Boolean Heaps

Andreas Podelski Thomas Wies
{podelski,wies}@mpi-inf.mpg.de

Max-Planck-Institut für Informatik
Saarbrücken, Germany

Static Analysis Symposium
London, September 2005

Motivation

Predicate Abstraction vs. Three-valued Shape Analysis

Predicate Abstraction

(e.g. SLAM)

transition graph

- nodes $\approx$ states
- edges $\approx$ transitions

abstract by state predicates

$\leadsto$ graph over abstract states

Three-valued Shape Analysis

(TVLA)

heap graph

- nodes $\approx$ heap objects
- edges $\approx$ pointer fields

abstract by heap predicates

$\leadsto$ graph over abstract objects

Motivation

Predicate Abstraction vs. Three-valued Shape Analysis

Predicate Abstraction

(e.g. SLAM)

transition graph

- nodes ≈ states
- edges ≈ transitions

abstract by state predicates

⇝ graph over abstract states

Three-valued Shape Analysis

(TVLA)

heap graph

- nodes ≈ heap objects
- edges ≈ pointer fields

abstract by heap predicates

⇝ graph over abstract objects

Problem: How can one cast the idea of predicates on heap objects in the framework of predicate abstraction?

Overview

Predicate Abstraction vs. Boolean Heap Programs

Predicate Abstraction

Concrete command:

c

State predicates:

$Pred = \{p_1, \ldots, p_n\}$

Abstract boolean program:

var $p_1, \ldots, p_n$: boolean
for each $p_i \in Pred$ **do**
  if $\mathrm{wp}^{\#}\ c\ p_i$ **then** $p_i :=$ true
  else if $\mathrm{wp}^{\#}\ c\ (\neg p_i)$ **then** $p_i :=$ false
  else $p_i := *$

Example

Concrete command:

var x : integer

$x := x + 1$

State predicates:

$p_1 \stackrel{def}{=} x = 0, \qquad p_2 \stackrel{def}{=} x > 0$

Abstract boolean program:

var p_1, p_2 : boolean
if false **then** $p_1 :=$ true
else if $p_1 \vee p_2$ **then** $p_1 :=$ false
else $p_1 := *$
if $p_1 \vee p_2$ **then** $p_2 :=$ true
else if $\neg p_1 \wedge \neg p_2$ **then** $p_2 :=$ false
else $p_2 := *$

Predicate Abstraction vs. Boolean Heap Programs

Predicate Abstraction

Concrete command:

c

State predicates:

$Pred = \{p_1, \ldots, p_n\}$

Abstract boolean program:

var $p_1, \ldots, p_n$: boolean

for each $p_i \in Pred$ **do**

  if $\mathrm{wp}^{\#}\ c\ p_i$ **then** $p_i :=$ true

  else if $\mathrm{wp}^{\#}\ c\ (\neg p_i)$ **then** $p_i :=$ false

  else $p_i := *$

Boolean Heap Programs

Concrete command:

c

Unary heap predicates:

$Pred = \{p_1(v), \ldots, p_n(v)\}$

Boolean heap program:

var V : set of bitvectors over $Pred$

for each $\overline{p} \in V$ **do**

 for each $p_i \in Pred$ **do**

  if $\overline{p} \rightarrow \mathrm{hwp}^{\#}\ c\ p_i$

   then $\overline{p}.p_i :=$ true

  else if $\overline{p} \rightarrow \mathrm{hwp}^{\#}\ c\ (\neg p_i)$

   then $\overline{p}.p_i :=$ false

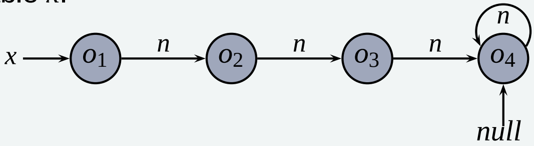
  else $\overline{p}.p_i := *$

Concrete Domain

Concrete domain - sets of program states.

Example

State s containing a 3-element, singly-linked list, accessible by program variable x.



States are represented as logical structures.

$$s \in State \;=\; (Var \rightarrow Heap) \times (Field \rightarrow Heap \rightarrow Heap)$$

Abstract Domain

Setup

Abstract domain

- is a finite lattice of closed formulas Ψ

$$\gamma\ \Psi = \{\, s \in \mathit{State} \mid s \models \Psi \,\}$$

- is parameterized by finite set of abstraction predicates Pred.

Abstraction predicates

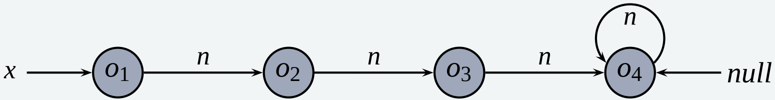
- are formulas in first-order logic or some extension, e.g. FO^{TC}
- have a free variable v
  ➜ denote sets of objects in the heap of a given state
- heap predicates.

Abstract Domain

Heap Predicate Abstraction

Example

$$Pred = \{v = x, v = null, v \in x.n^*\}$$

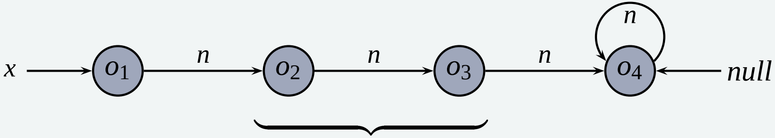


Abstract Domain

Heap Predicate Abstraction

Example

$$Pred = \{v = x, v = null, v \in x.n^*\}$$



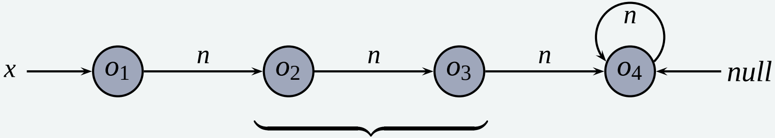
$$\begin{pmatrix} v = x \\ v \neq null \\ v \in x.n^* \end{pmatrix} \quad \begin{pmatrix} v \neq x \\ v \neq null \\ v \in x.n^* \end{pmatrix} \quad \begin{pmatrix} v \neq x \\ v = null \\ v \in x.n^* \end{pmatrix}$$

Abstract Domain

Heap Predicate Abstraction

Example

$$Pred = \{v = x, v = null, v \in x.n^*\}$$



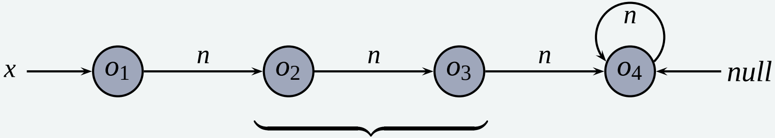
$$\forall v. \begin{pmatrix} v = x \\ \wedge\ v \neq null \\ \wedge\ v \in x.n^* \end{pmatrix} \vee \begin{pmatrix} v \neq x \\ \wedge\ v \neq null \\ \wedge\ v \in x.n^* \end{pmatrix} \vee \begin{pmatrix} v \neq x \\ \wedge\ v = null \\ \wedge\ v \in x.n^* \end{pmatrix}$$

Abstract Domain

Heap Predicate Abstraction

Example

$$Pred = \{v = x, v = null, v \in x.n^*\}$$



$$\forall v. \begin{pmatrix} & v = x \\ \wedge & v \neq null \\ \wedge & v \in x.n^* \end{pmatrix} \vee \begin{pmatrix} & v \neq x \\ \wedge & v \neq null \\ \wedge & v \in x.n^* \end{pmatrix} \vee \begin{pmatrix} & v \neq x \\ \wedge & v = null \\ \wedge & v \in x.n^* \end{pmatrix}$$

Boolean heap

Boolean heap $\approx$ over-approximation of all heap objects.

Abstract Domain

Abstract State

$\approx \quad \forall v.\ \varphi(v)$

$\approx$ Boolean heap

$\approx$ set of bitvectors

Abstract Domain

Abstract State

$\approx \quad \forall v.\ \varphi(v)$

$\approx$ Boolean heap

$\approx$ set of bitvectors

Abstract Domain

$\approx$ disjunctions of Boolean heaps

$\approx$ sets of sets of bitvectors

Programs and Predicate Transformers

Simple guarded command language:

$$c \in Com ::= \mathsf{assume}(b) \mid x := e \mid e_1.f := e_2$$

Programs and Predicate Transformers

Simple guarded command language:

$$c \in Com ::= \textsf{assume}(b) \mid x := e \mid e_1.f := e_2$$

Predicate transformers (defined as usual):

$\textsf{post} \in Com \to 2^{State} \to 2^{State}$	strongest post condition
$\textsf{wp} \in Com \to 2^{State} \to 2^{State}$	weakest (liberal) precondition

Programs and Predicate Transformers

Simple guarded command language:

$$c \in Com ::= \mathsf{assume}(b) \mid x := e \mid e_1.f := e_2$$

Predicate transformers (defined as usual):

$$\mathsf{post} \in Com \to 2^{State} \to 2^{State} \qquad \text{strongest post condition}$$

$$\mathsf{wp} \in Com \to 2^{State} \to 2^{State} \qquad \text{weakest (liberal) precondition}$$

Weakest Preconditions

play important role in predicate abstraction.

- ➜ Can wp be extended to formulas with free variables?
- ➜ Can wp be computed syntactically on formulas?

Heap Predicates

Denotation of a formula with free variables:

$$[\![n(v) = z]\!] = \lambda\, s \in \mathit{State}\,.\, \{\, o \in \mathit{Heap} \mid s\; n\; o = s\; z \,\}$$

or $[\![n(v) = z]\!] = \lambda\, o \in \mathit{Heap}\,.\, \{\, s \in \mathit{State} \mid s\; n\; o = s\; z \,\}$

Heap Predicates

Denotation of a formula with free variables:

$$[\![n(v) = z]\!] = \lambda s \in State\,.\,\{\, o \in Heap \mid s\ n\ o = s\ z \,\}$$

or $$[\![n(v) = z]\!] = \lambda o \in Heap\,.\,\{\, s \in State \mid s\ n\ o = s\ z \,\}$$

Definition

n-ary heap predicates and denotation of formulas:

$$HeapPred[n] \stackrel{def}{=} Heap^n \to 2^{State}$$

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

Heap Predicates

Denotation of a formula with free variables:

$$[\![n(v) = z]\!] = \lambda s \in State\,.\,\{\, o \in Heap \mid s\; n\; o = s\; z \,\}$$

or $$[\![n(v) = z]\!] = \lambda o \in Heap\,.\,\{\, s \in State \mid s\; n\; o = s\; z \,\}$$

Definition

n-ary heap predicates and denotation of formulas:

$$HeapPred[n] \stackrel{def}{=} Heap^n \to 2^{State}$$

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

➜ formulas denote heap predicates

Heap Predicates

Denotation of a formula with free variables:

$$\llbracket n(v) = z \rrbracket = \lambda\, s \in \mathit{State} \,.\, \{\, o \in \mathit{Heap} \mid s\; n\; o = s\; z \,\}$$

or $$\llbracket n(v) = z \rrbracket = \lambda\, o \in \mathit{Heap} \,.\, \{\, s \in \mathit{State} \mid s\; n\; o = s\; z \,\}$$

Definition

n-ary heap predicates and denotation of formulas:

$$\mathit{HeapPred}[n] \stackrel{\mathit{def}}{=} \mathit{Heap}^n \rightarrow 2^{\mathit{State}}$$

$$\llbracket \varphi(\overline{v}) \rrbracket \stackrel{\mathit{def}}{=} \lambda\, \overline{o} \,.\, \{\, s \in \mathit{State} \mid s, [\overline{v} \mapsto \overline{o}] \models \varphi(\overline{v}) \,\}$$

- ➜ formulas denote heap predicates
- ➜ closed formulas denote 0-ary heap predicates $\approx$ state predicates

Heap Predicate Transformers

Remember: $\mathit{HeapPred} = \mathit{Heap}^n \to 2^{\mathit{State}}$.

Lift predicate transformers post and wp to heap predicates.

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

Heap Predicate Transformers

Remember: $HeapPred = Heap^n \to 2^{State}$.

Lift predicate transformers post and wp to heap predicates.

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

Definition

Heap predicate transformers :

$$\begin{aligned} \text{hpost}, \text{hwp} &\in Com \to HeapPred \to HeapPred \\ \text{hpost}\ c &\stackrel{def}{=} \text{lift}\ (\text{post}\ c) \\ \text{hwp}\ c &\stackrel{def}{=} \text{lift}\ (\text{wp}\ c) \end{aligned}$$

Heap Predicate Transformers

Properties

1. Form Galois connection on Boolean algebra of heap predicates:

Heap Predicate Transformers

Properties

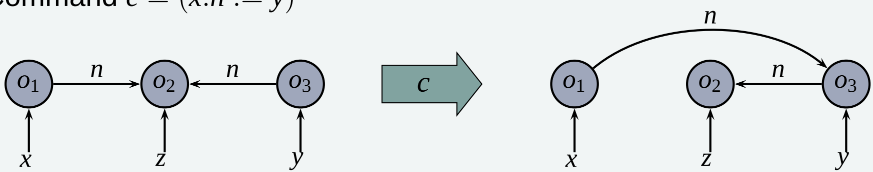
1. Form Galois connection on Boolean algebra of heap predicates:
2. hwp is computed by syntactic substitutions on formulas (all commands are deterministic):

$$
\begin{aligned}
\text{hwp}\ (\text{assume}\ b)\ [\![\varphi(\bar{v})]\!] &= [\![b \rightarrow \varphi(\bar{v})]\!] \\
\text{hwp}\ (x := e)\ [\![\varphi(\bar{v})]\!] &= [\![\varphi(\bar{v})[x := e]]\!] \\
\text{hwp}(e_1.f := e_2)\ [\![\varphi(\bar{v})]\!] &= [\![\varphi(\bar{v})[f := \lambda v\,.\ \text{if}\ v = e_1\ \text{then}\ e_2\ \text{else}\ f(v)]]\!].
\end{aligned}
$$

Weakest Heap Predicate Preconditions

Example

Command $c = (x.n := y)$

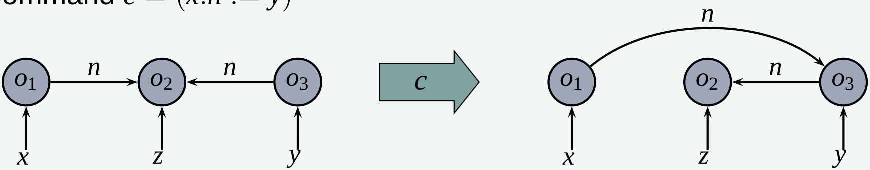


$$s', [v \mapsto o_1] \not\models n(v) = z$$
$$s', [v \mapsto o_3] \models n(v) = z$$

Weakest Heap Predicate Preconditions

Example

Command $c = (x.n := y)$



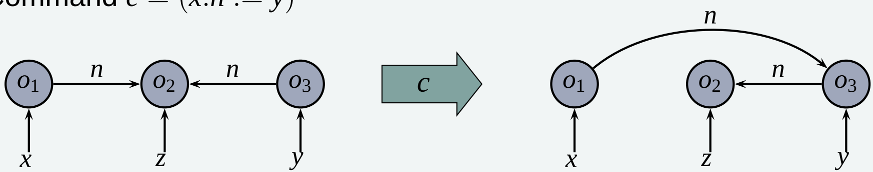
$$s', [v \mapsto o_1] \not\models n(v) = z$$
$$s', [v \mapsto o_3] \models n(v) = z$$

$$\text{hwp}\ c\ [\![n(v) = z]\!] = [\![(n(v) = z)[n := \lambda v.\ \text{if}\ v = x\ \text{then}\ y\ \text{else}\ n(v)]]\!]$$
$$\equiv [\![v = x \wedge y = z \vee v \neq x \wedge n(v) = z]\!]$$

Weakest Heap Predicate Preconditions

Example

Command $c = (x.n := y)$



$s, [v \mapsto o_1] \not\models \mathsf{hwp}\ c\ [\![n(v) = z]\!]$

$s, [v \mapsto o_3] \models \mathsf{hwp}\ c\ [\![n(v) = z]\!]$

$s', [v \mapsto o_1] \not\models n(v) = z$

$s', [v \mapsto o_3] \models n(v) = z$

$$\mathsf{hwp}\ c\ [\![n(v) = z]\!] = [\![(n(v) = z)[n := \lambda v \,.\, \mathsf{if}\ v = x\ \mathsf{then}\ y\ \mathsf{else}\ n(v)]]\!]$$

$$\equiv [\![v = x \wedge y = z \vee v \neq x \wedge n(v) = z]\!]$$

Symbolic Abstract Post

Best abstract post can be computed using hwp:

$$\text{post}^{\#}\ c\ \Psi = \bigwedge \{\ \Phi \in \mathit{AbsDom} \mid \Psi \models \text{hwp}\ c\ \Phi\ \}$$

Question:
Can it be computed efficiently?

Symbolic Abstract Post

Best abstract post can be computed using hwp:

$$\text{post}^{\#}\ c\ \Psi = \bigwedge \{\, \Phi \in \mathit{AbsDom} \mid \Psi \models \text{hwp}\ c\ \Phi \,\}$$

Question:
Can it be computed efficiently?

Use additional Cartesian abstraction
➜ Boolean heap program

Boolean heap program

var V : set of bitvectors over Pred
for each $\overline{p} \in V$ **do**
  for each $p_i \in \mathit{Pred}$ **do**
    if $\overline{p} \models$ hwp $c\ p_i$
      then $\overline{p}.p_i :=$ true
    else if $\overline{p} \models$ hwp $c\ (\neg p_i)$
      then $\overline{p}.p_i :=$ false
    else $\overline{p}.p_i := *$

Symbolic Abstract Post

Best abstract post can be computed using hwp:

$$\text{post}^{\#}\ c\ \Psi = \bigwedge \{\ \Phi \in \mathit{AbsDom} \mid \Psi \models \text{hwp}\ \ c\ \Phi\ \}$$

Question:
Can it be computed efficiently?

Use additional Cartesian abstraction
➜ Boolean heap program

Boolean heap program

var V : set of bitvectors over Pred
for each $\overline{p} \in V$ **do**
  for each $p_i \in \mathit{Pred}$ **do**
    if $\overline{p} \rightarrow \text{hwp}^{\#}\ \ c\ p_i$
      then $\overline{p}.p_i := \text{true}$
    else if $\overline{p} \rightarrow \text{hwp}^{\#}\ \ c\ (\neg p_i)$
      then $\overline{p}.p_i := \text{false}$
    else $\overline{p}.p_i := *$

$$\text{hwp}^{\#}\ c\ (p(v)) \stackrel{\mathit{def}}{=} \bigwedge \{\ \varphi(v) \in \mathcal{BC}(\mathit{Pred}) \mid \varphi(v) \models \text{hwp}\ \ c\ (p(v))\ \}$$

Tool Demo - Bohne

Boolean heaps - nothing else

- joined work with Martin Rinard's group at MIT
- plugin to Hob framework
- underlying logic: MSOL over trees
- more infos: `http://hob.csail.mit.edu`

Bohne verifies

- procedure contracts (specified in a set specification language)
- data structure invariants
- absence of null pointer dereferences.

Conclusion

Main Contributions

- new symbolic approach to shape analysis
- combines key ideas of predicate abstraction and three-valued shape analysis

Future Work

- inter-procedural analysis
- automated abstraction refinement
- combination with integer arithmetic
- ...