

# Boolean and Cartesian Abstraction for Model Checking C Programs

Thomas Ball

Andreas Podelski

Sriram K. Rajamani

A talk by: Mark Kaminski

Software Model Checking Seminar SS 2005

May 12, 2005

## 1 Introduction

- Motivation
- Basics

## 2 Abstraction: Definition and Computation

- Boolean Abstraction
- Cartesian Abstraction
- c2bp

## 3 Refining Abstraction

- Loss of Precision under Cartesian Abstraction
- Refinement Techniques

# Model Checking

- algorithmic technique for hardware and software verification
- analyzes the graph representation of a system's state space

# Model Checking

- algorithmic technique for hardware and software verification
- analyzes the graph representation of a system's state space
- traditionally applied to finite-state systems
  - hardware: circuit design, bus protocols
  - software: communication protocols
- problems: state-space explosion, infinite-state systems

## The Challenges:

- infinite control: recursion
- infinite data: unbounded data types

## The Challenges:

- infinite control: recursion  
    ⇒ pushdown automata
- infinite data: unbounded data types

# Software Model Checking

## The Challenges:

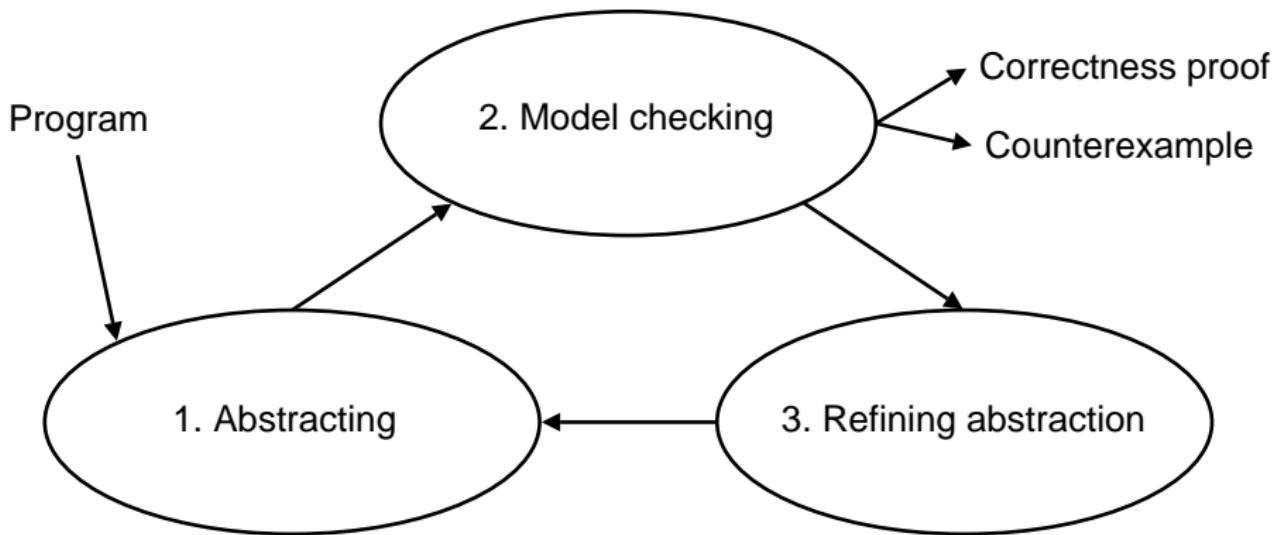
- infinite control: recursion  
⇒ pushdown automata
- infinite data: unbounded data types  
⇒ abstraction

# Software Model Checking

## The Challenges:

- infinite control: recursion  
⇒ pushdown automata
- infinite data: unbounded data types  
⇒ abstraction
- infinite control & infinite data  
⇒ combining the approaches (SLAM)

# The SLAM Process



## Given

- a set **States**
- a transition relation  
 $\rightarrow \subseteq \text{States} \times \text{States}$

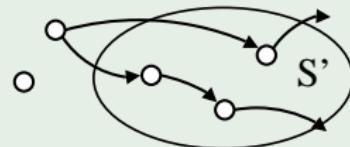
## Given

- a set **States**
- a transition relation  
 $\rightarrow \subseteq \text{States} \times \text{States}$

Let

- $\widetilde{\text{pre}}(S') = \{s \mid \forall s' \text{ s.t. } s \rightarrow s' : s' \in S'\}$

## Example



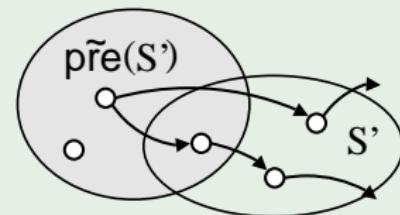
## Given

- a set **States**
- a transition relation  
 $\rightarrow \subseteq \text{States} \times \text{States}$

Let

- $\widetilde{\text{pre}}(S') = \{s \mid \forall s' \text{ s.t. } s \rightarrow s' : s' \in S'\}$

## Example



# $\widetilde{\text{pre}}$ and $\text{post}$

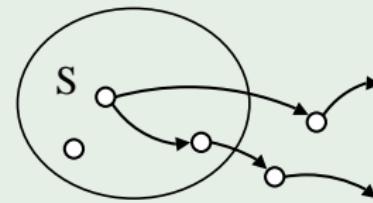
## Given

- a set **States**
- a transition relation  
 $\rightarrow \subseteq \text{States} \times \text{States}$

Let

- $\widetilde{\text{pre}}(S') = \{s \mid \forall s' \text{ s.t. } s \rightarrow s' : s' \in S'\}$
- $\text{post}(S) = \{s' \mid \exists s \in S : s \rightarrow s'\}$

## Example



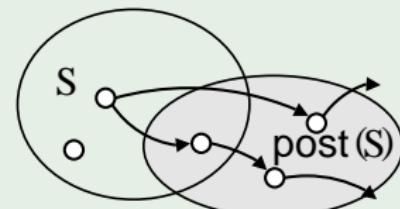
## Given

- a set **States**
- a transition relation  
 $\rightarrow \subseteq \text{States} \times \text{States}$

Let

- $\widetilde{\text{pre}}(S') = \{s \mid \forall s' \text{ s.t. } s \rightarrow s' : s' \in S'\}$
- $\text{post}(S) = \{s' \mid \exists s \in S : s \rightarrow s'\}$

## Example



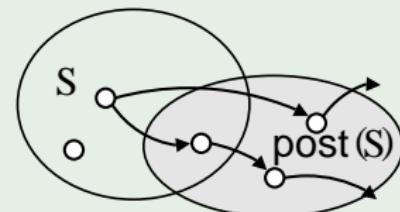
## Given

- a set **States**
- a transition relation  
 $\rightarrow \subseteq \text{States} \times \text{States}$

Let

- $\widetilde{\text{pre}}(S') = \{s \mid \forall s' \text{ s.t. } s \rightarrow s' : s' \in S'\}$
- $\text{post}(S) = \{s' \mid \exists s \in S : s \rightarrow s'\}$
- $\text{post}^*(S) = S \cup \text{post}(S) \cup \dots$

## Example



# Abstraction and Meaning

Given

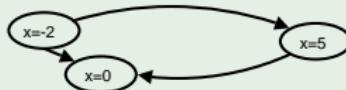
- equiv. relation  $R$
- $\text{AD} = 2^{\text{States}/R}$

The homomorphism

$$\alpha : 2^{\text{States}} \rightarrow \text{AD}$$

is called an *abstraction* function.

Example



# Abstraction and Meaning

Given

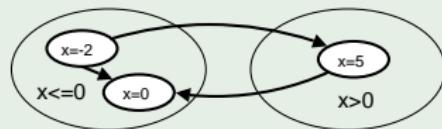
- equiv. relation  $R$
- $\text{AD} = 2^{\text{States}/R}$

The homomorphism

$$\alpha : 2^{\text{States}} \rightarrow \text{AD}$$

is called an *abstraction* function.

Example



# Abstraction and Meaning

Given

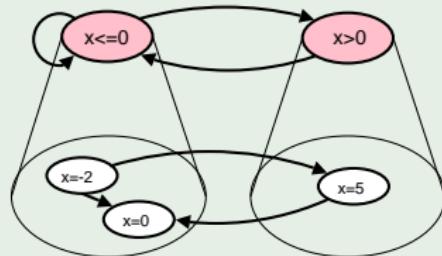
- equiv. relation  $R$
- $\text{AD} = 2^{\text{States}/R}$

The homomorphism

$$\alpha : 2^{\text{States}} \rightarrow \text{AD}$$

is called an *abstraction* function.

Example



# Abstraction and Meaning

Given

- equiv. relation  $R$
- $\text{AD} = 2^{\text{States}/R}$

The homomorphism

$$\alpha : 2^{\text{States}} \rightarrow \text{AD}$$

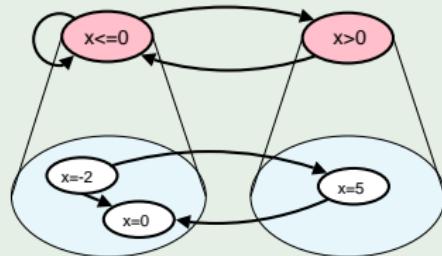
is called an *abstraction* function.

The homomorphism

$$\gamma : \text{AD} \rightarrow 2^{\text{States}}$$

is called a *meaning* function.

Example



# Abstraction and Meaning

Given

- equiv. relation  $R$
- $\text{AD} = 2^{\text{States}/R}$

The homomorphism

$$\alpha : 2^{\text{States}} \rightarrow \text{AD}$$

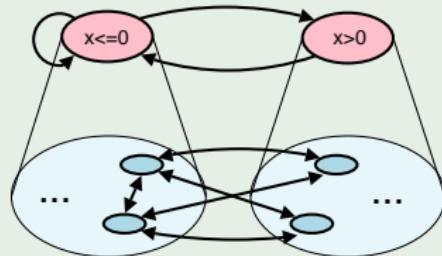
is called an *abstraction* function.

The homomorphism

$$\gamma : \text{AD} \rightarrow 2^{\text{States}}$$

is called a *meaning* function.

Example



## Abstraction and Meaning (2)

- $\alpha$  introduces an *abstract transition system*.
- If  $R$  is finite, so is the abstract system.
  - ⇒ Apply model checking to the abstract system.
- Often  $S \subsetneq \gamma(\alpha(S))$ : loss of precision

# Correctness (Safety)

Given

- initial states  $\text{init} \subseteq \text{States}$
- safe states  $\text{safe} \subseteq \text{States}$

A program is *correct* if  $\text{post}^*(\text{init}) \subseteq \text{safe}$

# Correctness (Safety)

Given

- initial states  $\text{init} \subseteq \text{States}$
- safe states  $\text{safe} \subseteq \text{States}$

A program is *correct* if  $\text{post}^*(\text{init}) \subseteq \text{safe}$

Question: How to check  $\text{post}^*(\text{init}) \subseteq \text{safe}$ ?

# Safe Invariants

A *safe invariant* is a set  $S$  such that

- ①  $S \subseteq \text{safe}$
- ②  $S \supseteq \text{post}(S)$
- ③  $S \supseteq \text{init}$

# Safe Invariants

A *safe invariant* is a set  $S$  such that

- ①  $S \subseteq \text{safe}$
- ②  $S \supseteq \text{post}(S)$
- ③  $S \supseteq \text{init}$

If  $S$  exists, then  $\text{post}^*(\text{init}) \subseteq \text{post}^*(S) \subseteq S \subseteq \text{safe}$

# Safe Invariants

A *safe invariant* is a set  $S$  such that

- ①  $S \subseteq \text{safe}$
- ②  $S \supseteq \text{post}(S)$
- ③  $S \supseteq \text{init}$

If  $S$  exists, then  $\text{post}^*(\text{init}) \subseteq \text{post}^*(S) \subseteq S \subseteq \text{safe}$

**Problem:** How to compute  $S$  for infinite-state transition systems?

# Safe Invariants

A *safe invariant* is a set  $S$  such that

- ①  $S \subseteq \text{safe}$
- ②  $S \supseteq \text{post}(S)$
- ③  $S \supseteq \text{init}$

If  $S$  exists, then  $\text{post}^*(\text{init}) \subseteq \text{post}^*(S) \subseteq S \subseteq \text{safe}$

**Problem:** How to compute  $S$  for infinite-state transition systems?

**Idea:** Compute the safe invariant for a finite abstract system.

# Computing Safe Invariants

## Task

Check  $\text{post}^*(\text{init}) \subseteq \text{safe}$

by computing a safe invariant  $S$  for an abstract transition system.

We define an abstract post operator  $\text{post}^\sharp = \alpha \circ \text{post} \circ \gamma$   
(i.e.  $\text{post}^\sharp(x) = \alpha(\text{post}(\gamma(x)))$ )

We

- ① compute  $S = \gamma(\text{post}^{\sharp*}(\alpha(\text{init})))$
- ② test whether  $S \subseteq \text{safe}$

# Computing Safe Invariants

## Task

Check  $\text{post}^*(\text{init}) \subseteq \text{safe}$

by computing a safe invariant  $S$  for an abstract transition system.

We define an abstract post operator  $\text{post}^\sharp = \alpha \circ \text{post} \circ \gamma$   
(i.e.  $\text{post}^\sharp(x) = \alpha(\text{post}(\gamma(x)))$ )

We

① compute  $S = \gamma(\text{post}^{\sharp*}(\alpha(\text{init})))$

② test whether  $S \subseteq \text{safe}$

Three cases possible:

① test succeeds  $\Rightarrow$  done

② test fails, counterexample valid  $\Rightarrow$  done

③ test fails, counterexample invalid  $\Rightarrow$  refine abstraction

# Defining $\text{post}_{b,c}^\sharp$ : Outline

Question: How to define the abstract post operator s.t. it can be computed efficiently?

# Defining $\text{post}_{\text{b.c}}^\sharp$ : Outline

Question: How to define the abstract post operator s.t. it can be computed efficiently?

We combine *Boolean* and *Cartesian* abstraction:

# Defining $\text{post}_{\text{b.c}}^\sharp$ : Outline

Question: How to define the abstract post operator s.t. it can be computed efficiently?

We combine *Boolean* and *Cartesian* abstraction:

- Boolean abstraction
  - *abstract domain*  $\text{AbsDom}_{\text{bool}}$
  - abstraction function  $\alpha_{\text{bool}} : 2^{\text{States}} \rightarrow \text{AbsDom}_{\text{bool}}$
  - meaning function  $\gamma_{\text{bool}} : \text{AbsDom}_{\text{bool}} \rightarrow 2^{\text{States}}$

# Defining $\text{post}_{\text{b.c}}^{\sharp}$ : Outline

Question: How to define the abstract post operator s.t. it can be computed efficiently?

We combine *Boolean* and *Cartesian* abstraction:

- Boolean abstraction
  - *abstract domain*  $\text{AbsDom}_{\text{bool}}$
  - abstraction function  $\alpha_{\text{bool}} : 2^{\text{States}} \rightarrow \text{AbsDom}_{\text{bool}}$
  - meaning function  $\gamma_{\text{bool}} : \text{AbsDom}_{\text{bool}} \rightarrow 2^{\text{States}}$
- Cartesian abstraction
  - *abstract domain*  $\text{AbsDom}_{\text{cartesian}}$
  - abstraction function  $\alpha_{\text{cartesian}} : \text{AbsDom}_{\text{bool}} \rightarrow \text{AbsDom}_{\text{cartesian}}$
  - meaning function  $\gamma_{\text{cartesian}} : \text{AbsDom}_{\text{cartesian}} \rightarrow \text{AbsDom}_{\text{bool}}$

# Defining $\text{post}_{b\cdot c}^\sharp$ : Outline

Question: How to define the abstract post operator s.t. it can be computed efficiently?

We combine *Boolean* and *Cartesian* abstraction:

- Boolean abstraction
  - *abstract domain*  $\text{AbsDom}_{\text{bool}}$
  - abstraction function  $\alpha_{\text{bool}} : 2^{\text{States}} \rightarrow \text{AbsDom}_{\text{bool}}$
  - meaning function  $\gamma_{\text{bool}} : \text{AbsDom}_{\text{bool}} \rightarrow 2^{\text{States}}$
- Cartesian abstraction
  - *abstract domain*  $\text{AbsDom}_{\text{cartesian}}$
  - abstraction function  $\alpha_{\text{cartesian}} : \text{AbsDom}_{\text{bool}} \rightarrow \text{AbsDom}_{\text{cartesian}}$
  - meaning function  $\gamma_{\text{cartesian}} : \text{AbsDom}_{\text{cartesian}} \rightarrow \text{AbsDom}_{\text{bool}}$

Then  $\text{post}_{b\cdot c}^\sharp = \alpha_{b\cdot c} \circ \text{post} \circ \gamma_{b\cdot c}$

where  $\alpha_{b\cdot c} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}}$ ,  $\gamma_{b\cdot c} = \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$

# Boolean Abstraction

We introduce a finite set of state predicates (“Boolean expressions”)

$$\mathcal{P} = \{p_1, \dots, p_n\}$$

where  $p_i$  denotes  $\{s \in \text{States} | s \models p_i\}$ .

# Boolean Abstraction

We introduce a finite set of state predicates (“Boolean expressions”)

$$\mathcal{P} = \{p_1, \dots, p_n\}$$

where  $p_i$  denotes  $\{s \in \text{States} | s \models p_i\}$ .

The associated equivalence relation

$$s = t \quad \text{iff} \quad \forall p \in \mathcal{P} : s \models p \iff t \models p$$

defines a partitioning of the state space into equivalence classes.

# Boolean Abstraction (2)

The equivalence classes

- form the abstract states
- can be represented by bitvectors of length  $n$

## Boolean Abstraction (2)

The equivalence classes

- form the abstract states
- can be represented by bitvectors of length  $n$

### Example

Let **States** =  $\{(x, y) \in \mathbb{Z} \times \mathbb{Z}\}$ ,  $\mathcal{P} = \{x > 1, y = 0\}$ .

Then  $\{(x, y) | x > 1, y = 0\} \cong \langle 1, 1 \rangle$ ,

$$\{(x, y) | x \leq 1, y = 0\} \cong \langle 0, 1 \rangle,$$
$$\{(x, y) | x > 1, y \neq 0\} \cong \langle 1, 0 \rangle,$$
$$\{(x, y) | x \leq 1, y \neq 0\} \cong \langle 0, 0 \rangle.$$

## Given

- $\mathcal{P} = \{p_1, \dots, p_n\}$
- *abstract domain*  
 $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- *concrete domain*  $2^{\text{States}}$

## Given

- $\mathcal{P} = \{p_1, \dots, p_n\}$
- *abstract domain*  
 $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- *concrete domain*  $2^{\text{States}}$

$$\begin{aligned}\alpha_{\text{bool}} : 2^{\text{States}} &\rightarrow \text{AbsDom}_{\text{bool}} \\ S &\mapsto \{\langle v_1, \dots, v_n \rangle \mid S \cap \{s \mid s \models v_1 \cdot p_1 \wedge \dots \wedge v_n \cdot p_n\} \neq \emptyset\}\end{aligned}$$

where  $0 \cdot p_i = \neg p_i$  and  $1 \cdot p_i = p_i$

## Given

- $\mathcal{P} = \{p_1, \dots, p_n\}$
- abstract domain  $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- concrete domain  $2^{\text{States}}$

## Example

$<0,0>$	$<0,1>$
 	 

$<1,0>$	$<1,1>$
  	 

$$\alpha_{\text{bool}} : 2^{\text{States}} \rightarrow \text{AbsDom}_{\text{bool}}$$

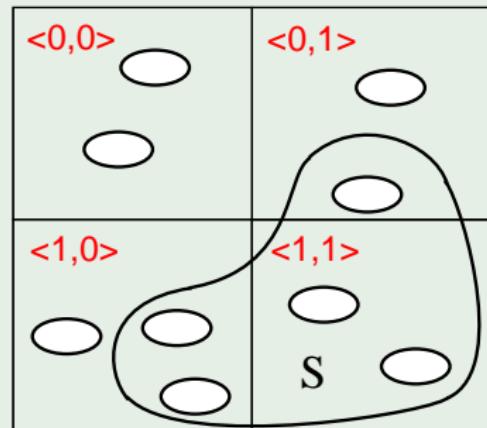
$$S \mapsto \{\langle v_1, \dots, v_n \rangle \mid S \cap \{s \mid s \models v_1 \cdot p_1 \wedge \dots \wedge v_n \cdot p_n\} \neq \emptyset\}$$

where  $0 \cdot p_i = \neg p_i$  and  $1 \cdot p_i = p_i$

## Given

- $\mathcal{P} = \{p_1, \dots, p_n\}$
- abstract domain  $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- concrete domain  $2^{\text{States}}$

## Example

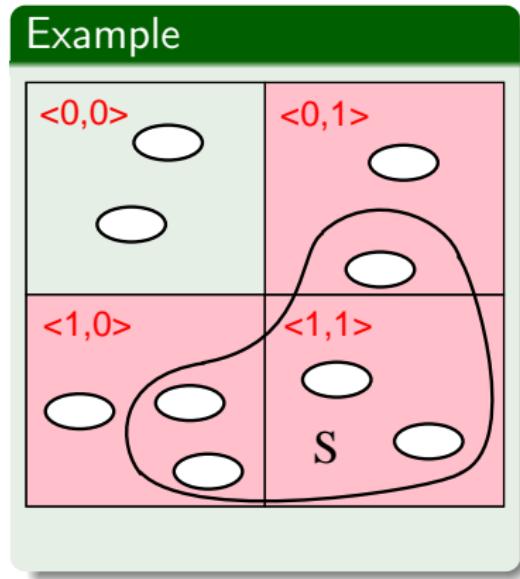


$$\begin{aligned}\alpha_{\text{bool}} : 2^{\text{States}} &\rightarrow \text{AbsDom}_{\text{bool}} \\ S &\mapsto \{\langle v_1, \dots, v_n \rangle \mid S \cap \{s \mid s \models v_1 \cdot p_1 \wedge \dots \wedge v_n \cdot p_n\} \neq \emptyset\}\end{aligned}$$

where  $0 \cdot p_i = \neg p_i$  and  $1 \cdot p_i = p_i$

## Given

- $\mathcal{P} = \{p_1, \dots, p_n\}$
- abstract domain  $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- concrete domain  $2^{\text{States}}$



$$\begin{aligned}\alpha_{\text{bool}} : 2^{\text{States}} &\rightarrow \text{AbsDom}_{\text{bool}} \\ S &\mapsto \{\langle v_1, \dots, v_n \rangle \mid S \cap \{s \mid s \models v_1 \cdot p_1 \wedge \dots \wedge v_n \cdot p_n\} \neq \emptyset\}\end{aligned}$$

where  $0 \cdot p_i = \neg p_i$  and  $1 \cdot p_i = p_i$

## Given

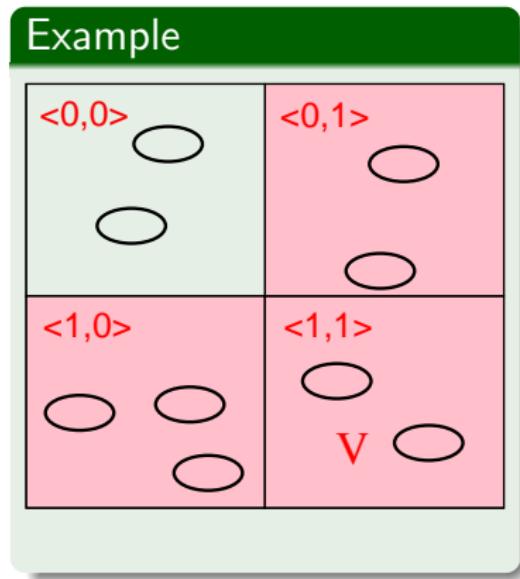
- $\mathcal{P} = \{p_1, \dots, p_n\}$
- *abstract domain*  
 $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- *concrete domain*  $2^{\text{States}}$

$$\begin{aligned}\gamma_{\text{bool}} : \text{AbsDom}_{\text{bool}} &\rightarrow 2^{\text{States}} \\ V &\mapsto \{s \mid \exists \langle v_1, \dots, v_n \rangle \in V : s \models v_1 \cdot p_1 \wedge \dots \wedge v_n \cdot p_n\}\end{aligned}$$

where  $0 \cdot p_i = \neg p_i$  and  $1 \cdot p_i = p_i$

## Given

- $\mathcal{P} = \{p_1, \dots, p_n\}$
- abstract domain  $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- concrete domain  $2^{\text{States}}$



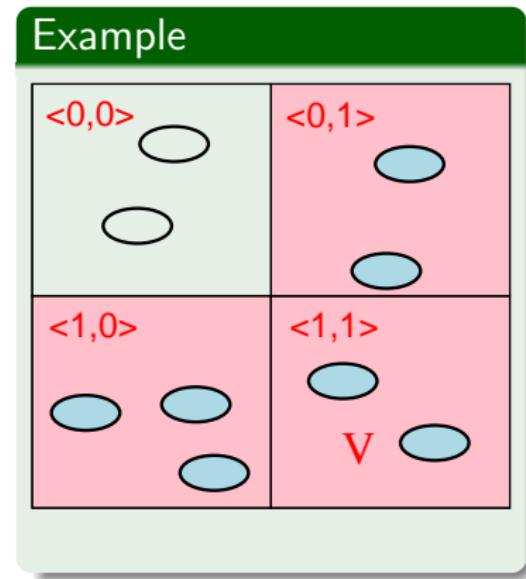
$$\gamma_{\text{bool}} : \text{AbsDom}_{\text{bool}} \rightarrow 2^{\text{States}}$$

$$V \mapsto \{s \mid \exists \langle v_1, \dots, v_n \rangle \in V : s \models v_1 \cdot p_1 \wedge \dots \wedge v_n \cdot p_n\}$$

where  $0 \cdot p_i = \neg p_i$  and  $1 \cdot p_i = p_i$

## Given

- $\mathcal{P} = \{p_1, \dots, p_n\}$
- abstract domain  $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- concrete domain  $2^{\text{States}}$



$$\gamma_{\text{bool}} : \text{AbsDom}_{\text{bool}} \rightarrow 2^{\text{States}}$$

$$V \mapsto \{s \mid \exists \langle v_1, \dots, v_n \rangle \in V : s \models v_1 \cdot p_1 \wedge \dots \wedge v_n \cdot p_n\}$$

where  $0 \cdot p_i = \neg p_i$  and  $1 \cdot p_i = p_i$

## Definition

$$\text{post}^\sharp_{\text{bool}} = \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}}$$

## Definition

$$\text{post}^\sharp_{\text{bool}} = \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}}$$

## Remarks:

- $\text{post}^\sharp_{\text{bool}}^*(\alpha_{\text{bool}}(\text{init}))$  – set of reachable states of the abstract transition system
- $\gamma_{\text{bool}}(\text{post}^\sharp_{\text{bool}}^*(\alpha_{\text{bool}}(\text{init})))$  – invariant of the concrete program

## $\alpha_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\alpha_{\text{cartesian}} &: 2^{D_1 \times \dots \times D_n} \rightarrow 2^{D_1} \times \dots \times 2^{D_n} \\ V &\mapsto \langle \Pi_1(V), \dots, \Pi_n(V) \rangle\end{aligned}$$

where  $\Pi_i(V) = \{v_i | \langle v_1, \dots, v_n \rangle \in V\}$

# $\alpha_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\alpha_{\text{cartesian}} &: 2^{D_1 \times \dots \times D_n} \rightarrow 2^{D_1} \times \dots \times 2^{D_n} \\ V &\mapsto \langle \Pi_1(V), \dots, \Pi_n(V) \rangle\end{aligned}$$

where  $\Pi_i(V) = \{v_i | \langle v_1, \dots, v_n \rangle \in V\}$

## Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$

## $\alpha_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\alpha_{\text{cartesian}} &: 2^{D_1 \times \dots \times D_n} \rightarrow 2^{D_1} \times \dots \times 2^{D_n} \\ V &\mapsto \langle \Pi_1(V), \dots, \Pi_n(V) \rangle\end{aligned}$$

where  $\Pi_i(V) = \{v_i | \langle v_1, \dots, v_n \rangle \in V\}$

### Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$   
 $\Pi_1(V) = \{0\}$   
 $\Pi_2(V) = \{1\}$   
 $\Pi_3(V) = \Pi_4(V) = \{0, 1\}$

# $\alpha_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\alpha_{\text{cartesian}} &: 2^{D_1 \times \dots \times D_n} \rightarrow 2^{D_1} \times \dots \times 2^{D_n} \\ V &\mapsto \langle \Pi_1(V), \dots, \Pi_n(V) \rangle\end{aligned}$$

where  $\Pi_i(V) = \{v_i \mid \langle v_1, \dots, v_n \rangle \in V\}$

## Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$   
 $\Pi_1(V) = \{0\}$   
 $\Pi_2(V) = \{1\}$   
 $\Pi_3(V) = \Pi_4(V) = \{0, 1\}$
- $\alpha_{\text{cartesian}}(V) = \langle \{0\}, \{1\}, \{0, 1\}, \{0, 1\} \rangle$

# $\gamma_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\gamma_{\text{cartesian}} &: 2^{D_1} \times \dots \times 2^{D_n} \rightarrow 2^{D_1 \times \dots \times D_n} \\ \langle M_1, \dots, M_n \rangle &\mapsto M_1 \times \dots \times M_n\end{aligned}$$

## $\gamma_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\gamma_{\text{cartesian}} &: 2^{D_1} \times \dots \times 2^{D_n} \rightarrow 2^{D_1 \times \dots \times D_n} \\ \langle M_1, \dots, M_n \rangle &\mapsto M_1 \times \dots \times M_n\end{aligned}$$

### Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$
- $\alpha_{\text{cartesian}}(V) = \langle \{0\}, \{1\}, \{0, 1\}, \{0, 1\} \rangle$

## $\gamma_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\gamma_{\text{cartesian}} &: 2^{D_1} \times \dots \times 2^{D_n} \rightarrow 2^{D_1 \times \dots \times D_n} \\ \langle M_1, \dots, M_n \rangle &\mapsto M_1 \times \dots \times M_n\end{aligned}$$

### Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$
- $\alpha_{\text{cartesian}}(V) = \langle \{0\}, \{1\}, \{0, 1\}, \{0, 1\} \rangle$
- $\gamma_{\text{cartesian}}(\alpha_{\text{cartesian}}(V)) =$   
 $\{\langle 0, 1, 0, 0 \rangle, \langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle, \langle 0, 1, 1, 1 \rangle\}$

## $\gamma_{\text{cartesian}}$ : Abstract Definition

$$\begin{aligned}\gamma_{\text{cartesian}} &: 2^{D_1} \times \dots \times 2^{D_n} \rightarrow 2^{D_1 \times \dots \times D_n} \\ \langle M_1, \dots, M_n \rangle &\mapsto M_1 \times \dots \times M_n\end{aligned}$$

### Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$
- $\alpha_{\text{cartesian}}(V) = \{\{0\}, \{1\}, \{0, 1\}, \{0, 1\}\}$
- $\gamma_{\text{cartesian}}(\alpha_{\text{cartesian}}(V)) =$   
 $\{\langle 0, 1, 0, 0 \rangle, \langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle, \langle 0, 1, 1, 1 \rangle\}$

⇒ loss of precision

# Trivectors

- We can write

- 0 for  $\{0\}$
- 1 for  $\{1\}$
- \* for  $\{0, 1\}$

# Trivectors

- We can write
  - 0 for  $\{0\}$
  - 1 for  $\{1\}$
  - \* for  $\{0, 1\}$
- $v \in \{0, 1, *\}^n$  is called a *trivector*

# Trivectors

- We can write
  - 0 for  $\{0\}$
  - 1 for  $\{1\}$
  - \* for  $\{0, 1\}$
- $v \in \{0, 1, *\}^n$  is called a *trivector*
- We define a partial order  $<$  such that

$$0 < 0, \quad 1 < 1, \quad 0 < *, \quad 1 < *$$

# Trivectors

- We can write
  - 0 for  $\{0\}$
  - 1 for  $\{1\}$
  - \* for  $\{0, 1\}$
- $v \in \{0, 1, *\}^n$  is called a *trivector*
- We define a partial order  $<$  such that

$$0 < 0, \quad 1 < 1, \quad 0 < *, \quad 1 < *$$

- and its pointwise extension

$$\langle v_1, \dots, v_n \rangle < \langle v'_1, \dots, v'_n \rangle \text{ iff } v_1 < v'_1, \dots, v_n < v'_n$$

Given

- $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- $\text{AbsDom}_{\text{cartesian}} = \{0, 1, *\}^n$

$$\begin{aligned}\alpha_{\text{cartesian}} &: \text{AbsDom}_{\text{bool}} \rightarrow \text{AbsDom}_{\text{cartesian}} \\ V &\mapsto \langle v_1, \dots, v_n \rangle\end{aligned}$$

where

$$v_i = \begin{cases} 0 & \text{if } \Pi_i(V) = \{0\} \\ 1 & \text{if } \Pi_i(V) = \{1\} \\ * & \text{if } \Pi_i(V) = \{0, 1\} \end{cases}$$

Given

- $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- $\text{AbsDom}_{\text{cartesian}} = \{0, 1, *\}^n$

$$\begin{aligned}\alpha_{\text{cartesian}} : \quad & \text{AbsDom}_{\text{bool}} \rightarrow \text{AbsDom}_{\text{cartesian}} \\ V \mapsto & \langle v_1, \dots, v_n \rangle\end{aligned}$$

where

$$v_i = \begin{cases} 0 & \text{if } \Pi_i(V) = \{0\} \\ 1 & \text{if } \Pi_i(V) = \{1\} \\ * & \text{if } \Pi_i(V) = \{0, 1\} \end{cases}$$

Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$   
 $\Pi_1(V) = \{0\}, \Pi_2(V) = \{1\}, \Pi_3(V) = \Pi_4(V) = \{0, 1\}$

Given

- $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- $\text{AbsDom}_{\text{cartesian}} = \{0, 1, *\}^n$

$$\begin{aligned}\alpha_{\text{cartesian}} : \text{AbsDom}_{\text{bool}} &\rightarrow \text{AbsDom}_{\text{cartesian}} \\ V &\mapsto \langle v_1, \dots, v_n \rangle\end{aligned}$$

where

$$v_i = \begin{cases} 0 & \text{if } \Pi_i(V) = \{0\} \\ 1 & \text{if } \Pi_i(V) = \{1\} \\ * & \text{if } \Pi_i(V) = \{0, 1\} \end{cases}$$

Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$   
 $\Pi_1(V) = \{0\}, \Pi_2(V) = \{1\}, \Pi_3(V) = \Pi_4(V) = \{0, 1\}$
- $\alpha_{\text{cartesian}}(V) = \langle 0, 1, *, * \rangle$

## Given

- $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- $\text{AbsDom}_{\text{cartesian}} = \{0, 1, *\}^n$

$$\begin{aligned}\gamma_{\text{cartesian}} &: \text{AbsDom}_{\text{bool}} \rightarrow \text{AbsDom}_{\text{cartesian}} \\ v &\mapsto \{v' \mid v' < v\}\end{aligned}$$

## Given

- $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- $\text{AbsDom}_{\text{cartesian}} = \{0, 1, *\}^n$

$$\begin{aligned}\gamma_{\text{cartesian}} &: \text{AbsDom}_{\text{bool}} \rightarrow \text{AbsDom}_{\text{cartesian}} \\ v &\mapsto \{v' \mid v' < v\}\end{aligned}$$

## Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$
- $\alpha_{\text{cartesian}}(V) = \langle 0, 1, *, * \rangle$

## Given

- $\text{AbsDom}_{\text{bool}} = 2^{\mathbb{B}^n}$
- $\text{AbsDom}_{\text{cartesian}} = \{0, 1, *\}^n$

$$\begin{aligned}\gamma_{\text{cartesian}} : \text{AbsDom}_{\text{bool}} &\rightarrow \text{AbsDom}_{\text{cartesian}} \\ v &\mapsto \{v' \mid v' < v\}\end{aligned}$$

## Example

- $V = \{\langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle\}$
- $\alpha_{\text{cartesian}}(V) = \langle 0, 1, *, * \rangle$
- $\begin{aligned}\gamma_{\text{cartesian}}(\alpha_{\text{cartesian}}(V)) \\ &= \{v \mid v < \langle 0, 1, *, * \rangle\} \\ &= \{\langle 0, 1, 0, 0 \rangle, \langle 0, 1, 0, 1 \rangle, \langle 0, 1, 1, 0 \rangle, \langle 0, 1, 1, 1 \rangle\}\end{aligned}$

# Combining the Abstractions: $\text{post}_{b \cdot c}^\sharp$

## Given

- $\alpha_{b \cdot c} : 2^{\text{States}} \rightarrow \text{AbsDom}_{\text{cartesian}}$   
 $\alpha_{b \cdot c} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}}$
- $\gamma_{b \cdot c} : \text{AbsDom}_{\text{cartesian}} \rightarrow 2^{\text{States}}$   
 $\alpha_{b \cdot c} = \alpha_{\text{bool}} \circ \alpha_{\text{cartesian}}$
- $\text{post}_{b \cdot c}^\sharp = \alpha_{b \cdot c} \circ \text{post} \circ \gamma_{b \cdot c}$

# Combining the Abstractions: $\text{post}_{b \cdot c}^\#$

## Given

- $\alpha_{b \cdot c} : 2^{\text{States}} \rightarrow \text{AbsDom}_{\text{cartesian}}$

$$\alpha_{b \cdot c} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}}$$

- $\gamma_{b \cdot c} : \text{AbsDom}_{\text{cartesian}} \rightarrow 2^{\text{States}}$

$$\alpha_{b \cdot c} = \alpha_{\text{bool}} \circ \alpha_{\text{cartesian}}$$

- $\text{post}_{b \cdot c}^\# = \alpha_{b \cdot c} \circ \text{post} \circ \gamma_{b \cdot c}$

- $\text{Inv}_1 = \gamma_{b \cdot c}(\text{post}_{b \cdot c}^\#{}^*(\alpha_{b \cdot c}(\text{init})))$

- is a safe invariant of the program

- is represented abstractly by one trivector

## Definition

A *Boolean program* is a program over ‘Boolean’ variables, i.e. variables with domain  $\{0, 1, *\}$ .

## Definition

A *Boolean program* is a program over ‘Boolean’ variables, i.e. variables with domain  $\{0, 1, *\}$ .

c2bp:

- A tool developed in the context of the SLAM project.

## Definition

A *Boolean program* is a program over ‘Boolean’ variables, i.e. variables with domain  $\{0, 1, *\}$ .

c2bp:

- A tool developed in the context of the SLAM project.
- Given a C program (i.e. an input transition system) and a set of  $n$  predicates  $\mathcal{P}$ , c2bp
  - computes a Boolean program over  $n$  variables  $v_1, \dots, v_n$  corresponding to  $\mathcal{P}$

# c2bp: Example

## C Program

```
int x, y, z, w;
```

```
void foo() {
    do {
        z = 0;
        x = y;
        if(w) {
            x++;
            z = 1;
        }
    } while(x!=y)
}
```

# c2bp: Example

## C Program

```
int x, y, z, w;  
  
void foo() {  
    do {  
        z = 0;  
        x = y;  
        if(w) {  
            x++;  
            z = 1;  
        }  
    } while(x!=y)  
}
```

## Boolean Program

```
decl b1, b2;  
/* b1 stands for (z=0),  
   b2 stands for (x=y) */  
void foo() begin  
    do  
        b1 := 1;  
        b2 := 1;  
        if(*) begin  
            b2 := H(0,b2); /* b2 ? 0 : * */  
            b1 := 0;  
        end  
        while(!b1)  
    end
```

## Definition

A *Boolean program* is a program over ‘Boolean’ variables, i.e. variables with domain  $\{0, 1, *\}$ .

c2bp:

- A tool developed in the context of the SLAM project.
- Given a C program (i.e. an input transition system) and a set of  $n$  predicates  $\mathcal{P}$ , c2bp
  - computes a Boolean program over  $n$  variables  $v_1, \dots, v_n$  corresponding to  $\mathcal{P}$

## Definition

A *Boolean program* is a program over ‘Boolean’ variables, i.e. variables with domain  $\{0, 1, *\}$ .

c2bp:

- A tool developed in the context of the SLAM project.
- Given a C program (i.e. an input transition system) and a set of  $n$  predicates  $\mathcal{P}$ , c2bp
  - computes a Boolean program over  $n$  variables  $v_1, \dots, v_n$  corresponding to  $\mathcal{P}$
  - translates every statement of the C program to an assignment

$$\langle v_1, \dots, v_n \rangle := \langle e_1, \dots, e_n \rangle$$

where  $e_1, \dots, e_n$  are expressions over  $v_1, \dots, v_n$

## c2bp and its Post Operator

Since  $\text{AbsDom}_{\text{cartesian}} = \{0, 1, *\}^n$ , a Boolean program with assignments

$$\langle v_1, \dots, v_n \rangle := \langle e_1, \dots, e_n \rangle$$

represents an operator  $\text{post}_{\text{c2bp}}^\#$  over trivectors such that

$$\text{post}_{\text{c2bp}}^\#(\langle v_1, \dots, v_n \rangle) = \langle v'_1, \dots, v'_n \rangle \text{ where } v'_i = e_i$$

# Computing post $_{c2bp}^{\sharp}$

## Definition

$$\widetilde{\text{pre}}(\{s | s \models v_i\})$$

# Computing post $_{c2bp}^{\sharp}$

## Definition

$$\{s\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$$

# Computing post $_{\text{c2bp}}^{\sharp}$

## Definition

$$e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$$

# Computing post $_{c2bp}^{\sharp}$

## Definition

- $e_i(1) = \nu e \in BE(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{pre}(\{s | s \models v_i\})$   
where  $e \leq e' \iff \{s | s \models e\} \subseteq \{s | s \models e'\}$

# Computing post $_{\text{c2bp}}^{\sharp}$

## Definition

- $e_i(1) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$   
where  $e \leq e' \iff \{s | s \models e\} \subseteq \{s | s \models e'\}$
- $e_i(0) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models \neg v_i\})$

# Computing post $_{c2bp}^{\sharp}$

## Definition

- $e_i(1) = \nu e \in BE(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{pre}(\{s | s \models v_i\})$   
where  $e \leq e' \iff \{s | s \models e\} \subseteq \{s | s \models e'\}$
- $e_i(0) = \nu e \in BE(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{pre}(\{s | s \models \neg v_i\})$

We write

$$H(e, e') = \begin{cases} 1 & \text{if } \langle v_1, \dots, v_n \rangle \models e \\ 0 & \text{if } \langle v_1, \dots, v_n \rangle \models e' \\ * & \text{if neither} \end{cases}$$

# Computing post $_{c2bp}^{\sharp}$

## Definition

- $e_i(1) = \nu e \in BE(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$   
where  $e \leq e' \iff \{s | s \models e\} \subseteq \{s | s \models e'\}$
- $e_i(0) = \nu e \in BE(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models \neg v_i\})$

We write

$$H(e, e') = \begin{cases} 1 & \text{if } \langle v_1, \dots, v_n \rangle \models e \\ 0 & \text{if } \langle v_1, \dots, v_n \rangle \models e' \\ * & \text{if neither} \end{cases}$$

Consider  $e_i = H(e_i(1), e_i(0))$ :

- $e_i = 1 \iff \langle v_1, \dots, v_n \rangle \models e_i(1)$

# Computing post $_{\text{c2bp}}^{\sharp}$

## Definition

- $e_i(1) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$   
where  $e \leq e' \iff \{s | s \models e\} \subseteq \{s | s \models e'\}$
- $e_i(0) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models \neg v_i\})$

We write

$$H(e, e') = \begin{cases} 1 & \text{if } \langle v_1, \dots, v_n \rangle \models e \\ 0 & \text{if } \langle v_1, \dots, v_n \rangle \models e' \\ * & \text{if neither} \end{cases}$$

Consider  $e_i = H(e_i(1), e_i(0))$ :

- $e_i = 1 \iff \langle v_1, \dots, v_n \rangle \models e_i(1)$   
 $\iff \gamma_{\text{b.c}}(\langle v_1, \dots, v_n \rangle) \subseteq \{s | s \models e_i(1)\}$

# Computing post $_{\text{c2bp}}^{\sharp}$

## Definition

- $e_i(1) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$   
where  $e \leq e' \iff \{s | s \models e\} \subseteq \{s | s \models e'\}$
- $e_i(0) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models \neg v_i\})$

We write

$$H(e, e') = \begin{cases} 1 & \text{if } \langle v_1, \dots, v_n \rangle \models e \\ 0 & \text{if } \langle v_1, \dots, v_n \rangle \models e' \\ * & \text{if neither} \end{cases}$$

Consider  $e_i = H(e_i(1), e_i(0))$ :

- $e_i = 1 \iff \langle v_1, \dots, v_n \rangle \models e_i(1)$   
 $\iff \gamma_{\text{b.c}}(\langle v_1, \dots, v_n \rangle) \subseteq \{s | s \models e_i(1)\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$

# Computing post $_{\text{c2bp}}^{\sharp}$

## Definition

- $e_i(1) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$   
where  $e \leq e' \iff \{s | s \models e\} \subseteq \{s | s \models e'\}$
- $e_i(0) = \nu e \in \text{BE}(v_1, \dots, v_n). \{s | s \models e\} \subseteq \widetilde{\text{pre}}(\{s | s \models \neg v_i\})$

We write

$$H(e, e') = \begin{cases} 1 & \text{if } \langle v_1, \dots, v_n \rangle \models e \\ 0 & \text{if } \langle v_1, \dots, v_n \rangle \models e' \\ * & \text{if neither} \end{cases}$$

Consider  $e_i = H(e_i(1), e_i(0))$ :

- $e_i = 1 \iff \langle v_1, \dots, v_n \rangle \models e_i(1)$   
 $\iff \gamma_{\text{b.c}}(\langle v_1, \dots, v_n \rangle) \subseteq \{s | s \models e_i(1)\} \subseteq \widetilde{\text{pre}}(\{s | s \models v_i\})$
- $e_i = 0 \iff \langle v_1, \dots, v_n \rangle \models e_i(0)$   
 $\iff \gamma_{\text{b.c}}(\langle v_1, \dots, v_n \rangle) \subseteq \{s | s \models e_i(0)\} \subseteq \widetilde{\text{pre}}(\{s | s \models \neg v_i\})$

$\text{post}_{\text{c2bp}}^\#$  and  $\text{post}_{\text{b}\cdot\text{c}}^\#$

Then  $v'_i = H(e_i(1), e_i(0))$ , i.e.

$$\text{post}_{\text{c2bp}}^\#(\langle v_1, \dots, v_n \rangle) = \langle H(e_1(1), e_1(0)), \dots, H(e_n(1), e_n(0)) \rangle$$

$\text{post}_{\text{c2bp}}^\#$  and  $\text{post}_{\text{b.c}}^\#$

Then  $v'_i = H(e_i(1), e_i(0))$ , i.e.

$$\text{post}_{\text{c2bp}}^\#(\langle v_1, \dots, v_n \rangle) = \langle H(e_1(1), e_1(0)), \dots, H(e_n(1), e_n(0)) \rangle$$

## Proposition 1

$$\text{post}_{\text{c2bp}}^\# = \text{post}_{\text{b.c}}^\#$$

# Complexity and Performance

- We need  $\mathbf{F}(\mathbf{S}) = \nu \mathbf{e} \in \mathbf{BE}(v_1, \dots, v_n). \{s \mid s \models e\} \subseteq \mathbf{S}$   
for  $2n$  sets  $S$  (i.e. for  $\widetilde{\text{pre}}(\{s \mid s \models v_i\})$ ,  $\widetilde{\text{pre}}(\{s \mid s \models \neg v_i\})$ )

# Complexity and Performance

- We need  $\mathbf{F}(\mathbf{S}) = \nu \mathbf{e} \in \mathbf{BE}(v_1, \dots, v_n). \{s \mid s \models e\} \subseteq \mathbf{S}$  for  $2n$  sets  $S$  (i.e. for  $\widetilde{\text{pre}}(\{s \mid s \models v_i\})$ ,  $\widetilde{\text{pre}}(\{s \mid s \models \neg v_i\})$ )
- To compute each  $F(S)$ , we need

$$\mathcal{C} = \bigwedge_{i \in I} l_i$$

where  $l_i \in \{p_i, \neg p_i\}$ ,  $\{s \mid s \models \mathcal{C}\} \subseteq S$  and  $I$  minimal  
( $\mathcal{C}$  minimal implicant of  $S$ ).

# Complexity and Performance

- We need  $\mathbf{F}(\mathbf{S}) = \nu \mathbf{e} \in \mathbf{BE}(v_1, \dots, v_n). \{s \mid s \models e\} \subseteq \mathbf{S}$  for  $2n$  sets  $S$  (i.e. for  $\widetilde{\text{pre}}(\{s \mid s \models v_i\})$ ,  $\widetilde{\text{pre}}(\{s \mid s \models \neg v_i\})$ )
- To compute each  $F(S)$ , we need

$$\mathcal{C} = \bigwedge_{i \in I} l_i$$

where  $l_i \in \{p_i, \neg p_i\}$ ,  $\{s \mid s \models \mathcal{C}\} \subseteq S$  and  $I$  minimal  
( $\mathcal{C}$  minimal implicant of  $S$ ).

- Complexity:  $O(2^n) \cdot 2 \cdot n$

# Complexity and Performance

- We need  $\mathbf{F}(\mathbf{S}) = \nu \mathbf{e} \in \mathbf{BE}(\mathbf{v}_1, \dots, \mathbf{v}_n). \{s \mid s \models e\} \subseteq \mathbf{S}$  for  $2n$  sets  $S$  (i.e. for  $\widetilde{\text{pre}}(\{s \mid s \models v_i\})$ ,  $\widetilde{\text{pre}}(\{s \mid s \models \neg v_i\})$ )
- To compute each  $F(S)$ , we need

$$\mathcal{C} = \bigwedge_{i \in I} l_i$$

where  $l_i \in \{p_i, \neg p_i\}$ ,  $\{s \mid s \models \mathcal{C}\} \subseteq S$  and  $I$  minimal ( $\mathcal{C}$  minimal implicant of  $S$ ).

- Complexity:  $O(2^n) \cdot 2 \cdot n$
- Optimizations:
  - $\mathcal{C}$  implicant  $\implies$  skip  $\mathcal{C} \wedge p_j$  for all  $j$

# Complexity and Performance

- We need  $\mathbf{F}(\mathbf{S}) = \nu \mathbf{e} \in \mathbf{BE}(v_1, \dots, v_n). \{s \mid s \models e\} \subseteq \mathbf{S}$  for  $2n$  sets  $S$  (i.e. for  $\widetilde{\text{pre}}(\{s \mid s \models v_i\})$ ,  $\widetilde{\text{pre}}(\{s \mid s \models \neg v_i\})$ )
- To compute each  $F(S)$ , we need

$$\mathcal{C} = \bigwedge_{i \in I} l_i$$

where  $l_i \in \{p_i, \neg p_i\}$ ,  $\{s \mid s \models \mathcal{C}\} \subseteq S$  and  $I$  minimal ( $\mathcal{C}$  minimal implicant of  $S$ ).

- Complexity:  $O(2^n) \cdot 2 \cdot n$
- Optimizations:
  - $\mathcal{C}$  implicant  $\implies$  skip  $\mathcal{C} \wedge p_j$  for all  $j$
  - $\{s \mid s \models \mathcal{C}\} \cap S = \emptyset \implies$  skip  $\mathcal{C} \wedge p_j$  for all  $j$

# Complexity and Performance

- We need  $\mathbf{F}(\mathbf{S}) = \nu \mathbf{e} \in \mathbf{BE}(\mathbf{v}_1, \dots, \mathbf{v}_n). \{s \mid s \models \mathbf{e}\} \subseteq \mathbf{S}$  for  $2n$  sets  $S$  (i.e. for  $\widetilde{\text{pre}}(\{s \mid s \models v_i\})$ ,  $\widetilde{\text{pre}}(\{s \mid s \models \neg v_i\})$ )
- To compute each  $F(S)$ , we need

$$\mathcal{C} = \bigwedge_{i \in I} l_i$$

where  $l_i \in \{p_i, \neg p_i\}$ ,  $\{s \mid s \models \mathcal{C}\} \subseteq S$  and  $I$  minimal ( $\mathcal{C}$  minimal implicant of  $S$ ).

- Complexity:  $O(2^n) \cdot 2 \cdot n$
- Optimizations:
  - $\mathcal{C}$  implicant  $\implies$  skip  $\mathcal{C} \wedge p_j$  for all  $j$
  - $\{s \mid s \models \mathcal{C}\} \cap S = \emptyset \implies$  skip  $\mathcal{C} \wedge p_j$  for all  $j$
  - restricting  $|I|$ , e.g.  $|I| \leq 3$

# Complexity and Performance

- We need  $\mathbf{F}(\mathbf{S}) = \nu \mathbf{e} \in \mathbf{BE}(v_1, \dots, v_n). \{s \mid s \models e\} \subseteq \mathbf{S}$  for  $2n$  sets  $S$  (i.e. for  $\widetilde{\text{pre}}(\{s \mid s \models v_i\})$ ,  $\widetilde{\text{pre}}(\{s \mid s \models \neg v_i\})$ )
- To compute each  $F(S)$ , we need

$$\mathcal{C} = \bigwedge_{i \in I} l_i$$

where  $l_i \in \{p_i, \neg p_i\}$ ,  $\{s \mid s \models \mathcal{C}\} \subseteq S$  and  $I$  minimal ( $\mathcal{C}$  minimal implicant of  $S$ ).

- Complexity:  $O(2^n) \cdot 2 \cdot n$
- Optimizations:
  - $\mathcal{C}$  implicant  $\implies$  skip  $\mathcal{C} \wedge p_j$  for all  $j$
  - $\{s \mid s \models \mathcal{C}\} \cap S = \emptyset \implies$  skip  $\mathcal{C} \wedge p_j$  for all  $j$
  - restricting  $|I|$ , e.g.  $|I| \leq 3$
- Performance: Processing C programs with several hundred lines,  $\approx 35$  predicates in a few minutes.

# Loss of Precision under Cartesian Abstraction

## Definition

Given a meaning  $\gamma$  and an operator  $F$ ,  
 $F$  does not lose precision under abstraction to  $F^\sharp$  on  $a$  if

$$\gamma \circ F^\sharp(a) = F \circ \gamma(a)$$

# Loss of Precision under Cartesian Abstraction

## Definition

Given a meaning  $\gamma$  and an operator  $F$ ,  
 $F$  does not lose precision under abstraction to  $F^\sharp$  on  $a$  if

$$\gamma \circ F^\sharp(a) = F \circ \gamma(a)$$

## Proposition 2

If  $\text{post}_{\text{bool}}^\sharp$  is deterministic (i.e. maps singleton sets to singleton sets), then the Cartesian abstraction does not lose precision on  $\langle v_1, \dots, v_n \rangle$  such that  $v_i \neq *$  for all  $i$ .

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{b.c}^\sharp(\langle 0, 0, 0 \rangle) =$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{b,c}^{\sharp}(\langle 0, 0, 0 \rangle) =$   
 $\text{post}_{b,c}^{\sharp} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{b.c}^{\sharp}(\langle 0, 0, 0 \rangle) =$   
 $\text{post}_{b.c}^{\sharp} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :
  - $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{b.c}^\sharp(\langle 0, 0, 0 \rangle) =$   
 $\text{post}_{b.c}^\sharp = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}:$ 
  - $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{b.c}^\sharp(\langle 0, 0, 0 \rangle) =$   
 $\text{post}_{b.c}^\sharp = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}:$ 
  - $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{b.c}^{\sharp}(\langle 0, 0, 0 \rangle) =$   
 $\text{post}_{b.c}^{\sharp} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}:$ 
  - $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$
  - $\alpha_{\text{bool}}(\{\langle x, y \rangle | x = y, y \neq 5\}) = \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{\text{b.c}}^{\sharp}(\langle 0, 0, 0 \rangle) =$   
 $\text{post}_{\text{b.c}}^{\sharp} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}:$ 
  - $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$
  - $\alpha_{\text{bool}}(\{\langle x, y \rangle | x = y, y \neq 5\}) = \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$
- $\alpha_{\text{cartesian}}(\{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}) = \langle *, *, 0 \rangle$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
- $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
- $\text{post}_{b.c}^{\sharp}(\langle 0, 0, 0 \rangle) = \langle *, *, 0 \rangle$
- $\text{post}_{b.c}^{\sharp} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :
  - $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$
  - $\alpha_{\text{bool}}(\{\langle x, y \rangle | x = y, y \neq 5\}) = \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$
- $\alpha_{\text{cartesian}}(\{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}) = \langle *, *, 0 \rangle$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
  - $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
  - $\text{post}_{\text{b.c}}^{\sharp}(\langle 0, 0, 0 \rangle) = \langle *, *, 0 \rangle$
- $\text{post}_{\text{b.c}}^{\sharp} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :
- $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$
  - $\alpha_{\text{bool}}(\{\langle x, y \rangle | x = y, y \neq 5\}) = \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$   
 $(= \text{post}_{\text{bool}}^{\sharp}(\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)))$
  - $\alpha_{\text{cartesian}}(\{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}) = \langle *, *, 0 \rangle$

# Loss of Precision: Example 1

- ‘C program’:  $x = y$
  - $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
  - $\text{post}_{b.c}^\#(\langle 0, 0, 0 \rangle) = \langle *, *, 0 \rangle$
- $\text{post}_{b.c}^\# = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :
- $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$
  - $\alpha_{\text{bool}}(\{\langle x, y \rangle | x = y, y \neq 5\}) = \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$   
 $(= \text{post}_{\text{bool}}^\#(\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)))$
  - $\alpha_{\text{cartesian}}(\{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}) = \langle *, *, 0 \rangle$

Therefore:

$$\begin{aligned}\gamma_{\text{cartesian}} \circ \text{post}_{b.c}^\#(\langle 0, 0, 0 \rangle) &= \{\langle 0, 0, 0 \rangle, \langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 1, 0 \rangle\} \\ &\supsetneq \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\} \\ &= \text{post}_{\text{bool}}^\# \circ \gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)\end{aligned}$$

# Loss of Precision: Example 1

- 'C program':  $x = y$
  - $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
  - $\text{post}_{b.c}^\#(\langle 0, 0, 0 \rangle) = \langle *, *, 0 \rangle$
- $\text{post}_{b.c}^\# = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :
- $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$
  - $\alpha_{\text{bool}}(\{\langle x, y \rangle | x = y, y \neq 5\}) = \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$   
 $(= \text{post}_{\text{bool}}^\#(\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)))$
  - $\alpha_{\text{cartesian}}(\{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}) = \langle *, *, 0 \rangle$

Therefore:

$$\begin{aligned}\gamma_{\text{cartesian}} \circ \text{post}_{b.c}^\#(\langle 0, 0, 0 \rangle) &= \{\langle 0, 0, 0 \rangle, \langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 1, 0 \rangle\} \\ &\supsetneq \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\} \\ &= \text{post}_{\text{bool}}^\# \circ \gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)\end{aligned}$$

Reason:  $\text{post}_{\text{bool}}^\#$  not deterministic

# Loss of Precision: Example 1

- 'C program':  $x = y$
  - $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
  - $\text{post}_{b.c}^{\#}(\langle 0, 0, 0 \rangle) = \langle *, *, 0 \rangle$
- $\text{post}_{b.c}^{\#} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :
- $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{ \langle 0, 0, 0 \rangle \}$
  - $\gamma_{\text{bool}}(\{ \langle 0, 0, 0 \rangle \}) = \{ \langle x, y \rangle | x = 5, y \neq 5 \}$
  - $\text{post}(\{ \langle x, y \rangle | x = 5, y \neq 5 \}) = \{ \langle x, y \rangle | x = y, y \neq 5 \}$
  - $\alpha_{\text{bool}}(\{ \langle x, y \rangle | x = y, y \neq 5 \}) = \{ \langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle \}$   
 $(= \text{post}_{\text{bool}}^{\#}(\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)))$
  - $\alpha_{\text{cartesian}}(\{ \langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle \}) = \langle *, *, 0 \rangle$

Therefore:

$$\begin{aligned}\gamma_{\text{cartesian}} \circ \text{post}_{b.c}^{\#}(\langle 0, 0, 0 \rangle) &= \{ \langle 0, 0, 0 \rangle, \langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 1, 0 \rangle \} \\ &\supsetneq \{ \langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle \} \\ &= \text{post}_{\text{bool}}^{\#} \circ \gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)\end{aligned}$$

Reason:  $\text{post}_{\text{bool}}^{\#}$  not deterministic

# Loss of Precision: Example 1

- 'C program':  $x = y$
  - $\mathcal{P} = \{x > 5, x < 5, y = 5\}$  (Note:  $\neg(x > 5) \wedge \neg(x < 5) \iff x = 5$ )
  - $\text{post}_{b.c}^{\#}(\langle 0, 0, 0 \rangle) = \langle *, *, 0 \rangle$
- $\text{post}_{b.c}^{\#} = \alpha_{\text{cartesian}} \circ \alpha_{\text{bool}} \circ \text{post} \circ \gamma_{\text{bool}} \circ \gamma_{\text{cartesian}}$ :
- $\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle) = \{\langle 0, 0, 0 \rangle\}$
  - $\gamma_{\text{bool}}(\{\langle 0, 0, 0 \rangle\}) = \{\langle x, y \rangle | x = 5, y \neq 5\}$
  - $\text{post}(\{\langle x, y \rangle | x = 5, y \neq 5\}) = \{\langle x, y \rangle | x = y, y \neq 5\}$
  - $\alpha_{\text{bool}}(\{\langle x, y \rangle | x = y, y \neq 5\}) = \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}$   
 $(= \text{post}_{\text{bool}}^{\#}(\gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)))$
  - $\alpha_{\text{cartesian}}(\{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\}) = \langle *, *, 0 \rangle$

Therefore:

$$\begin{aligned}\gamma_{\text{cartesian}} \circ \text{post}_{b.c}^{\#}(\langle 0, 0, 0 \rangle) &= \{\langle 0, 0, 0 \rangle, \langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle, \langle 1, 1, 0 \rangle\} \\ &\supsetneq \{\langle 1, 0, 0 \rangle, \langle 0, 1, 0 \rangle\} \\ &= \text{post}_{\text{bool}}^{\#} \circ \gamma_{\text{cartesian}}(\langle 0, 0, 0 \rangle)\end{aligned}$$

Reason:  $\text{post}_{\text{bool}}^{\#}$  not deterministic

# Loss of Precision: Example 2

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$

# Loss of Precision: Example 2

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- Then  $\text{post}_{\text{bool}}^{\sharp} \circ \gamma_{\text{cartesian}}(\langle *, * \rangle) = \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}$

# Loss of Precision: Example 2

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- Then  $\text{post}_{\text{bool}}^\# \circ \gamma_{\text{cartesian}}(\langle *, * \rangle) = \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}$   
whereas  $\text{post}_{\text{b.c.}}^\#(\langle *, * \rangle) = \langle *, * \rangle$ .

# Loss of Precision: Example 2

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- Then  $\text{post}_{\text{bool}}^{\sharp} \circ \gamma_{\text{cartesian}}(\langle *, * \rangle) = \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}$   
whereas  $\text{post}_{\text{b.c.}}^{\sharp}(\langle *, * \rangle) = \langle *, * \rangle$ .

⇒ Lost of precision

# Loss of Precision: Example 2

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- Then  $\text{post}_{\text{bool}}^\# \circ \gamma_{\text{cartesian}}(\langle *, * \rangle) = \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}$   
whereas  $\text{post}_{\text{b.c.}}^\#(\langle *, * \rangle) = \langle *, * \rangle$ .

⇒ Lost of precision

Note that  $\text{post}_{\text{bool}}^\#$  is deterministic:

$$\text{post}_{\text{bool}}^\#(\{\langle 0, 0 \rangle\}) = \{\langle 0, 0 \rangle\} \text{ e.t.c.}$$

# Loss of Precision: Example 2

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- Then  $\text{post}_{\text{bool}}^\# \circ \gamma_{\text{cartesian}}(\langle *, * \rangle) = \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}$   
whereas  $\text{post}_{\text{b.c.}}^\#(\langle *, * \rangle) = \langle *, * \rangle$ .

⇒ Lost of precision

Note that  $\text{post}_{\text{bool}}^\#$  is deterministic:

$$\text{post}_{\text{bool}}^\#(\{\langle 0, 0 \rangle\}) = \{\langle 0, 0 \rangle\} \text{ e.t.c.}$$

Reason for the loss of precision:  $\text{post}_{\text{b.c.}}^\#$  applied to a trivector containing  $*$ .

# Control Points

Until now: predicates  $p_l$  for every control point  $l$  of the program:

$$p_l = 1 \iff \text{the state is at location } l$$

# Control Points

Until now: predicates  $p_l$  for every control point  $l$  of the program:

$$p_l = 1 \iff \text{the state is at location } l$$

$\implies$  lots of predicates, great loss of precision

# Control Points

Until now: predicates  $p_l$  for every control point  $l$  of the program:

$$p_l = 1 \iff \text{the state is at location } l$$

$\implies$  lots of predicates, great loss of precision

Now:

- Concrete domain: sequence of state spaces indexed by program locations  $((2^{\text{States}})^{\text{Loc}})$

# Control Points

Until now: predicates  $p_l$  for every control point  $l$  of the program:

$$p_l = 1 \iff \text{the state is at location } l$$

$\implies$  lots of predicates, great loss of precision

Now:

- Concrete domain: sequence of state spaces indexed by program locations  $((2^{\text{States}})^{\text{Loc}})$
- Its elements: vectors of sets of states ( $S = \langle S[l] \rangle_{l \in \text{Loc}}$ )  
States consist only of values representing data variables.

# Control Points

Until now: predicates  $p_l$  for every control point  $l$  of the program:

$$p_l = 1 \iff \text{the state is at location } l$$

$\implies$  lots of predicates, great loss of precision

Now:

- Concrete domain: sequence of state spaces indexed by program locations  $((2^{\text{States}})^{\text{Loc}})$
- Its elements: vectors of sets of states ( $S = \langle S[l] \rangle_{l \in \text{Loc}}$ )  
States consist only of values representing data variables.
- Abstract domain:  $(\text{AbsDom}_{\text{cartesian}})^{\text{Loc}}$

# Control Points

Until now: predicates  $p_l$  for every control point  $l$  of the program:

$$p_l = 1 \iff \text{the state is at location } l$$

$\implies$  lots of predicates, great loss of precision

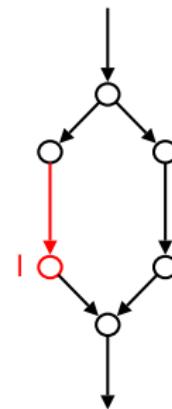
Now:

- Concrete domain: sequence of state spaces indexed by program locations  $((2^{\text{States}})^{\text{Loc}})$
- Its elements: vectors of sets of states ( $S = \langle S[l] \rangle_{l \in \text{Loc}}$ )  
States consist only of values representing data variables.
- Abstract domain:  $(\text{AbsDom}_{\text{cartesian}})^{\text{Loc}}$
- $\text{post} = \langle \text{post}[l] \rangle_{l \in \text{Loc}},$   
 $\text{post}_{\text{b.c.}}^\# = \langle \text{post}_{\text{b.c.}}^\#[l] \rangle_{l \in \text{Loc}}$

# post[ $l$ ]

Two cases:

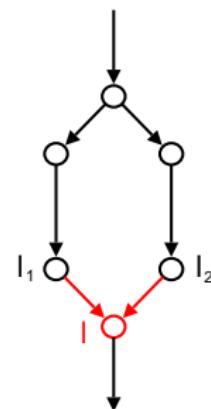
- $l$  has a unique predecessor:  
 $\text{post}[l]$  defined by the unique edge leading into  $l$



# post[ $l$ ]

Two cases:

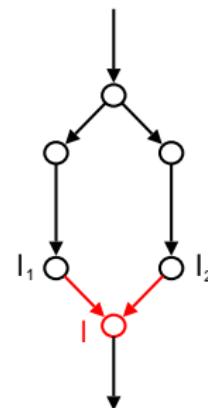
- $l$  has a unique predecessor:  
 $\text{post}[l]$  defined by the unique edge leading into  $l$
- $l$  is a “join” location with predecessors  $l_1, l_2$ :
  - $\text{post}[l](S) = S[l_1] \cup S[l_2]$
  - $\text{post}_{b.c}^\sharp[l](\langle \dots, v[l_1], \dots, v[l_2], \dots \rangle) = v[l_1] \sqcup v[l_2]$



# post[ $l$ ]

Two cases:

- $l$  has a unique predecessor:  
 $\text{post}[l]$  defined by the unique edge leading into  $l$
- $l$  is a “join” location with predecessors  $l_1, l_2$ :
  - $\text{post}[l](S) = S[l_1] \cup S[l_2]$
  - $\text{post}_{b.c}^\sharp[l](\langle \dots, v[l_1], \dots, v[l_2], \dots \rangle) = v[l_1] \sqcup v[l_2]$



## Example

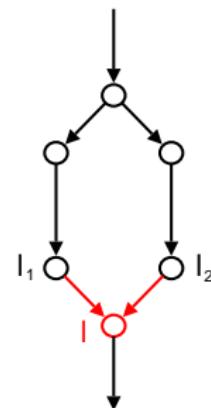
$$S[l_1] = \langle 0, 0 \rangle, S[l_2] = \langle 1, 1 \rangle$$

$$\Rightarrow \text{post}[l](S) = \langle *, * \rangle$$

# post[ $l$ ]

Two cases:

- $l$  has a unique predecessor:  
 $\text{post}[l]$  defined by the unique edge leading into  $l$
- $l$  is a “join” location with predecessors  $l_1, l_2$ :
  - $\text{post}[l](S) = S[l_1] \cup S[l_2]$
  - $\text{post}_{b.c}^\sharp[l](\langle \dots, v[l_1], \dots, v[l_2], \dots \rangle) = v[l_1] \sqcup v[l_2]$



## Example

$$S[l_1] = \langle 0, 0 \rangle, S[l_2] = \langle 1, 1 \rangle$$

$$\Rightarrow \text{post}[l](S) = \langle *, * \rangle$$

⇒ loss of precision

# Disjunctive Completion

Question: How to reduce loss of precision under Cartesian abstraction?

# Disjunctive Completion

Question: How to reduce loss of precision under Cartesian abstraction?

Disjunctive completion: trivectors  $\rightarrow$  sets of trivectors

# Disjunctive Completion

Question: How to reduce loss of precision under Cartesian abstraction?

Disjunctive completion: trivectors  $\rightarrow$  sets of trivectors

- $\text{AbsDom}_{b.c.\vee} = 2^{\{0,1,*\}^n}$

# Disjunctive Completion

Question: How to reduce loss of precision under Cartesian abstraction?

Disjunctive completion: trivectors  $\rightarrow$  sets of trivectors

- $\text{AbsDom}_{b.c.\vee} = 2^{\{0,1,*\}^n}$
- for  $V \in \text{AbsDom}_{b.c.\vee}$  :  
 $\text{post}_{b.c.\vee}^\#(V) = \{\text{post}_{b.c.}^\#(v) \mid v \in V\}$

# The Focus Operation ( $\text{focus}[1, 2]$ )

We define:

- $\text{focus}[1, 2](\langle v_1, v_2, v_3, \dots, v_n \rangle)$   
 $= \{\langle v'_1, v'_2, v_3, \dots, v_n \rangle \mid v'_1, v'_2 \in \{0, 1\}, v'_1 \leq v_1, v'_2 \leq v_2\}$

# The Focus Operation ( $\text{focus}[1, 2]$ )

We define:

- $\text{focus}[1, 2](\langle v_1, v_2, v_3, \dots, v_n \rangle)$   
 $= \{\langle v'_1, v'_2, v_3, \dots, v_n \rangle \mid v'_1, v'_2 \in \{0, 1\}, v'_1 \leq v_1, v'_2 \leq v_2\}$

## Example

$$\text{focus}[1, 2](\langle *, 1, * \rangle) = \{\langle 0, 1, * \rangle, \langle 1, 1, * \rangle\}$$

# The Focus Operation ( $\text{focus}[1, 2]$ )

We define:

- $\text{focus}[1, 2](\langle v_1, v_2, v_3, \dots, v_n \rangle) = \{\langle v'_1, v'_2, v_3, \dots, v_n \rangle \mid v'_1, v'_2 \in \{0, 1\}, v'_1 \leq v_1, v'_2 \leq v_2\}$

## Example

$$\text{focus}[1, 2](\langle *, 1, * \rangle) = \{\langle 0, 1, * \rangle, \langle 1, 1, * \rangle\}$$

- $\text{focus}[1, 2](V) = \bigcup_{v \in V} \text{focus}[1, 2](v)$

# The Focus Operation ( $\text{focus}[1, 2]$ )

We define:

- $\text{focus}[1, 2](\langle v_1, v_2, v_3, \dots, v_n \rangle) = \{\langle v'_1, v'_2, v_3, \dots, v_n \rangle \mid v'_1, v'_2 \in \{0, 1\}, v'_1 \leq v_1, v'_2 \leq v_2\}$

## Example

$$\text{focus}[1, 2](\langle *, 1, * \rangle) = \{\langle 0, 1, * \rangle, \langle 1, 1, * \rangle\}$$

- $\text{focus}[1, 2](V) = \bigcup_{v \in V} \text{focus}[1, 2](v)$
- $\text{post}_{\text{b.c.}\cdot\vee\cdot[1,2]}^\sharp(V) = \{\text{post}_{\text{b.c.}}^\sharp(v) \mid v \in \text{focus}[1, 2](V)\}$

# The Focus Operation: Example 2 (ctd.)

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- $\text{post}_{\text{b.c}}^{\#}(\langle *, * \rangle) = \langle *, * \rangle$

# The Focus Operation: Example 2 (ctd.)

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- $\text{post}_{b.c}^\#(\langle *, * \rangle) = \langle *, * \rangle$

Then:

$$\begin{aligned}& \text{post}_{b.c \cdot \vee \cdot [1,2]}^\#(\{\langle *, * \rangle\}) \\&= \text{post}_{b.c \cdot \vee}^\#(\{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle\}) \\&= \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}\end{aligned}$$

# The Focus Operation: Example 2 (ctd.)

- Program:
  - 2 Boolean variables  $x_1, x_2$
  - program “assume( $x_1 = x_2$ )”:  $x_1 = x_2 \wedge x'_1 = x_1 \wedge x'_2 = x_2$
- $\mathcal{P} = \{x_1, x_2\}$
- $\text{post}_{b.c}^\#(\langle *, * \rangle) = \langle *, * \rangle$

Then:

$$\begin{aligned}& \text{post}_{b.c.\vee.[1,2]}^\#(\{\langle *, * \rangle\}) \\&= \text{post}_{b.c.\vee}^\#(\{\langle 0, 0 \rangle, \langle 0, 1 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle\}) \\&= \{\langle 0, 0 \rangle, \langle 1, 1 \rangle\}\end{aligned}$$

⇒ no loss of precision under the ‘focussed’ abstraction

### Proposition 3

For every deterministic operator **post**, there exists a focus operation such that **post** does not loose precision under 'focussed' Cartesian abstraction.

### Proposition 3

For every deterministic operator **post**, there exists a focus operation such that **post** does not loose precision under ‘focussed’ Cartesian abstraction.

We define the abstract post operator  $\text{post}_{\text{slam}}^\#$  used in SLAM for each control point  $l$  as

$$\text{post}_{\text{slam}}^\#[l] = \text{post}_{b \cdot c \cdot \vee \cdot [1, \dots, n]}^\#[l]$$

# Conclusion

- Abstraction a crucial issue in model checking software
- SLAM approach:
  - automatic construction and refinement of abstractions
  - Combination of Boolean and Cartesian abstraction
  - Efficient computation of the abstract post operator

# For Further Reading

 T. Ball, A. Podelski, and S. K. Rajamani.

Boolean and Cartesian Abstraction for Model Checking C Programs.  
In *Proceedings of TACAS: Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 2031, pp. 268–283 Genova, Italy, April 2001. Springer-Verlag.

 S. Graf and H. Saidi.

Construction of abstract state graphs with PVS. In *Proceedings of the 9th International Conference on Computer Aided Verification (CAV 97)*, LNCS 1254, pp. 72–83, Haifa, Israel, June 1997.  
Springer-Verlag.