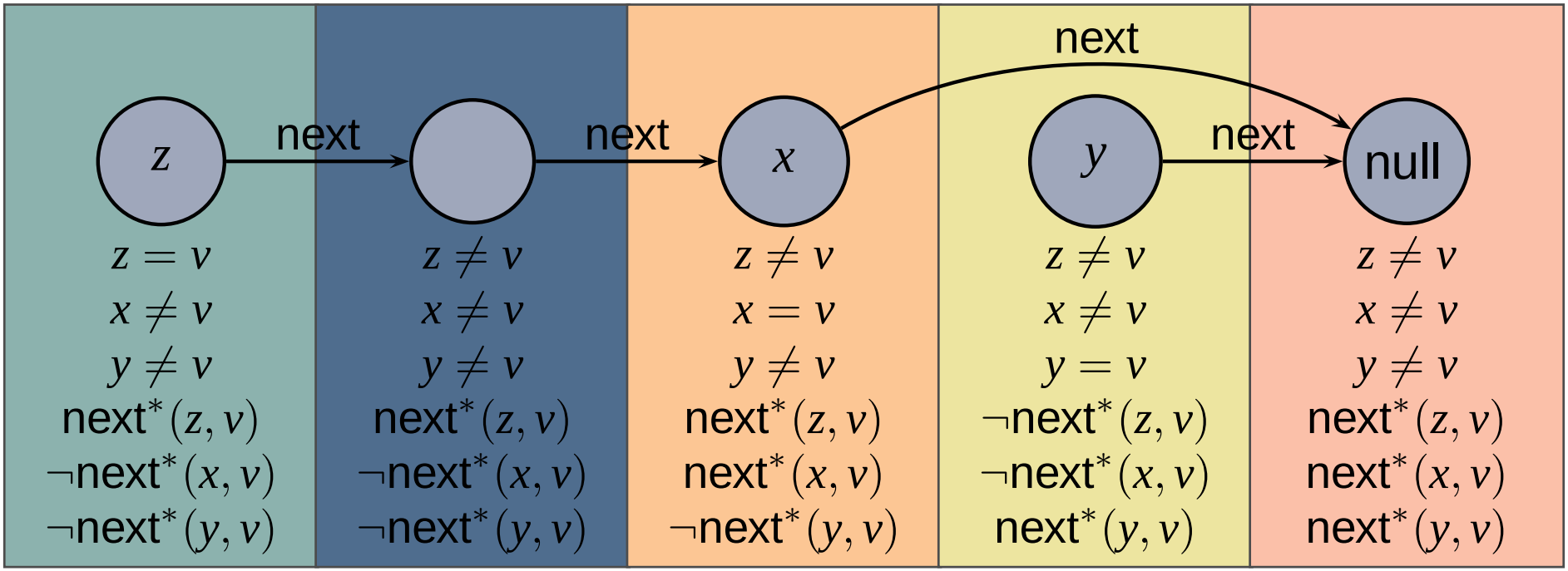
$$\begin{aligned} \mathsf{lift} &\in (2^{\mathsf{State}} \to 2^{\mathsf{State}}) \to \mathsf{HeapPred} \to \mathsf{HeapPred} \\ \mathsf{lift}\ \tau\ p &= \lambda o \,.\, \tau\ (p\ o) \end{aligned}$$

Definition

Heap predicate transformers :

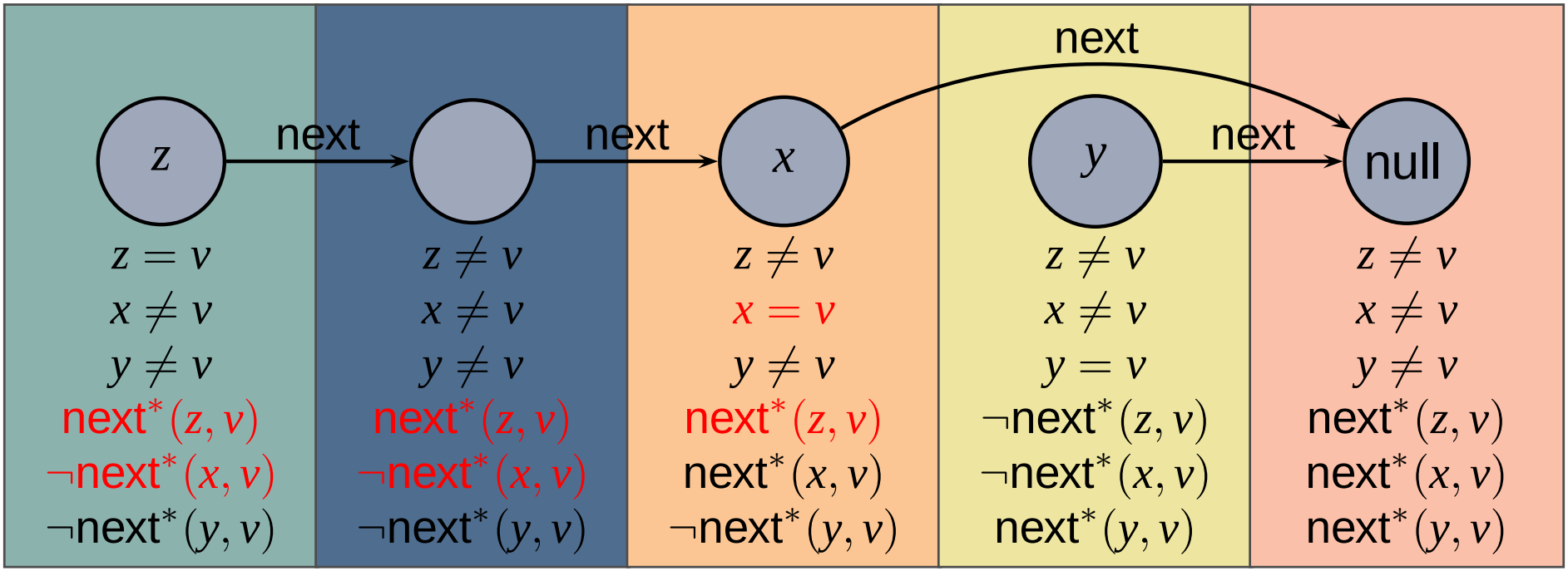
$$\begin{aligned} \mathsf{hpost}, \mathsf{hwlp} &\in \mathsf{Com} \to \mathsf{HeapPred} \to \mathsf{HeapPred} \\ \mathsf{hpost}\ c &\stackrel{def}{=} \mathsf{lift}\ (\mathsf{post}\ c) \\ \mathsf{hwlp}\ c &\stackrel{def}{=} \mathsf{lift}\ (\mathsf{wlp}\ c) \end{aligned}$$

Destructive Updates and Non-local Properties



command $c = (\texttt{x.next:=y})$

Destructive Updates and Non-local Properties

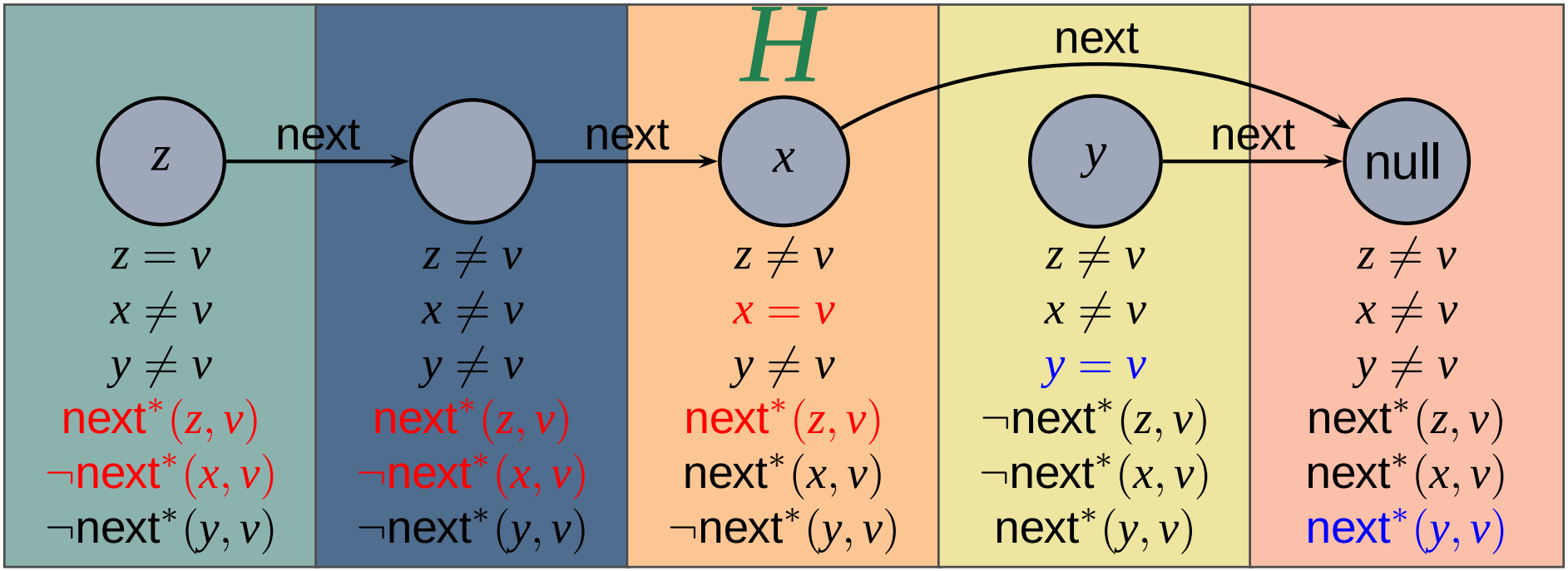


command $c = (\texttt{x.next:=y})$

$$\mathrm{wlp}_c^{\#}(\mathrm{next}^*(z,v)) = \begin{array}{l} \mathrm{next}^*(z,v) \wedge \neg\mathrm{next}^*(x,v) \ \vee \\ \mathrm{next}^*(z,v) \wedge x = v \end{array}$$

$$\mathrm{wlp}_c^{\#}(P) = \bigvee \{\, \phi \in \mathrm{BoolExp}(\mathrm{Pred}) \mid \phi \models \mathrm{wlp}_c(P) \,\}$$

Destructive Updates and Non-local Properties



command $c = (\texttt{x.next:=y})$

$$\text{wlp}_c^{\#}(H, \text{next}^*(z, v)) = \begin{array}{l} \text{next}^*(z, v) \wedge \neg\text{next}^*(x, v) \ \vee \\ \text{next}^*(z, v) \wedge x = v \ \vee \ y = v \ \vee \ \text{next}^*(y, v) \end{array}$$

$$\text{wlp}_c^{\#}(H, P) = \bigvee \{ \phi \in \text{BoolExp}(\text{Pred}) \mid H \wedge \phi \models \text{wlp}_c(P) \}$$

Context-sensitive Cartesian Post

$$\alpha \circ \mathsf{post}\gamma \circ (H) \models \mathsf{CartesianPost}(\Gamma, \forall v\,.\, \bigvee_i C_i \text{ as } H)$$

$$= \forall v\,.\, \bigvee_i \bigwedge \{\, P \mid \Gamma \wedge C_i \models \mathsf{wlp}(P) \,\}$$

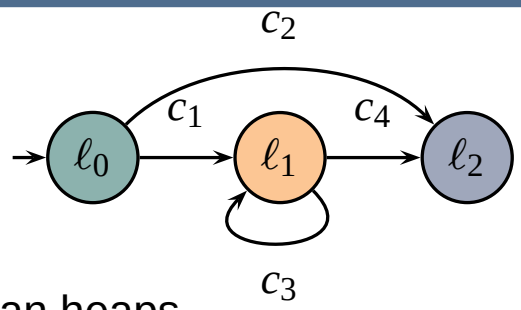
Context-sensitive Cartesian Post

Compute effect of heap updates locally

- for each abstract object C_i
- and independently for each heap predicate P
- but take into account some global information Γ with $H \models \Gamma$

On-Demand Abstraction

CFG

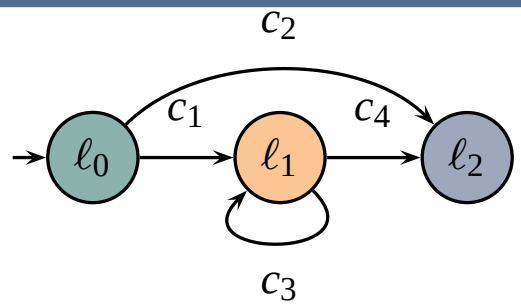


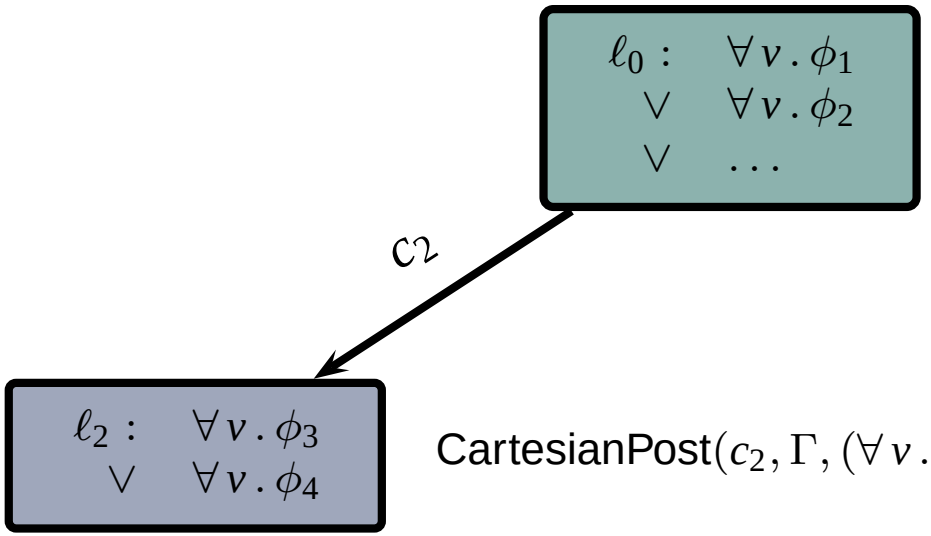
$$\begin{array}{rl} \ell_0 : & \forall v \,.\, \phi_1 \\ \vee & \forall v \,.\, \phi_2 \\ \vee & \ldots \end{array}$$

Initial Boolean heaps

On-Demand Abstraction

CFG

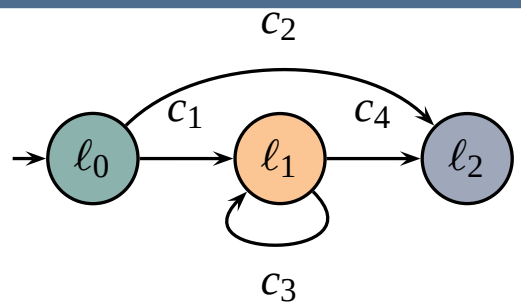


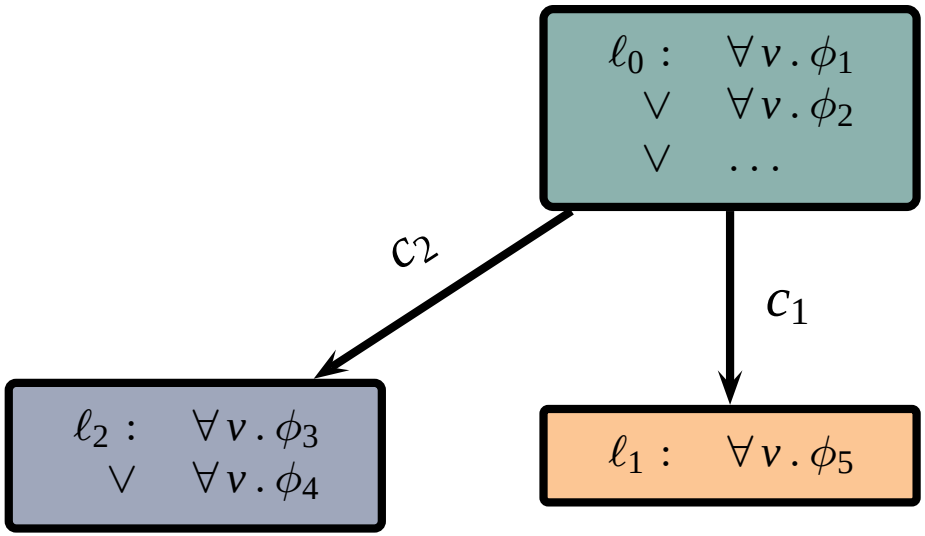


$\mathsf{CartesianPost}(c_2, \Gamma, (\forall v \,.\, \phi_1) \vee (\forall v \,.\, \phi_2) \vee \ldots)$

On-Demand Abstraction

CFG

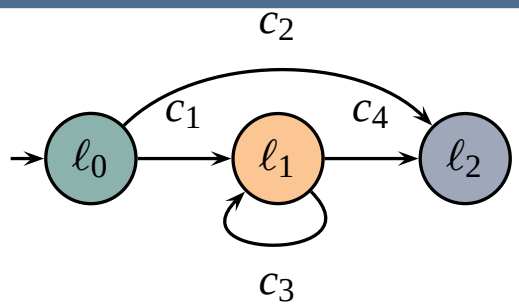


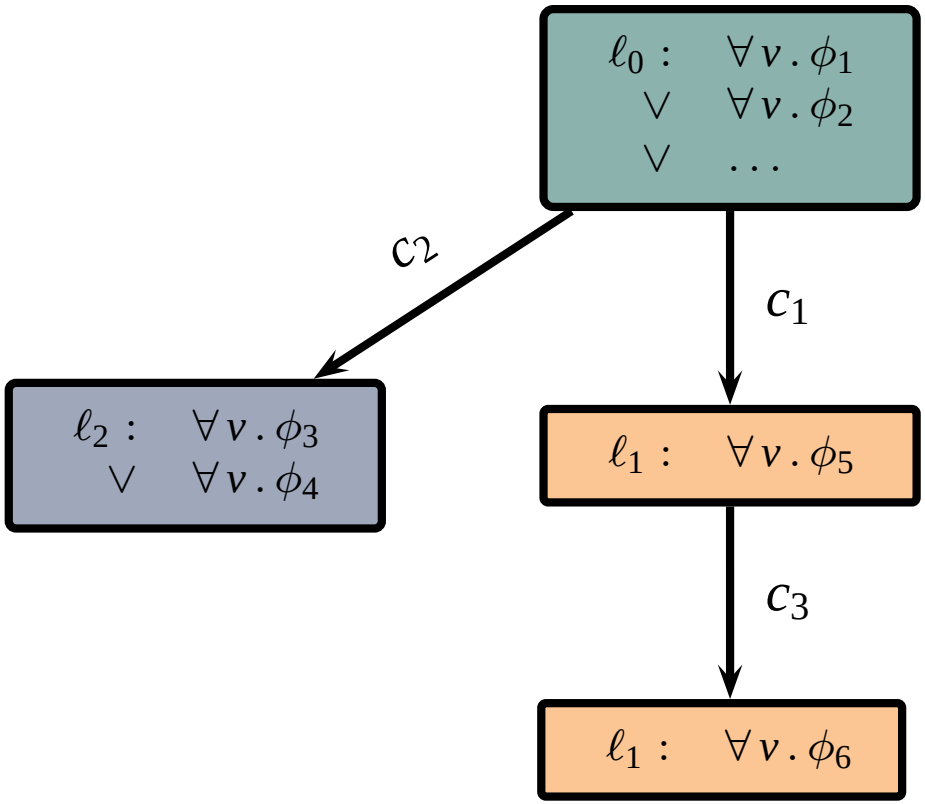


$\mathsf{CartesianPost}(c_3, \Gamma, (\forall v \,.\, \phi_1) \vee (\forall v \,.\, \phi_2) \vee \ldots)$

On-Demand Abstraction

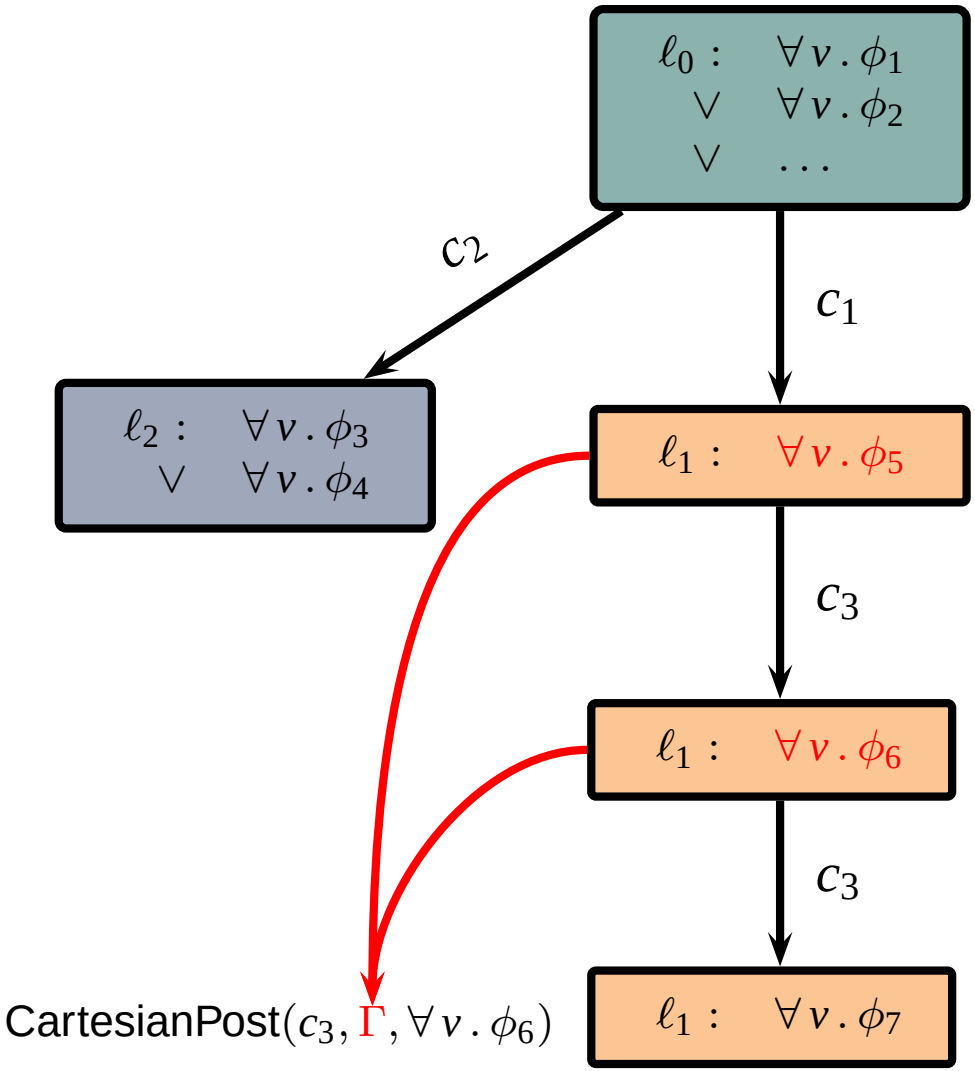
CFG



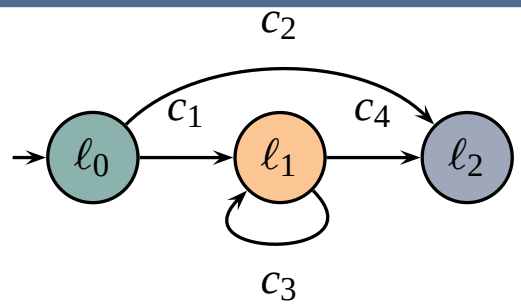


CartesianPost$(c_3, \Gamma, \forall v \,.\, \phi_5)$

On-Demand Abstraction



CFG



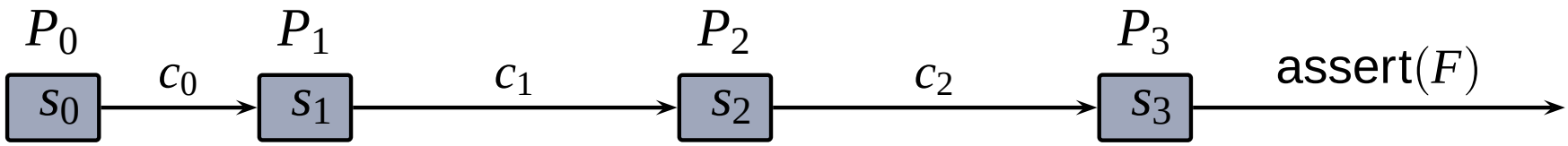
- abstraction takes into account already discovered Boolean heaps at each prog. location.
- abstract transformers are precomputed and recomputed on-demand only if Γ changes.
- decision procedure calls are cached 'semantically' to make recomputation fast.

Outline

1. Boolean heaps (abstract domain)
2. Cartesian post (abstract transformer)
3. Abstraction refinement

Counterexample Guided Abstraction Refinement

Abstract error trace

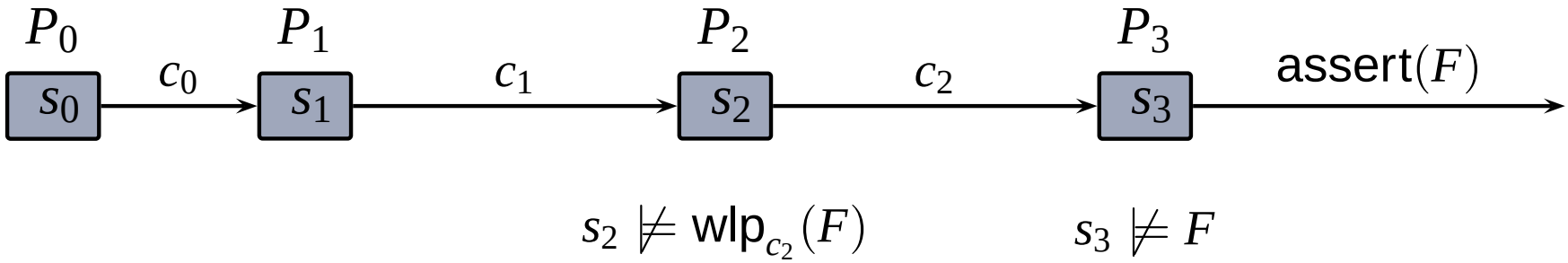


$s_3 \not\models F$

assertion fails

Counterexample Guided Abstraction Refinement

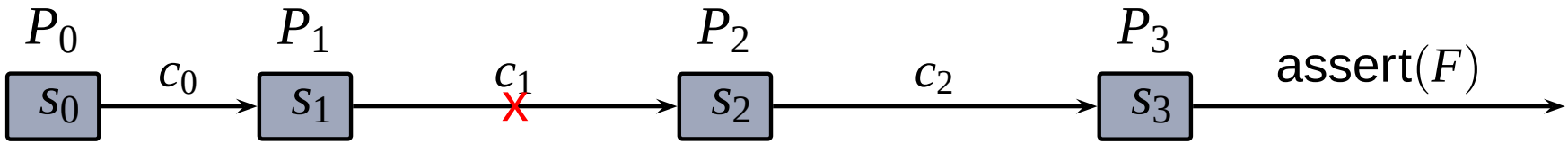
Abstract error trace



backwards analyze error trace

Counterexample Guided Abstraction Refinement

Abstract error trace

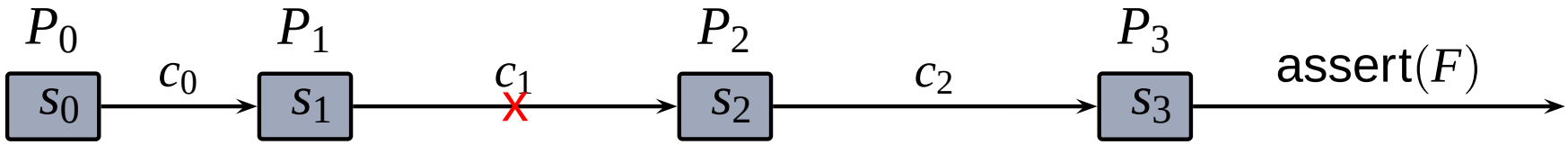


$s_1 \models \mathsf{wlp}_{c_1;c_2}(F)$ $\quad$ $s_2 \not\models \mathsf{wlp}_{c_2}(F)$ $\quad$ $s_3 \not\models F$

error trace is spurious

Counterexample Guided Abstraction Refinement

Abstract error trace



$s_1 \models \mathsf{wlp}_{c_1;c_2}(F)$ $\quad$ $s_2 \not\models \mathsf{wlp}_{c_2}(F)$ $\quad$ $s_3 \not\models F$

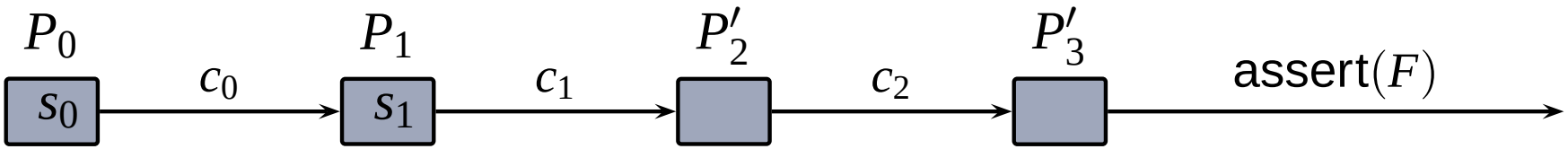
error trace is spurious

Lazy Abstraction [Henzinger, Jhala, Majumdar POPL 2002]

Add new predicates along spurious part of path...

$$Preds'_3 = P_3 \cup \mathsf{atoms}(F)$$
$$Preds'_2 = P_2 \cup \mathsf{atoms}(\mathsf{wlp}_{c_2}(F))$$

...and keep safe part of path



Progress of Abstraction Refinement

Theorem (Progress)

If analysis is based on best abstract post $\text{post}^{\#}$ *then refinement step eliminates spurious error trace.*

Without progress abstraction refinement might run into a deadlock where the same counterexample is produced over and over again, but no new predicates are generated.

Progress of Abstraction Refinement

Theorem (Progress)

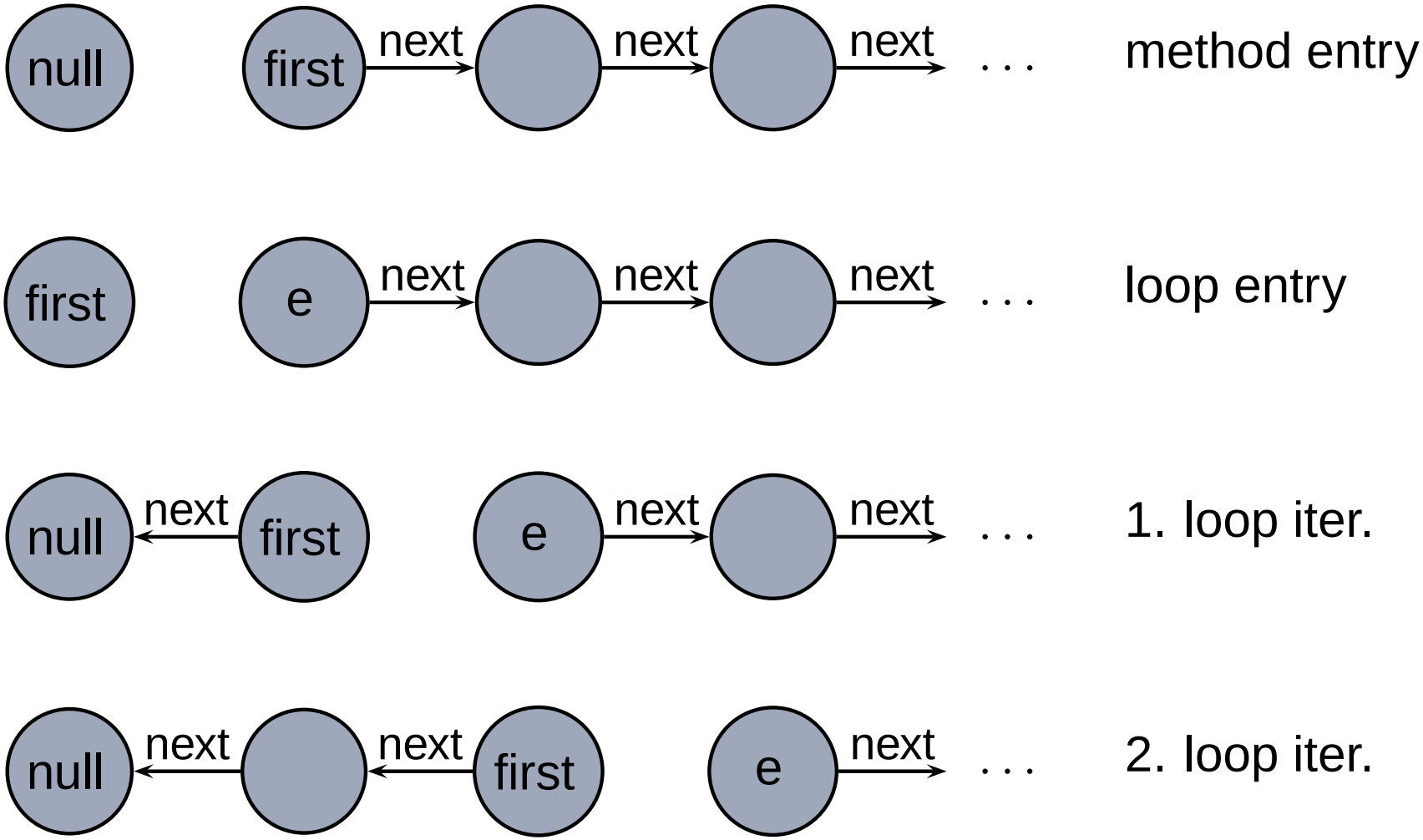
If analysis is based on best abstract post $\text{post}^{\#}$ *then refinement step eliminates spurious error trace.*

Without progress abstraction refinement might run into a deadlock where the same counterexample is produced over and over again, but no new predicates are generated.

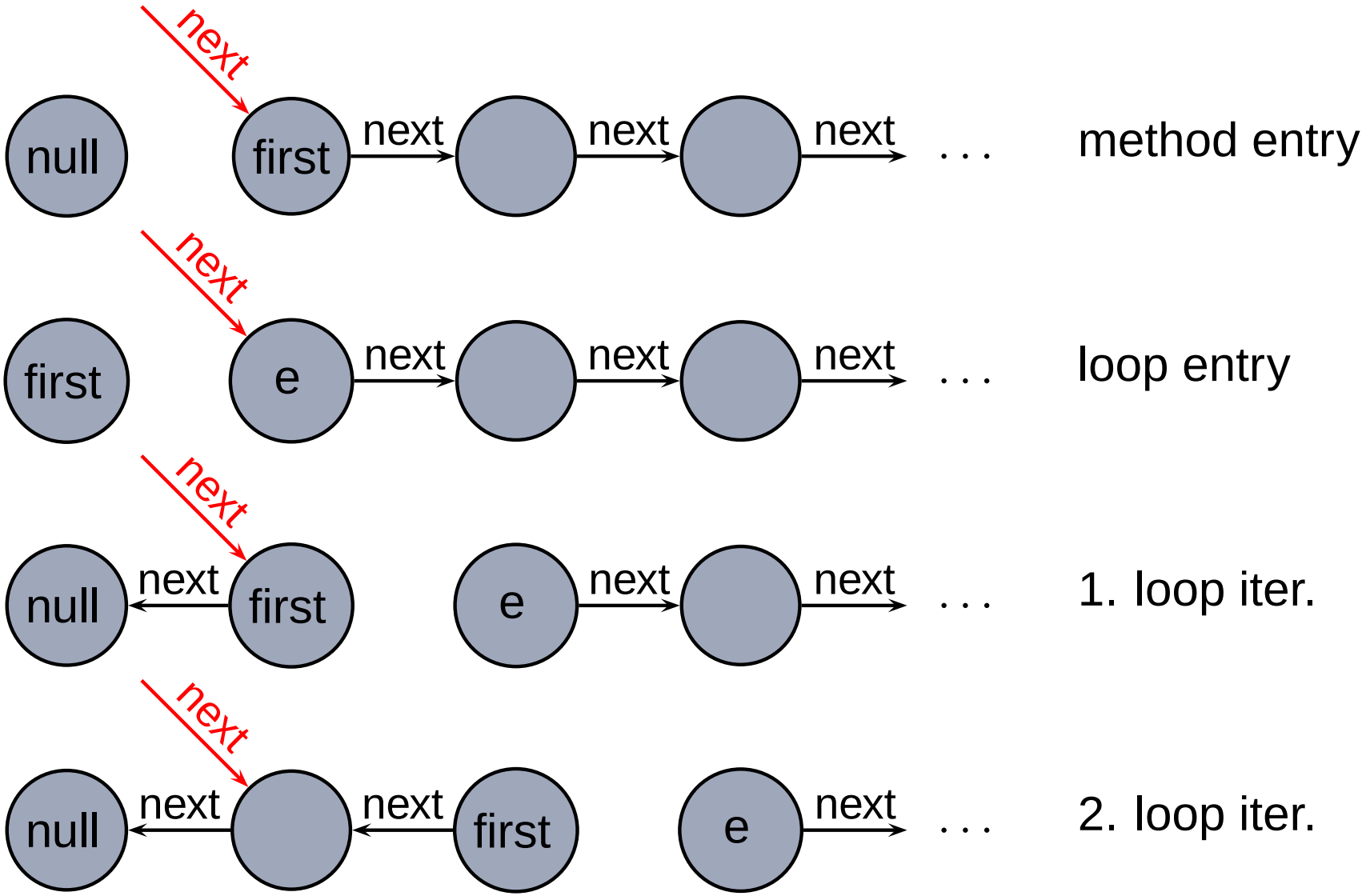
Does progress property also hold for Cartesian post?

Is progress even relevant in practice?

List Reversal

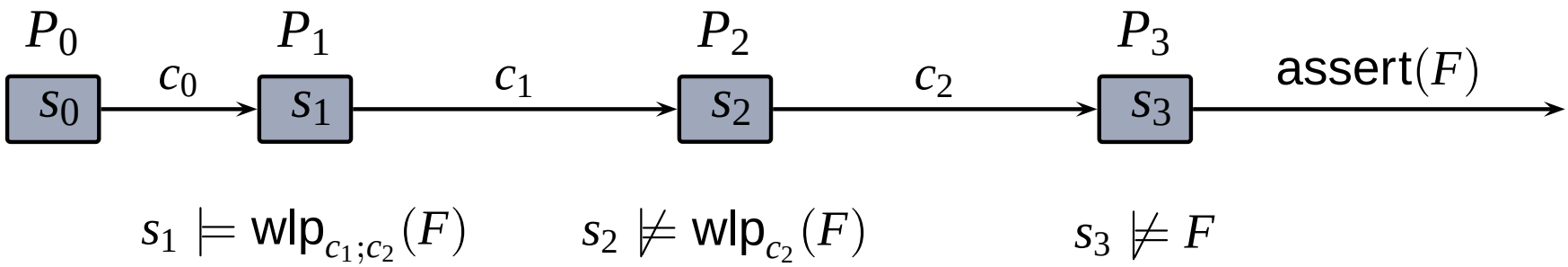


List Reversal



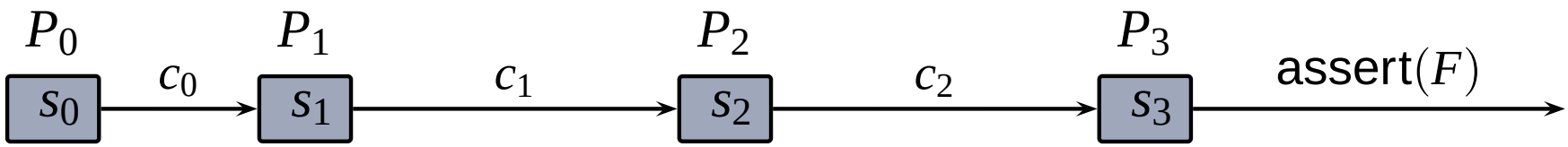
How do we maintain progress property for Cartesian post?

Use counterexample to refine not only abstract domain, but also abstract post.



How do we maintain progress property for Cartesian post?

Use counterexample to refine not only abstract domain, but also abstract post.



$s_1 \models \mathsf{wlp}_{c_1;c_2}(F)$ $\quad$ $s_2 \not\models \mathsf{wlp}_{c_2}(F)$ $\quad$ $s_3 \not\models F$

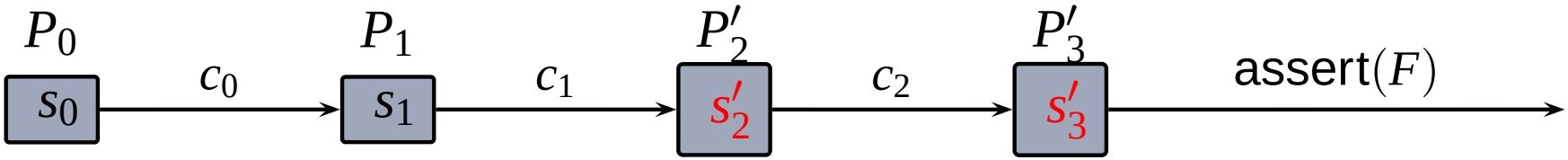
$$
\begin{array}{rll}
s_1 & \models \mathsf{wlp}_{c_1;c_2}(F) & \text{implies} \\
\mathsf{post}_{c_1}(s_1) & \models \mathsf{wlp}_{c_2}(F) & \text{implies} \\
\mathsf{post}_{c_1;c_2}(s_1) & \models F &
\end{array}
$$

Weakest preconditions along spurious part of trace are invariants for that trace.

Idea

Conjoin image of Cartesian post with abstract weakest preconditions

$$P'_2 = P_2 \cup \text{atoms}(\text{wlp}_{c_2}(F))$$
$$P'_3 = P_3 \cup \text{atoms}(F)$$



$$s'_2 = \text{CartesianPost}(s_1) \ \wedge \ \alpha[P'_2](\text{wlp}_{c_2}(F))$$
$$s'_3 = \text{CartesianPost}(s'_2) \ \wedge \ \alpha[P'_3](F)$$

➜ Progress is guaranteed

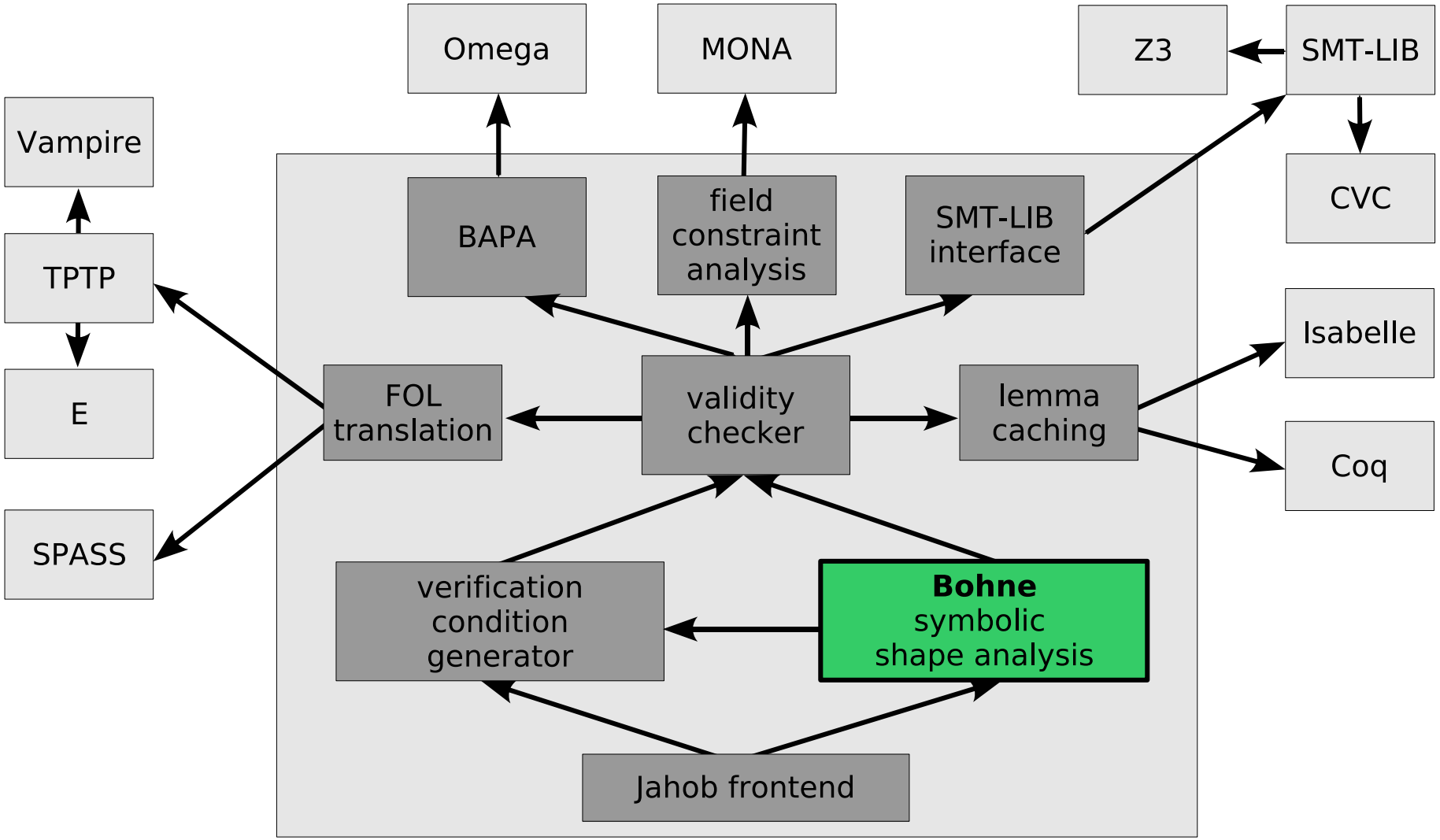
➜ Counterexamples guide splitting of abstract objects/states.

Bohne as Part of Jahob

Jahob system

- verification of data structure consistency properties
- Bohne is a component of Jahob
- Jahob uses Boolean algebra of sets and relations to specify procedure contracts
  ➜ fits nicely with Boolean heaps
- Jahob integrates reasoning procedures for many theories
  ➜ Bohne can be used beyond shape analysis

Jahob System Architecture



Conclusion

Bohne - symbolic shape analyzer

- verifies complex user-specified properties of Java programs
  - procedure contracts
  - data structure invariants
- infers loop invariants automatically
  - disjunctions of universally quantified Boolean combinations of predicates on heap objects
  - predicates are inferred automatically

Predicate Abstraction

Software model-checker: SLAM, BLAST, ARMC, MAGIC, ACSAR, . . .
Successfully applied to control-intensive software, e.g. device drivers.

Benefits of predicate abstraction

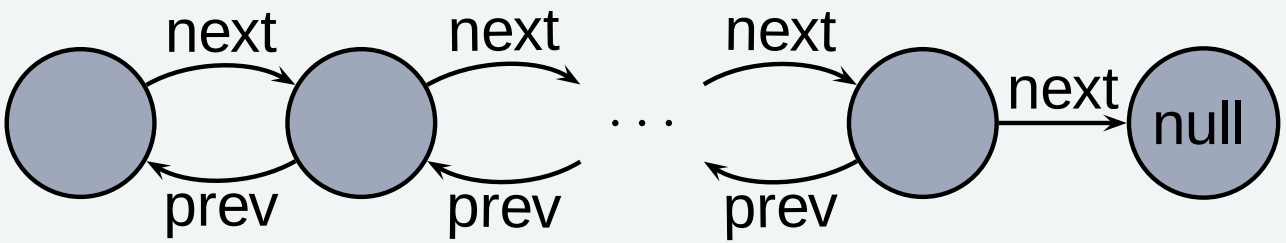
- generic framework
- useful even if predicates are manually supplied
- combination with abstraction refinement results in fully automated verification technique

Bohne has all this.

Quantified Invariants

Example

Doubly-linked lists



field prev is inverse of field next:

$$\forall v \,.\, \mathsf{next}(v) \neq \mathsf{null} \;\rightarrow\; \mathsf{prev}(\mathsf{next}(v)) = v$$

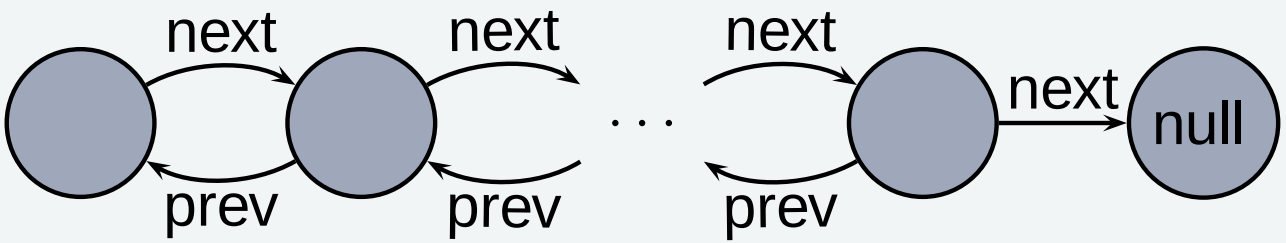
field next is acyclic:

$$\forall v \,.\, \mathsf{next}^*(v, \mathsf{null})$$

Quantified Invariants

Example

Doubly-linked lists



field prev is inverse of field next:

$$\forall v \,.\, \text{next}(v) \neq \text{null} \;\rightarrow\; \text{prev}(\text{next}(v)) = v$$

field next is acyclic:

$$\forall v \,.\, \text{next}^*(v, \text{null})$$

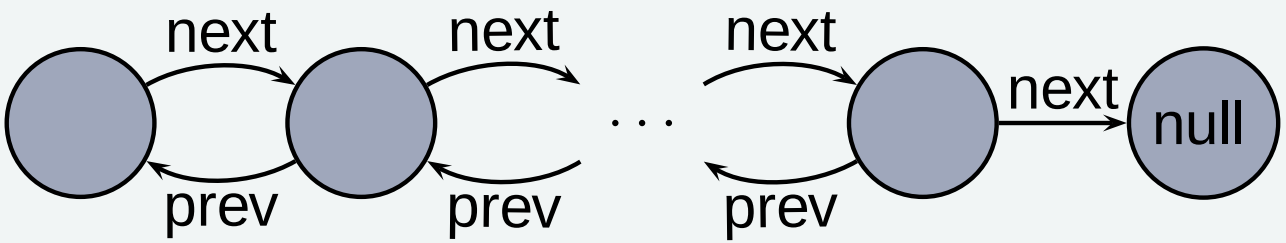
Potential solutions with classical predicate abstraction

- quantified predicates
  - ➜ decision procedures might not be able to handle quantifiers
  - ➜ finding the right predicates is as hard as finding the invariant

Quantified Invariants

Example

Doubly-linked lists



field prev is inverse of field next:

$$\forall v \,.\, \text{next}(v) \neq \text{null} \;\rightarrow\; \text{prev}(\text{next}(v)) = v$$

field next is acyclic:

$$\forall v \,.\, \text{next}^*(v, \text{null})$$

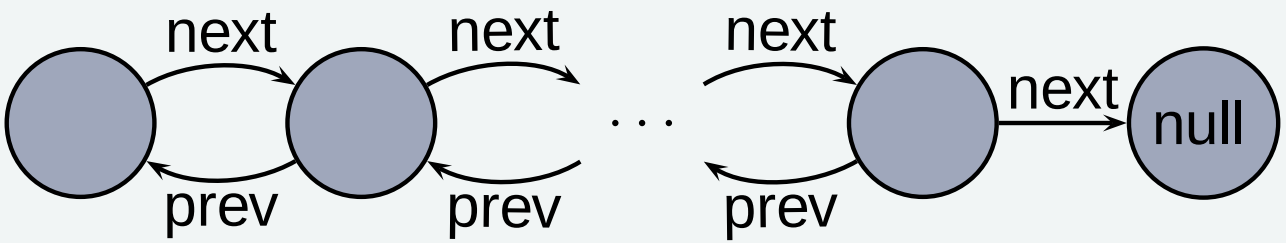
Potential solutions with classical predicate abstraction

- quantified predicates
- Skolemization [Flanagan, Qadeer]
  ➜ does not work well for non-local properties such as reachability

Quantified Invariants

Example

Doubly-linked lists



field prev is inverse of field next:

$$\forall v \,.\, \text{next}(v) \neq \text{null} \;\rightarrow\; \text{prev}(\text{next}(v)) = v$$

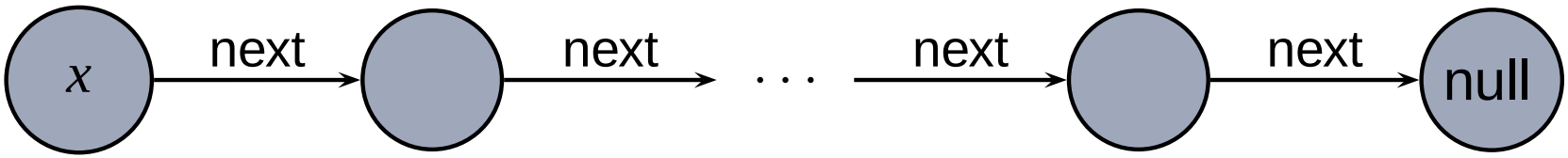
field next is acyclic:

$$\forall v \,.\, \text{next}^*(v, \text{null})$$

Potential solutions with classical predicate abstraction

- quantified predicates
- Skolemization [Flanagan, Qadeer]
- indexed predicates [Lahiri]
  ➜ quantified invariants, but no disjunctive completion

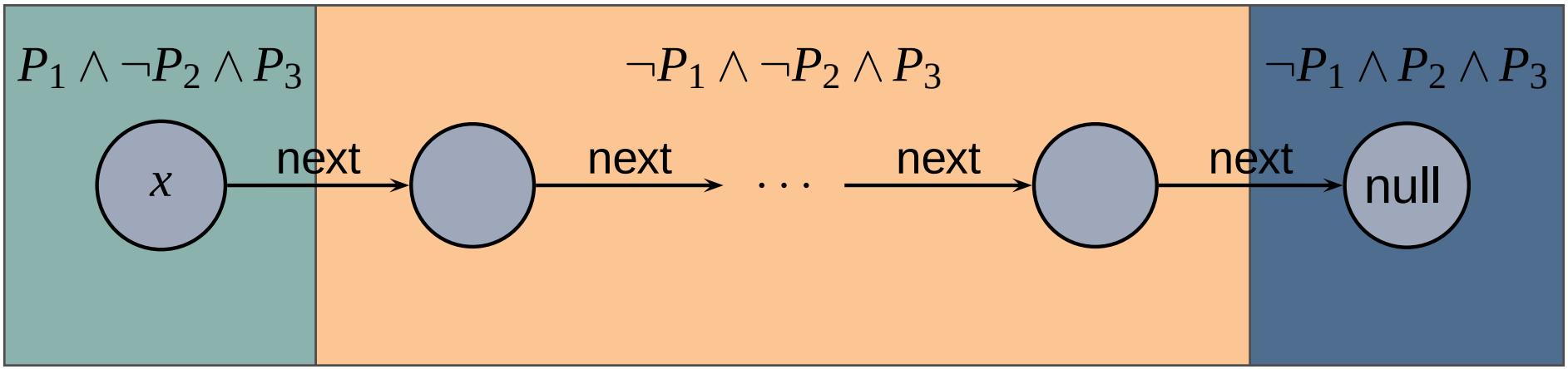
Three-valued shape analysis [Sagiv, Reps, Wilhelm]



Three-valued shape analysis [Sagiv, Reps, Wilhelm]

Partition heap according to finitely many predicates on heap objects.

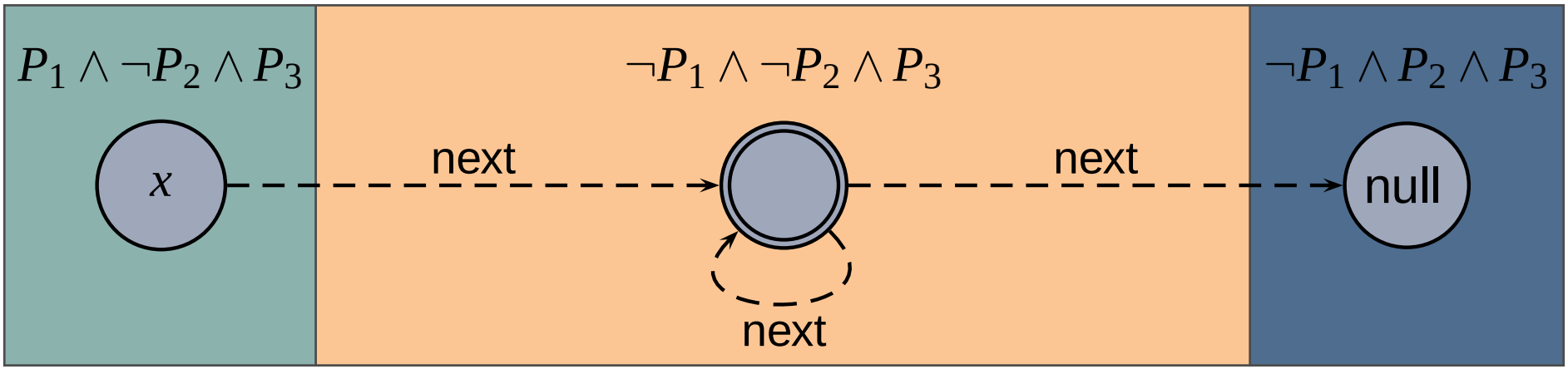
$$P_1 = \{ v \mid v = x \} \quad P_2 = \{ v \mid v = \mathsf{null} \} \quad P_3 = \{ v \mid \mathsf{next}^*(x, v) \}$$



Three-valued shape analysis [Sagiv, Reps, Wilhelm]

Partition heap according to finitely many predicates on heap objects.

$$P_1 = \{ v \mid v = x \} \quad P_2 = \{ v \mid v = \text{null} \} \quad P_3 = \{ v \mid \text{next}^*(x, v) \}$$



➜ shape graph

Trend is to move to symbolic approaches or combine with symbolic approaches (e.g. lazy shape analysis).