# Domain approximations for finite set constraint variables

## An integrated approach


Patrick Pekczynski

Supervisor: Guido Tack
Responsible Professor: Prof. Gert Smolka
Timeframe: February 2006 - January 2007

Programming Systems Lab
Department of Computer Science
Saarland University, Saarbrücken


12.02.2007

# Outline

# Mainstream

## Constraint Programming

- largely restricted to finite domain variables (FDVar)

### Example

- variables:
  - $x \in [1..6]$
  - $y \in \{1, 3, 6, 8, 12\}$
  - $z \in \{10\}$
- constraints as relations between variables
  - $x + y = z$
  - $x \neq y$

# Beyond finite domains

## When do we use sets?

- constraints are domain specific
- interested in collection of elements
- symmetries among elements have to be avoided
  - students in tutorial groups
  - players in a team
  - workers at a shift
- use finite set variables (FSVar)

# Beyond finite domains

## Example

- variables:
  - $g \in \{\{1,3\},\{2,7,12\},\{11,\ldots,14\}\}$
  - $h \in \{\{1,3,5,6\},\{7,9,13\},\{1,\ldots,20\}\}$
  - $u \in \{\emptyset,\ldots,\{1,\ldots,20\}\}$
- constraints:
  - $g \subset h$
  - $|h| = 4$
  - $u = g \cup h$

# Set variables as computation domain

## Finite domain constraint variable $x : D$

### Definition

- Variable $x \in \mathtt{Var}$
- associated with finite domain $D \in Dom$

### Components for integer variable

- Finite set of *variables* $\mathtt{Var}$.
- Finite universe $\mathcal{U} \subset \mathbb{Z}$.
- Finite set of *values* $\mathtt{Val} = \mathcal{U}$.
- Finite set of possible *domains* $Dom = \mathcal{P}(\mathtt{Val})$.

# Set variables as computation domain

## Finite domain constraint variable $x : D$

### Definition

- Variable $x \in \mathtt{Var}$
- associated with finite domain $D \in Dom$

### Components for set variable

- Finite set of *variables* $\mathtt{Var}$.
- Finite universe $\mathcal{U} \subset \mathbb{Z}$.
- Finite set of *values* $\mathtt{Val} = \mathcal{P}(\mathcal{U})$.
- Finite set of possible *domains* $Dom = \mathcal{P}(\mathtt{Val})$.

# Representation problem

## Size Issue

- Assume set variable $x : D = \mathcal{P}(\{1, \ldots, 400\})$
- $|D| = 2^{400}$
- Naive enumeration of all values $\Rightarrow$ exponential size $\mathcal{O}\left(2^N\right)$
- impracticable representation

# Domain Approximation - a viable solution

## Domain Approximation $\mathcal{A}$

- theoretical framework by Benhamou [Ben96]
- representative subset
  $\mathcal{A} \subseteq Dom$
- closed under intersection
  $\forall A, B \in \mathcal{A} : (A \cap B) \in \mathcal{A}$
- Elements of $\mathcal{A}$ are called *approximate domains*

### Required approximate domains

| | |
|---|---|
| $\emptyset$ | set with no values |
| `Val` | set with all values |
| $D \in Dom, \lvert D \rvert = 1$ | sets containing a single value |

# What approximations are there?

## Overview of approximations

- $(\mathcal{S})$ Set bounds approximation
- $(\mathcal{C})$ Cardinality set bounds approximation
- $(\mathcal{F})$ Full domain approximation

# Set bounds approximation $(\mathcal{S})$

## Theoretical foundations of $(\mathcal{S})$

- Puget in [Pug92]
  - First to introduce set bounds representation in constraint programming
- Gervet in [Ger95, chp. 4]
  - Describes representation in full detail
  - Reference implementation for set constraint solver *Conjunto* [Ger94]

# Set bounds approximation $(\mathcal{S})$

## Convex Set Bounds $(\mathcal{S})$

- approximate a domain $D \in Dom$
- $E \in (\mathcal{S})$ is the smallest convex interval with respect to $\subseteq$ containing $D$

$$\begin{aligned} E &= [\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq} \\ &= \{T \in Dom \mid \inf(D) \subseteq T \subseteq \sup(D)\} \\ &= [\inf(D) .. \sup(D)]_{\subseteq} \\ &= \left[\bigcap_{d \in D} d .. \bigcup_{d \in D} d\right]_{\subseteq} \end{aligned}$$

- $(\mathcal{S}) \overset{\text{def}}{=} \{E \in Dom \mid E \text{ is convex wrt. } \subseteq\}$

# Set Bounds Approximation $(\mathcal{S})$ - Pros and Cons

## Pros $(\mathcal{S})$

- guaranteed linear size
- space efficiency by definition of $(\mathcal{S})$
  - represent only two sets $\lfloor E \rfloor$, $\lceil E \rceil$ instead of exponentially many
- *extension* property identified by Gervet in [Ger95, sect.4.2.3 p.45]:
  - set variable $x : E$, $E \in (\mathcal{S})$
  - variable assignment $\alpha \in \mathtt{Var} \to \mathtt{Val}$:

$$\begin{aligned} \forall v \in \lfloor E \rfloor \quad &\Rightarrow \quad v \in \alpha(x) \\ \forall v \notin \lceil E \rceil \quad &\Rightarrow \quad v \notin \alpha(x) \end{aligned}$$

# Set Bounds Approximation $(\mathcal{S})$ - Pros and Cons

## Pros $(\mathcal{S})$

- guaranteed linear size
- space efficiency by definition of $(\mathcal{S})$
  - represent only two sets $\lfloor E \rfloor, \lceil E \rceil$ instead of exponentially many
- *extension* property identified by Gervet in [Ger95, sect.4.2.3 p.45]:
  - set variable $x : E, E \in (\mathcal{S})$
  - variable assignment $\alpha \in \texttt{Var} \to \texttt{Val}$:

$$\begin{array}{lcl} \forall v \in \lfloor E \rfloor & \Rightarrow & v \in \alpha(x) \\ \forall v \notin \lceil E \rceil & \Rightarrow & v \notin \alpha(x) \end{array}$$

## Cons $(\mathcal{S})$

- $\lfloor E \rfloor$ represented twice, since $\lfloor E \rfloor \subseteq \lceil E \rceil$.

# Cardinality set bounds approximation $(\mathcal{C})$

## More fine grained version of $(\mathcal{S})$

- Used in most constraint solvers:
  - *Mozart* [The06b]
  - *Gecode* [The06a]
  - *ILOG* [ILO00]
- based on set bounds approximation $(\mathcal{S})$
- imposes additional cardinality restrictions

# Cardinality set bounds approximation $\left(\mathcal{C}\right)$

## Extending $\left(\mathcal{S}\right)$ to $\left(\mathcal{C}\right)$

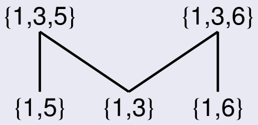
- set variable $x : E$ and $E \in \left(\mathcal{S}\right)$
- adding basic cardinality constraints $l \leq |x| \leq r$
- translate into cardinality restrictions for $\lfloor E \rfloor$ and $\lceil E \rceil$:
  $\mathtt{card}(E, l, r) = l \leq |\lfloor E \rfloor| \wedge |\lceil E \rceil| \leq r$
- $\left(\mathcal{C}\right)_l^r \stackrel{\text{def}}{=} \left(\mathcal{S}\right) \cap \left\{T \in \left(\mathcal{S}\right) \mid \mathtt{card}(T, l, r)\right\}$

# Representing a set variable $x : D$ in $(\mathcal{C})$

## Set variable $x : D$

$D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\},\{1,3,6\}\}$

## Corresponding Hesse-Diagram

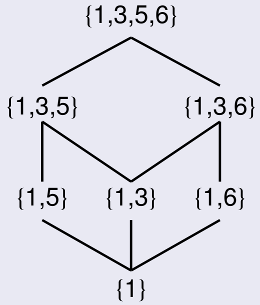


# Representing a set variable $x : D$ in $\left(\mathcal{C}\right)$

## Approximate domain $E \in \left(\mathcal{S}\right)$

$E = [\{1\}..\{1, 3, 5, 6\}]_{\subseteq}$

## Corresponding Hesse-Diagram

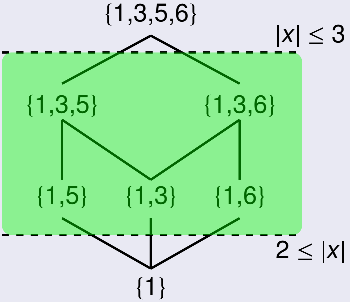


# Representing a set variable $x : D$ in $(\mathcal{C})$

## Approximate domain $F \in (\mathcal{C})_2^3$

$$F = E \cap \{T \in (\mathcal{S}) \mid \mathtt{card}(T, 2, 3)\}$$

## Corresponding Hesse-Diagram



# Full domain approximation $\left(\mathcal{F}\right)$

## From Dom to $\left(\mathcal{F}\right)$

- choose *Dom* itself as approximation of *Dom*
- approximate a domain $D \in Dom$ by $D$
- $\left(\mathcal{F}\right) \stackrel{\text{def}}{=} Dom$

# Full domain approximation $\left(\mathcal{F}\right)$ - Pros and Cons

## Pros $\left(\mathcal{F}\right)$

- Exact representation of the complete domain
- Encodes characteristic function

$$\chi_D : \mathbb{Z} \mapsto \mathbb{B} : \chi_D(i) = \begin{cases} 1 & \text{if } i \in D \\ 0 & \text{otherwise} \end{cases}$$

to represent a set $D$

# Full domain approximation $(\mathcal{F})$ - Pros and Cons

## Pros $(\mathcal{F})$

- Exact representation of the complete domain
- Encodes characteristic function

$$\chi_D : \mathbb{Z} \mapsto \mathbb{B} : \chi_D(i) = \begin{cases} 1 & \text{if } i \in D \\ 0 & \text{otherwise} \end{cases}$$

to represent a set $D$

## Cons $(\mathcal{F})$

- Space efficiency must be obtained by choice of data structure
- With full approximation still exponential size possible

# Efficient data structure for $\left(\mathcal{F}\right)$

## Theoretical foundations

- Hawkins Lagoon and Stuckey in [HLS04]
  - First to introduce a full domain approximation
- Use reduced ordered binary decision diagrams (ROBDDs)

# Efficient data structure for $\left(\mathcal{F}\right)$

## Representing a set variable $x : D$ in $\left(\mathcal{F}\right)$

- $D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\},\{1,3,6\}\}$
- vector of Boolean variables $b = \langle b_1, b_2, b_3, b_4, b_5, b_6 \rangle$
- *ROBDD* representing all valuations of formula $\phi$

$$
\begin{array}{cllr}
\phi & = & \neg\,(b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge \neg b_6) & \neg\{1\} \\
 & \vee & b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge \neg b_6 & \{1,3\} \\
 & \vee & b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge b_5 \wedge \neg b_6 & \{1,5\} \\
 & \vee & b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge b_6 & \{1,6\} \\
 & \vee & b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge b_5 \wedge \neg b_6 & \{1,3,5\} \\
 & \vee & b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge b_6 & \{1,3,6\} \\
 & \vee & \neg\,(b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge b_5 \wedge b_6) & \neg\{1,3,5,6\}
\end{array}
$$

# Efficient data structure for $\left(\mathcal{F}\right)$

## Representing a set variable $x : D$ in $\left(\mathcal{F}\right)$

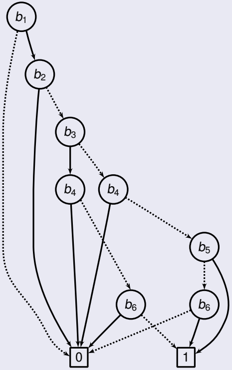
- $D = [\{1\}..\{1,3,5,6\}]_{\subseteq} \setminus \{\{1\},\{1,3,5,6\}\}$
- vector of Boolean variables $b = \langle b_1, b_2, b_3, b_4, b_5, b_6 \rangle$
- *ROBDD* representing all valuations of formula $\phi$

$$\begin{array}{rcl} \phi & = & b_1 \wedge \neg b_2 \wedge \neg b_4 \\ & \wedge & \neg\left(b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge \neg b_6\right) \\ & \wedge & \neg\left(b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge b_5 \wedge b_6\right) \end{array}$$

# Efficient data structure for $\left(\mathcal{F}\right)$

## Representing a set variable $x : D$ in $\left(\mathcal{F}\right)$

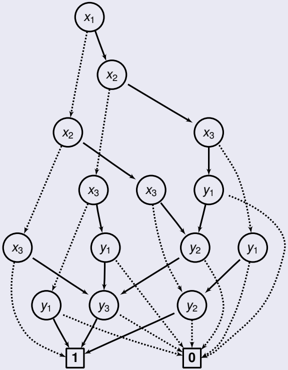
- $D = [\{1\}..\{1,3,5,6\}]_{\subseteq} \setminus \{\{1\},\{1,3,5,6\}\}$
- vector of Boolean variables $b = \langle b_1, b_2, b_3, b_4, b_5, b_6 \rangle$
- *ROBDD* representing all valuations of formula $\phi$



# Set constraints in $(\mathcal{F})$

## Modeling advantage: constraints as ROBDDs

- constraint $x \subseteq y$, $x, y : \mathcal{P}(\{1, 2, 3\})$
- naive modeling yields:



# Set constraints in $(\mathcal{F})$

## Modeling advantage: constraints as ROBDDs

- constraint $x \subseteq y$, $x, y : \mathcal{P}(\{1, 2, 3\})$
- Choosing the variable order as
  1. $\mathcal{V} = \{v_1, \ldots, v_m\}$
  2. associated vectors of Boolean variables $\langle v_{i,1}, \ldots, v_{i,N} \rangle$, where $i \in \{1, \ldots, m\}$ and $N \stackrel{\text{def}}{=} |\mathcal{U}|$
  3. Fix variable order $(\prec)$ as

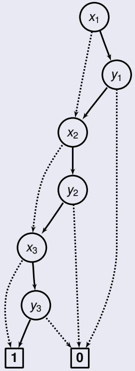
$$v_{1,1} \prec v_{1,m} \prec v_{1,2} \prec \cdots \prec v_{1,N} \prec \cdots \prec v_{m,N}$$

  4. *guarantees* linear size of ROBDD except cardinality constraints [LS04]

# Set constraints in $(\mathcal{F})$

## Modeling advantage: constraints as ROBDDs

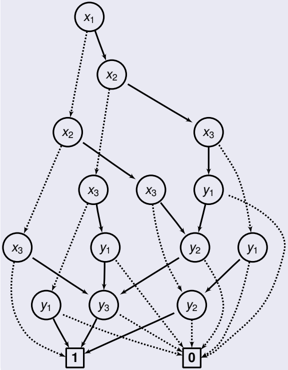
- constraint $x \subseteq y$, $x, y : \mathcal{P}(\{1, 2, 3\})$
- According to specified order:

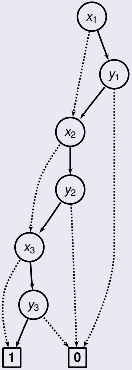


# Set constraints in $(\mathcal{F})$

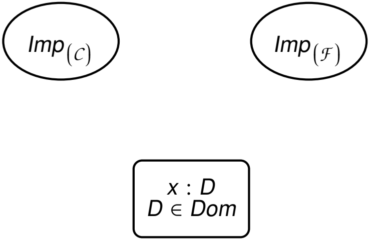
## Modeling advantage: constraints as ROBDDs

- constraint $x \subseteq y$, $x, y : \mathcal{P}(\{1, 2, 3\})$
- Comparison



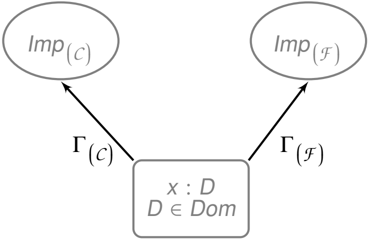



# Different domain approximations available




- Howto connect them ?

# Different domain approximations available




- Howto connect them ?
- Using variable views

# Variable Views

## Variable Views [ST06]

1. Mapping $V : \mathcal{A} \to \mathcal{B}$ between domain approximations $\mathcal{A}$ and $\mathcal{B}$
2. Adaptor for given domain approximation $\mathcal{A}$
   - map $D \in Dom$ to $A \in \mathcal{A}$
   - prescribe internal representation (data structures)
3. Propagation interface providing propagation services
   - domain lookup
   - domain update
4. Simulating non-existing variable representations using existing variable representations

# Using views to connect approximations

## Adaptor functionality

- Set bounds view $\Gamma_{(\mathcal{S})} : \mathcal{A} \to (\mathcal{S})$
  $$\Gamma_{(\mathcal{S})}(A) = \left[\bigcap_{a \in A} a .. \bigcup_{a \in A} a\right]_{\subseteq}$$
- Cardinality set bounds view $\Gamma_{(\mathcal{C})} : \mathcal{A} \to (\mathcal{C})$
  $$\Gamma_{(\mathcal{C})}(A) = \Gamma_{(\mathcal{S})}(A) \cap \left\{T \in (\mathcal{S}) \mid \mathtt{card}(T, l, r)\right\}$$
- Full domain view $\Gamma_{(\mathcal{F})} : \mathcal{A} \to (\mathcal{F})$
  $$\Gamma_{(\mathcal{F})}(A) = A$$

# Using views as propagation interface

## Modeling set constraints in $(\mathcal{C})$

- $x : D = [\lfloor D \rfloor .. \lceil D \rceil]_{\subseteq}$
- $y : E = [\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$
- $z : F = [\lfloor F \rfloor .. \lceil F \rceil]_{\subseteq}$

| constraint | propagator $p_{(\mathcal{C})}$ |
|---|---|
| $x \subseteq y$ | $x \subseteq \lceil E \rceil \wedge \lfloor D \rfloor \subseteq y$ |
| $x \cap y = z$ | $\lfloor D \rfloor \cap \lfloor E \rfloor \subseteq z \wedge z \subseteq \lceil D \rceil \cap \lceil E \rceil$ |
| | $\lfloor F \rfloor \subseteq x \wedge x \nsubseteq \lfloor E \rfloor \setminus \lceil F \rceil$ |
| | $\lfloor F \rfloor \subseteq y \wedge y \nsubseteq \lfloor D \rfloor \setminus \lceil F \rceil$ |

# Using views as propagation interface

## Access to data structures

- propagators $p_{(\mathcal{C})}$ on set variable $x : E = [\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$
  - lookup interval bounds $\lfloor E \rfloor$ and $\lceil E \rceil$
  - modify interval bounds $\lfloor E \rfloor$ and $\lceil E \rceil$
- forwarded by propagation interface through variable view $Id_{(\mathcal{C})} : \left(\mathcal{C}\right) \rightarrow \left(\mathcal{C}\right), Id_{(\mathcal{C})}(E) = E$

# Using views as propagation interface

## Domain information through iteration

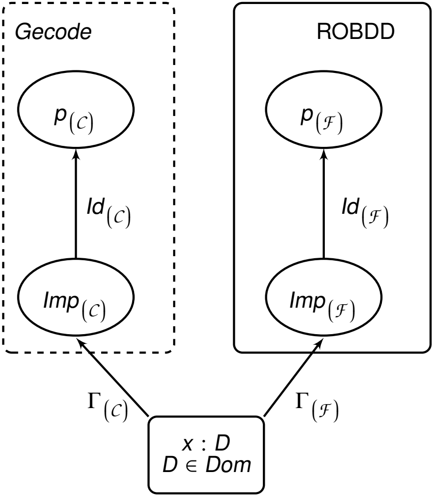
- introduced by Schulte and Tack [ST06]
- iterator `iter` provides functions:

  | | |
  |---|---|
  | `operator()()` | test whether we can iterate further |
  | `operator++()` | increment to next value in set |

- depending on structure:

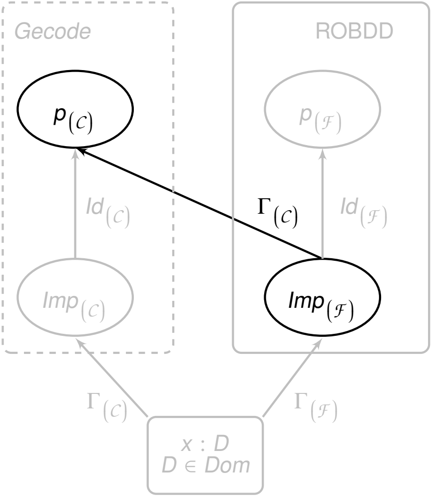
  | | |
  |---|---|
  | `val()` | value access |
  | `min()` | minimum of subinterval |
  | `max()` | maximum of subinterval |

# Combining orthogonal concepts




- Orthogonal concepts
- Do they just coexist ?

# Combining orthogonal concepts



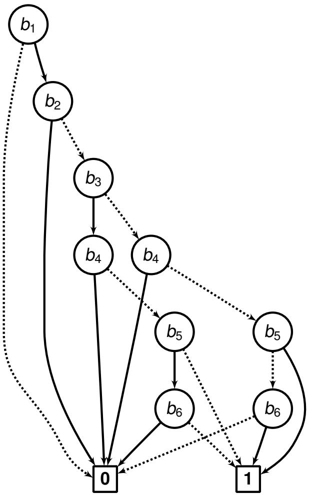

- Orthogonal concepts
- Do they just coexist ?
- No, we can connect them using variable views

# Crossing domain approximations

### From $(\mathcal{F})$ to $(\mathcal{C})$

- Simulate Cardinality set bounds interface for ROBDD representing $D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\},\{1,3,6\}\}$
- Apply view $\Gamma_{(\mathcal{C})}$ on domain $D \in (\mathcal{F})$.
  1. extract set bounds
  2. extract cardinality bounds

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

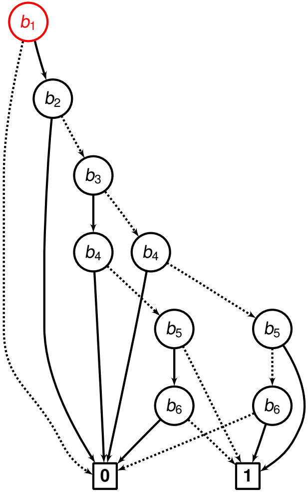
**Inspected node**

**Flag**

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

**Inspected node**

$(b_1, b_2, \perp)$ **Card (0,0)**

**Flag**

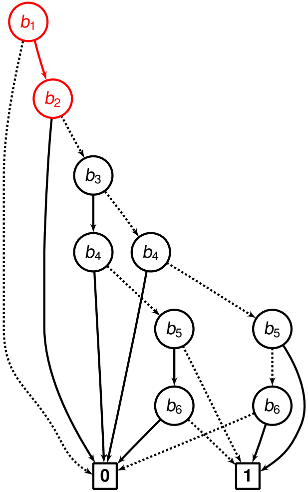
**INIT**

$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$

$[\emptyset..\{1, 2, 3, 4, 5, 6\}]_{\subseteq}$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




Result Start

**Stack**

$(b_2, \perp, b_3)$

**Inspected node**

$(b_1, b_2, \perp)$ **Card (0,0)**

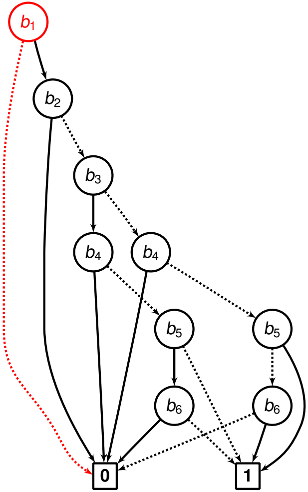
**Flag**

`INIT`

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

$[\emptyset..\{1, 2, 3, 4, 5, 6\}]_{\subseteq}$

# Simulate $\left(\mathcal{C}\right)$ representation with $\left(\mathcal{F}\right)$ using $\Gamma_{\left(\mathcal{C}\right)}$




Result Start

**Stack**

$(b_2, \perp, b_3)$

**Inspected node**

$(b_1, b_2, \perp)$ **Card (0,0)**

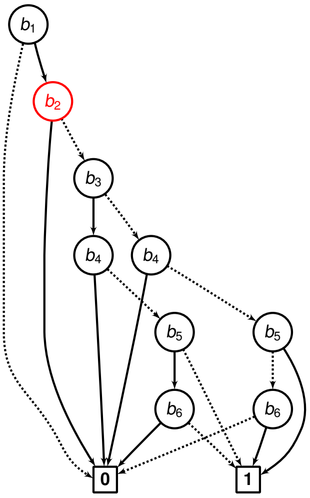
**Flag**

`FIX_GLB`

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

$[\{1\}..\{1,2,3,4,5,6\}]_{\subseteq}$

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

**Inspected node**

$(b_2,\perp,b_3)$ **Card (1,1)**

**Flag**

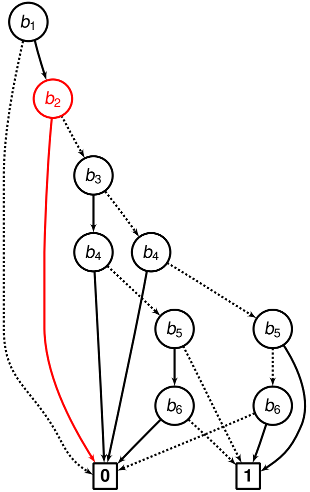
UNDET

**$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$**

$[\{1\}..\{1,2,3,4,5,6\}]_\subseteq$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

**Inspected node**

$(b_2, \perp, b_3)$ **Card (1,1)**

**Flag**

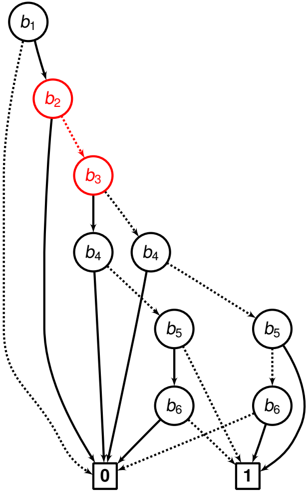
**FIX_NOT_LUB**

$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$

$[\{1\}..\{1,3,4,5,6\}]_{\subseteq}$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_3, b_4, b_4)$

**Inspected node**

$(b_2, \perp, b_3)$ **Card (1,1)**

**Flag**

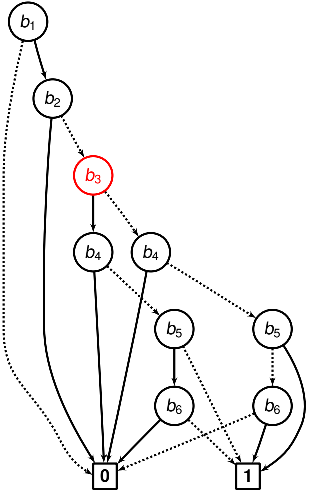
**FIX_NOT_LUB**

**$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$**

$[\{1\}..\{1,3,4,5,6\}]_\subseteq$

Result  Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

**Inspected node**

$(b_3, b_4, b_4)$ **Card (1,1)**

**Flag**

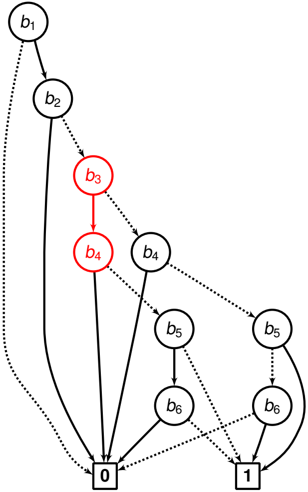
**FIX_UNKNOWN**

$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$

$[\{1\}..\{1, 3, 4, 5, 6\}]_\subseteq$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




Result Start

**Stack**

$(b_4, \perp, b_5)$

**Inspected node**

$(b_3, b_4, b_4)$ **Card (1,1)**

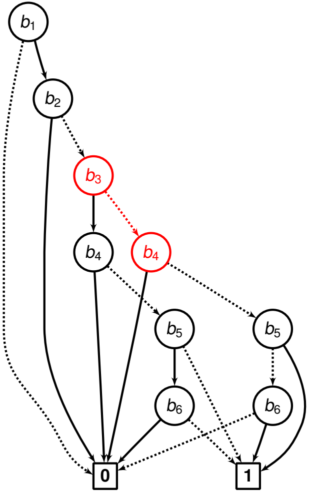
**Flag**

**FIX_UNKNOWN**

$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$

$[\{1\}..\{1,3,4,5,6\}]_\subseteq$

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_4,\perp,b_5)$ $(b_4,\perp,b_5)$

**Inspected node**

$(b_3,b_4,b_4)$ **Card (1,1)**

**Flag**

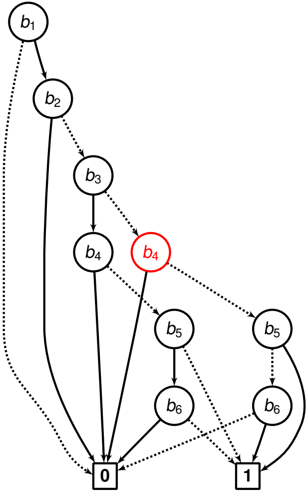
**FIX_UNKNOWN**

$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$

$[\{1\}..\{1,3,4,5,6\}]_\subseteq$

Result  Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_4, \perp, b_5)$

**Inspected node**

$(b_4, \perp, b_5)$ **Card (1,1)**

**Flag**

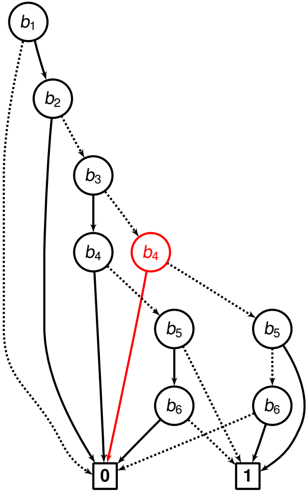
**UNDET**

$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$

$[\{1\}..\{1,3,4,5,6\}]_\subseteq$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_4,\perp,b_5)$

**Inspected node**

$(b_4,\perp,b_5)$ **Card (1,1)**

**Flag**

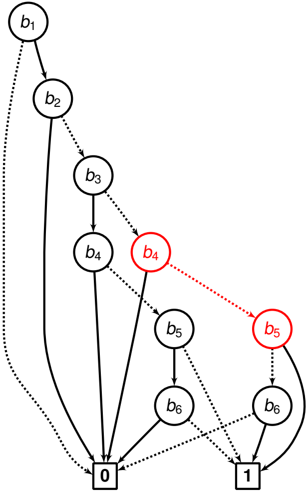
**FIX_NOT_LUB**

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

$[\{1\}..\{1,3,4,5,6\}]_{\subseteq}$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




▸ Result ▸ Start

**Stack**

$(b_5, \top, b_6)$ $(b_4, \bot, b_5)$

**Inspected node**

$(b_4, \bot, b_5)$ **Card (1,1)**

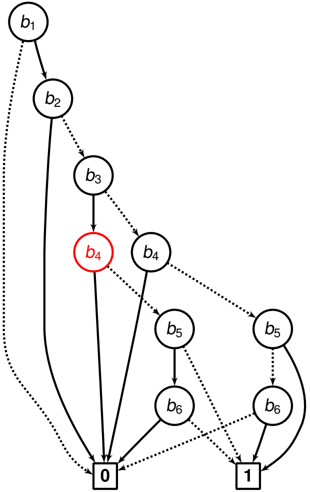
**Flag**

**FIX_NOT_LUB**

$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$

$[\{1\}..\{1, 3, 4, 5, 6\}]_{\subseteq}$

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_5, \top, b_6)$

**Inspected node**

$(b_4, \perp, b_5)$ **Card (2,2)**

**Flag**

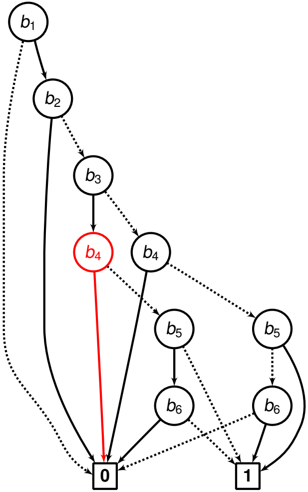
**FIX_NOT_LUB**

**$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$**

$[\{1\}..\{1,3,4,5,6\}]_\subseteq$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




Result    Start

**Stack**

$(b_5,\top,b_6)$

**Inspected node**

$(b_4,\perp, b_5)$ **Card (2,2)**

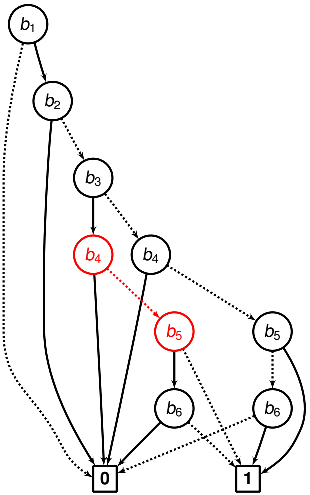
**Flag**

**FIX_NOT_LUB**

**$[\lfloor E\rfloor..\lceil E\rceil]_\subseteq$**

$[\{1\}..\{1,3,5,6\}]_\subseteq$

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_5,\top,b_6)$ $(b_5,b_6,\top)$

**Inspected node**

$(b_4,\bot,b_5)$ **Card (2,2)**

**Flag**

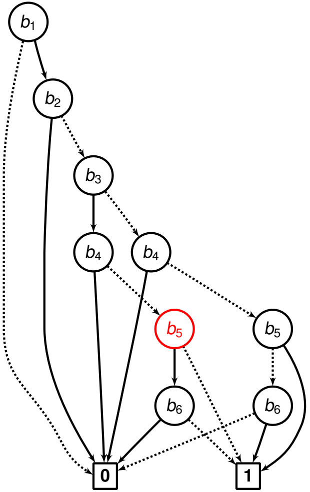
**FIX_NOT_LUB**

$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$

$[\{1\}..\{1,3,5,6\}]_{\subseteq}$

Result  Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




Result Start

**Stack**

$(b_5, \top, b_6)$

**Inspected node**

$(b_5, b_6, \top)$ **Card (2,2)**

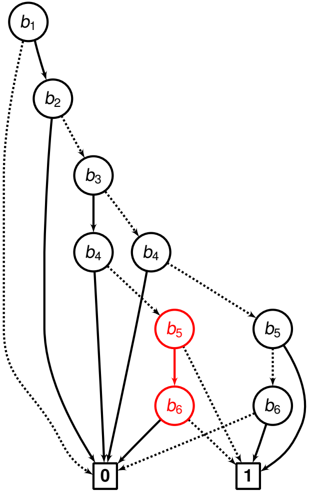
**Flag**

UNDET

**$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$**

$[\{1\}..\{1,3,5,6\}]_\subseteq$

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_5, \top, b_6)$ $(b_6, \bot, \top)$

**Inspected node**

$(b_5, b_6, \top)$ **Card (2,2)**

**Flag**

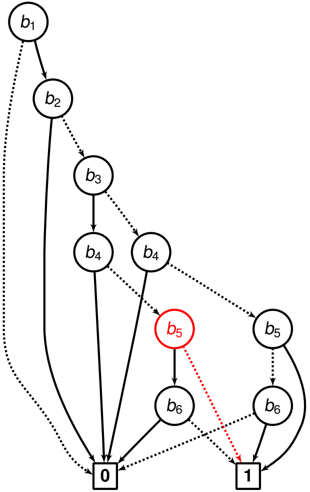
**UNDET**

**$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$**

$[\{1\}..\{1,3,5,6\}]_\subseteq$

Result  Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_5,\top,b_6)$ $(b_6,\bot,\top)$

**Inspected node**

$(b_5,b_6,\top)$ **Card (2,2)**

**Flag**

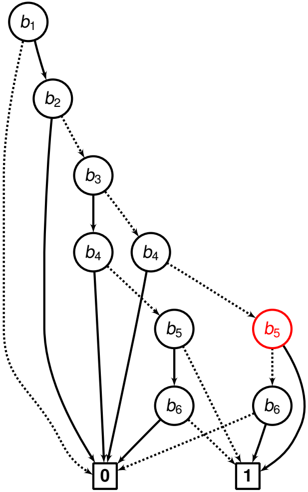
`FIX_NOT_LUB`

**$[\lfloor E\rfloor..\lceil E\rceil]_\subseteq$**

$[\{1\}..\{1,3,5,6\}]_\subseteq$

Result Start

# Simulate $\left(\mathcal{C}\right)$ representation with $\left(\mathcal{F}\right)$ using $\Gamma_{\left(\mathcal{C}\right)}$




**Stack**

$(b_6,\bot,\top)$

**Inspected node**

$(b_5,\top,b_6)$ **Card (1,1)**

**Flag**

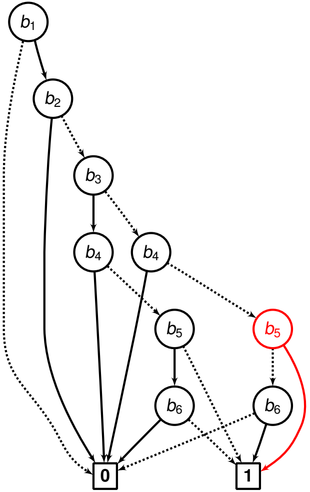
**FIX_NOT_LUB**

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

$[\{1\}..\{1,3,5,6\}]_{\subseteq}$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_6, \perp, \top)$

**Inspected node**

$(b_5, \top, b_6)$ **Card (1,1)**

**Flag**

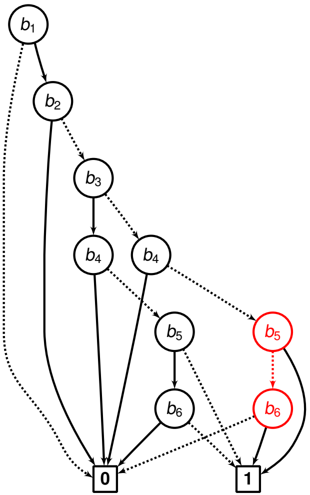
**UNDET**

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

$[\{1\}..\{1, 3, 5, 6\}]_{\subseteq}$

Result    Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_6,\top,\bot)$ $(b_6,\bot,\top)$

**Inspected node**

$(b_5,\top,b_6)$ **Card (1,1)**

**Flag**

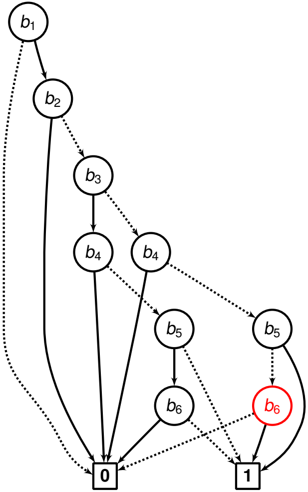
**UNDET**

$[\lfloor E\rfloor..\lceil E\rceil]_\subseteq$

$[\{1\}..\{1,3,5,6\}]_\subseteq$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_6, \perp, \top)$

**Inspected node**

$(b_6, \top, \perp)$ **Card (1,1)**

**Flag**

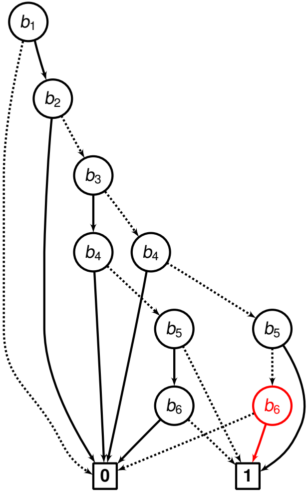
**UNDET**

$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$

$[\{1\}..\{1,3,5,6\}]_{\subseteq}$

Result  Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

$(b_6, \perp, \top)$

**Inspected node**

$(b_6, \top, \perp)$ **Card (2,2)**

**Flag**

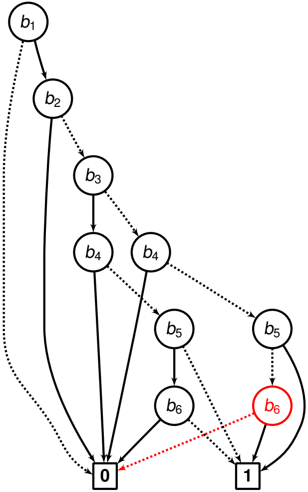
**FIX_GLB**

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

$[\{1\}..\{1, 3, 5, 6\}]_{\subseteq}$

Result Start

# Simulate $\left(\mathcal{C}\right)$ representation with $\left(\mathcal{F}\right)$ using $\Gamma_{\left(\mathcal{C}\right)}$




**Stack**

$(b_6, \perp, \top)$

**Inspected node**

$(b_6, \top, \perp)$ **Card (2,2)**

**Flag**

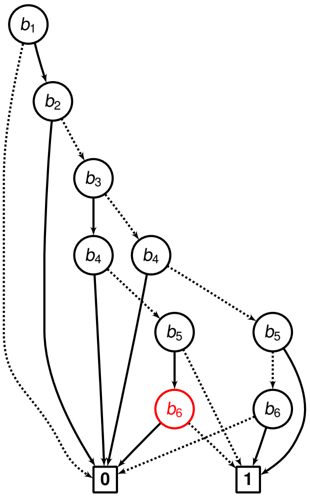
`FIX_GLB`

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

$[\{1\}..\{1,3,5,6\}]_{\subseteq}$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

**Inspected node**

$(b_6, \perp, \top)$ **Card (3,3)**

**Flag**

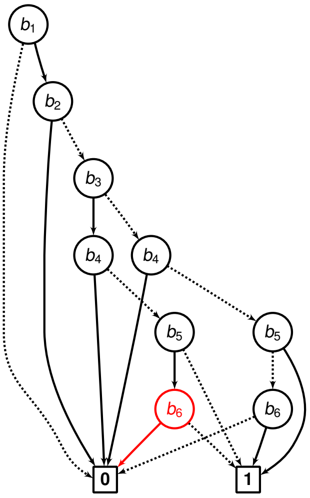
`FIX_GLB`

$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$

$[\{1\}..\{1, 3, 5, 6\}]_\subseteq$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




Result Start

**Stack**

**Inspected node**

$(b_6, \bot, \top)$ **Card (3,3)**

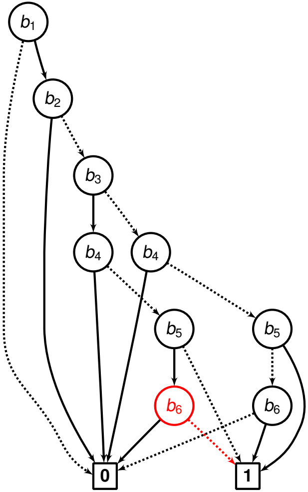
**Flag**

**UNDET**

**$[\lfloor E \rfloor .. \lceil E \rceil]_\subseteq$**

$[\{1\}..\{1,3,5,6\}]_\subseteq$

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




Result Start

**Stack**

**Inspected node**

$(b_6,\perp,\top)$ **Card (3,3)**

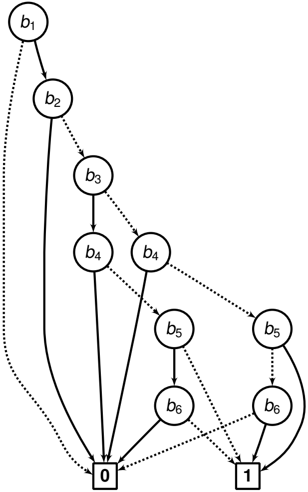
**Flag**

**UNDET**

$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$

$[\{1\}..\{1,3,5,6\}]_{\subseteq}$

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$




**Stack**

**Inspected node**

**Flag**

**$[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$**

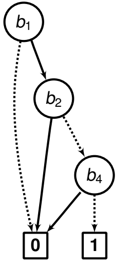
$[\{1\}..\{1, 3, 5, 6\}]_{\subseteq}$

Result Start

# Simulate $(\mathcal{C})$ representation with $(\mathcal{F})$ using $\Gamma_{(\mathcal{C})}$

## Resulting

- resulting ROBDD represents
  1. $E = \Gamma_{(\mathcal{C})}(D) = [\{1\}..\{1,3,5,6\}]_{\subseteq}$
  2. cardinality restricitions `card(E,2,3)`



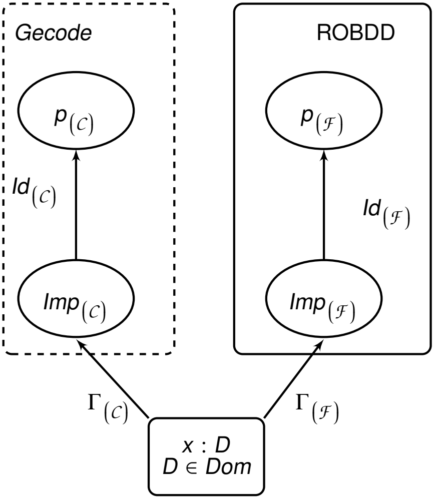

## Cardinality

$c \in [2..3]$

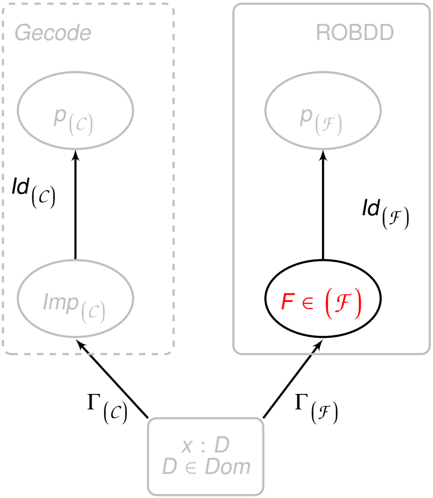
## $[\lfloor E \rfloor .. \lceil E \rceil]_{\subseteq}$

$[\{1\}..\{1,3,5,6\}]_{\subseteq}$

# Weaken propagation

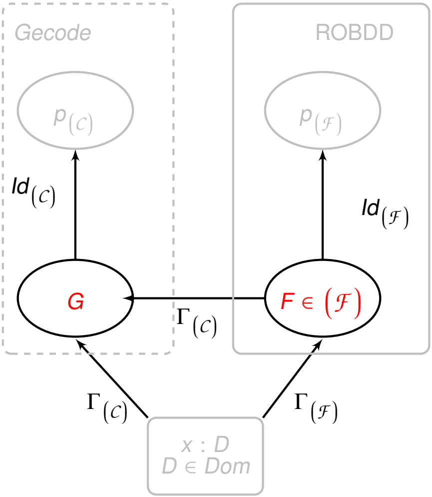


# Weaken propagation



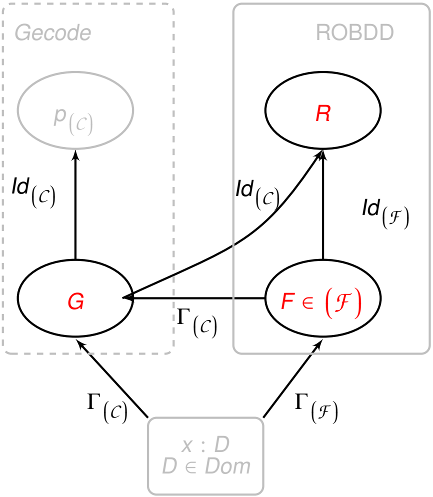

1. $F \in (\mathcal{F})$ is $x$-component of domain tuple $\vec{F}$, $\vec{F}.x = F$

# Weaken propagation



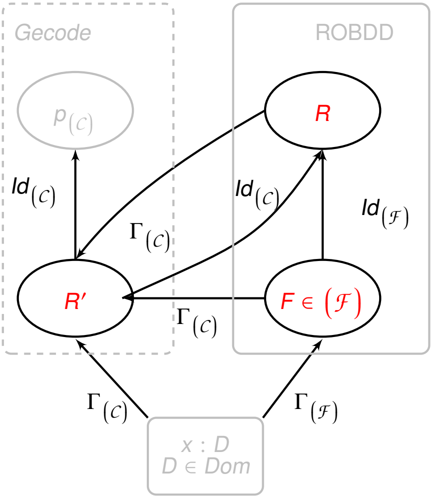

1. $F \in (\mathcal{F})$ is $x$-component of domain tuple $\vec{F}$, $\vec{F}.x = F$
2. Map $F$ to the respective cardinality set bounds $G = \Gamma_{(\mathcal{C})}(F)$

# Weaken propagation




1. $F \in (\mathcal{F})$ is $x$-component of domain tuple $\vec{F}$, $\vec{F}.x = F$
2. Map $F$ to the respective cardinality set bounds $G = \Gamma_{(\mathcal{C})}(F)$
3. Since $G \in (\mathcal{C}) \subset (\mathcal{F})$ apply $p_{(\mathcal{F})} : (\mathcal{F})^n \to (\mathcal{F})^n$
4. Propagation result $R = p_{(\mathcal{F})}(\vec{G}).x$

# Weaken propagation




1. $F \in (\mathcal{F})$ is $x$-component of domain tuple $\overrightarrow{F}$, $\overrightarrow{F}.x = F$
2. Map $F$ to the respective cardinality set bounds $G = \Gamma_{(\mathcal{C})}(F)$
3. Since $G \in (\mathcal{C}) \subset (\mathcal{F})$ apply $p_{(\mathcal{F})} : (\mathcal{F})^n \to (\mathcal{F})^n$
4. Propagation result $R = p_{(\mathcal{F})}(\overrightarrow{G}).x$
5. Map result $R$ again to $R' = \Gamma_{(\mathcal{C})}(R)$.

# Using variable views to weaken propagation

## Changing consistency of propagation result

- Use a domain-consistent propagator

$$p_{(\mathcal{F})} : (\mathcal{F})^n \to (\mathcal{F})^n$$

- obtain bounds( $(\mathcal{C})$ )-consistent propagation $\beta_{(\mathcal{C})} : (\mathcal{F})^n \to (\mathcal{C})^n$,
  $\beta_{(\mathcal{C})}(\vec{F}) = \Gamma_{(\mathcal{C})}(p_{(\mathcal{F})}(\Gamma_{(\mathcal{C})}(\vec{F})))$

# Contributions

1. Implemented presented concepts in *Gecode*

# Contributions

## Gecode Constraint Library

- **ge**neric
- **co**nstraint
- **de**velopment
- **e**nvironment

***Gecode*** [The06a], a C++ library for constraint programming. Version 1.3.1 available from http://www.gecode.org

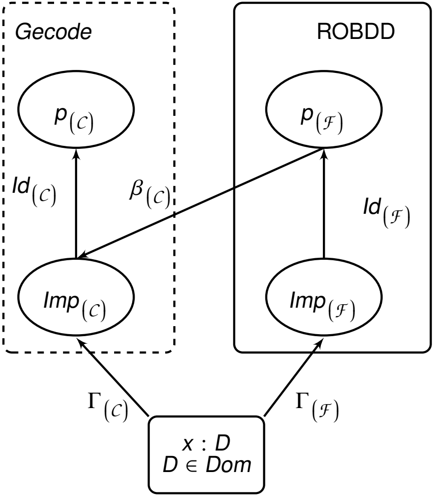
## Developers

- **Dr. Christian Schulte** (head, KTH, Sweden)
- **Guido Tack** (PS Lab, Saarbrücken, Germany)

# Contributions

1. Implemented presented concepts in *Gecode*
   1. ROBDD set component
   2. Simulation $(\mathcal{C})$ with $(\mathcal{F})$ using view $\Gamma_{(\mathcal{C})}$
   3. Propagation across approximations using $\beta_{(\mathcal{C})}$

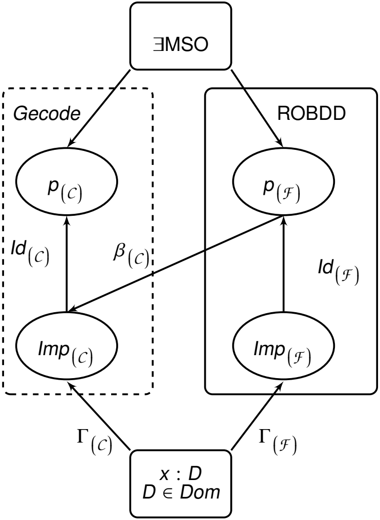
# Contributions



# Contributions

1. Implemented presented concepts in *Gecode*
   1. ROBDD set component
   2. Simulation $(\mathcal{C})$ with $(\mathcal{F})$ using view $\Gamma_{(\mathcal{C})}$
   3. Propagation across approximations using $\beta_{(\mathcal{C})}$
   4. Also implemented:
      1. $\beta_{(\mathcal{S})}$ for proper set bounds
      2. $\beta_{(\mathcal{L})}$ for lexicographic bounds
2. First framework to connect different implementations for set variables via variable views.
3. Prototype for generating set propagators from ∃MSO as uniform specification language [TSS06]

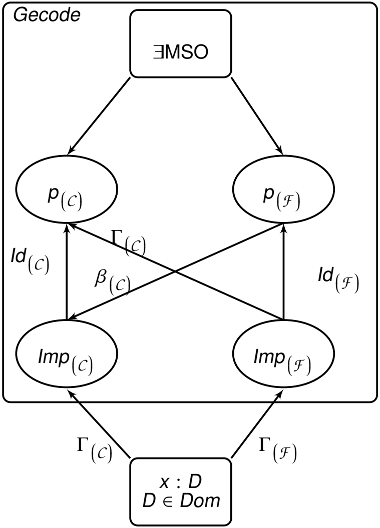
# Contributions



# Contributions

1. Implemented presented concepts in *Gecode*
   1. ROBDD set component
   2. Simulation $\left(\mathcal{C}\right)$ with $\left(\mathcal{F}\right)$ using view $\Gamma_{(\mathcal{C})}$
   3. Propagation across approximations using $\beta_{(\mathcal{C})}$
   4. Also implemented:
      1. $\beta_{(\mathcal{S})}$ for proper set bounds
      2. $\beta_{(\mathcal{L})}$ for lexicographic bounds
2. First framework to connect different implementations for set variables via variable views.
3. Prototype for generating set propagators from uniform specification language
4. Introduce different implementation for $\left(\mathcal{C}\right)$
   - compared different implementations for $\left(\mathcal{C}\right)$

# Contributions - Completing the picture



# Outlook and future work

- Generalize results to multisets by introducing
  1. approximations
  2. views
  3. constraints
- Comparison of used data structures with different data structures for example:
  1. Bit vectors

# References I

[AB00] Francisco Azevedo and Pedro Barahona.
Applications of an extended set constraint solver, 2000.

[Ben96] Frédéric Benhamou.
Heterogeneous constraint solving.
In Michael Hanus and Mario Rodríguez-Artalejo, editors, *ALP*, volume 1139 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 1996.

[Ger94] Carmen Gervet.
Conjunto: constraint logic programming with finite set domains.
In Maurice Bruynooghe, editor, *Logic Programming - Proceedings of the 1994 International Symposium*, pages 339–358, Massachusetts Institute of Technology, 1994. The MIT Press.

# References II


[Ger95] Carmen Gervet.
*Set Intervals in Constraint Logic Programming*.
PhD thesis, L'Université de Franche-Comté, 1995.

[HLS04] Peter Hawkins, Vitaly Lagoon, and Peter J. Stuckey.
Set bounds and (split) set domain propagation using ROBDDs.
In Geoffrey I. Webb and Xinghuo Yu, editors, *Australian Conference on Artificial Intelligence*, volume 3339 of *Lecture Notes in Computer Science*, pages 706–717. Springer, 2004.

[ILO00] ILOG Inc., Mountain View, CA, USA.
*ILOG Solver 5.0 reference Manual*, 2000.

# References III


[LS04] Vitaly Lagoon and Peter J. Stuckey.
Set domain propagation using ROBDDs.
In Mark Wallace, editor, *CP*, volume 3258 of *Lecture Notes in Computer Science*, pages 347–361. Springer, 2004.

[Pug92] Jean-Francois Puget.
Pecos a high level constraint programming language.
In *Singapore International Conference on Intelligent Systems (SPICIS)*, September 1992.

[ST06] Christian Schulte and Guido Tack.
Views and iterators for generic constraint implementations.
In Mats Carlsson, Francois Fages, Brahim Hnich, and Francesca Rossi, editors, *Recent Advances in Constraints, 2005*, volume 3978 of *Lecture Notes in Computer Science*, pages 118–132. Springer, 2006.

# References IV

[The06a] The Gecode team.
Generic constraint development environment.
Available from http://www.gecode.org, 2006.

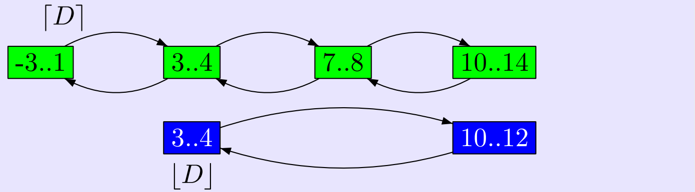
[The06b] The Mozart Consortium.
The Mozart programming system.
http://www.mozart-oz.org, 2006.

[TSS06] Guido Tack, Christian Schulte, and Gert Smolka.
Generating propagators for finite set constraints.
In Fréderic Benhamou, editor, *12th International Conference on Principles and Practice of Constraint Programming*, volume 4204 of *Lecture Notes in Computer Science*, pages 575–589. Springer, 2006.

# Sets in Gecode

## Representation of finite integer sets

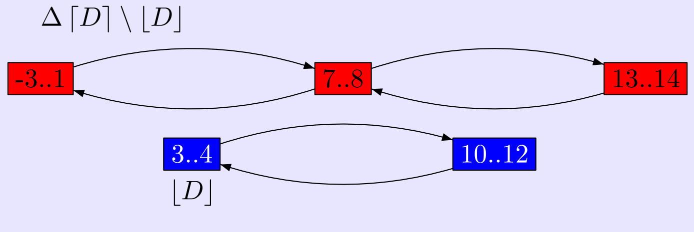
- bounds representation of domain $D$ by $[\lfloor D \rfloor .. \lceil D \rceil]$
- each bound represented by a range list
- $D = [\{3, 4, 10, 11, 12\}..\{-3, -2, -1, 0, 1, 3, 4, 7, 8, 10, 11, 12, 13, 14\}]$



# Sets in Gecode

## Representation of finite integer sets

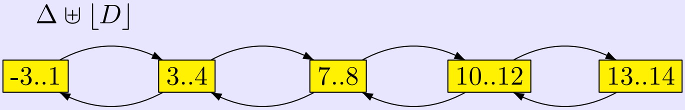
- remove $\lfloor D \rfloor$ from $\lceil D \rceil$
- obtain $\Delta(D) = \lceil D \rceil \setminus \lfloor D \rfloor$



# Sets in Gecode

## Minimal bounds representation [AB00]

- minimal bounds rep
- store disjoint union of $\Delta(D)$ and $\lfloor D \rfloor$ such that $\Delta(D) \uplus \lfloor D \rfloor = \lceil D \rceil$



# Fragment of ∃MSO

## Fragment

$$\begin{array}{lcl} S & ::= & \exists x.\langle S\rangle \mid \langle F\rangle \\ F & ::= & \forall v.\langle B\rangle \mid \exists v.\langle B\rangle \mid \langle F\rangle \wedge \langle F\rangle \\ B & ::= & \langle B\rangle \wedge \langle B\rangle \mid \langle B\rangle \vee \langle B\rangle \mid \neg\langle B\rangle \mid v \in x \in \mathtt{Var} \mid \perp \end{array}$$

## Example

Express constraint in ∃MSO

- constraint $c \equiv x \cap y = z$
- ∃MSO-formula $\phi_c = \forall v.v \in x \wedge y \in y \Leftrightarrow v \in z$