Diploma Thesis: 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

21.03.2007

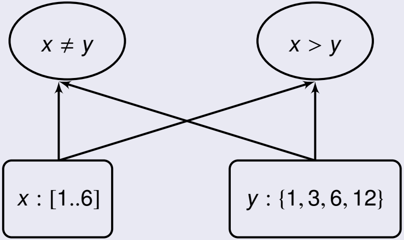
In the next 30 minutes...

Main aspects

- practical focus
- software architectural point of view
- data structures and algorithms
- research area: constraint programming

Constraint Programming - A quick reminder

Essential components



propagators implementing constraints on the variables

problem variables

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $x_{ij} : \{1, \ldots, 9\}$

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $x_{ij} : \{1, \ldots, 9\}$
- variable $x_{28} : \{1\}$, etc.

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $x_{ij} : \{1, \ldots, 9\}$
- variable $x_{28} : \{1\}$, etc.
- alldifferent row r_i

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $x_{ij} : \{1, \ldots, 9\}$
- variable $x_{28} : \{1\}$, etc.
- alldifferent row r_i
- alldifferent col c_i

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $x_{ij} : \{1, \ldots, 9\}$
- variable $x_{28} : \{1\}$, etc.
- alldifferent row r_i
- alldifferent col c_i
- alldifferent 3×3-block b_i

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $x_{ij} : \{1, \ldots, 9\}$
- variable $x_{28} : \{1\}$, etc.
- alldifferent row r_i
- alldifferent col c_i
- alldifferent 3×3-block b_i

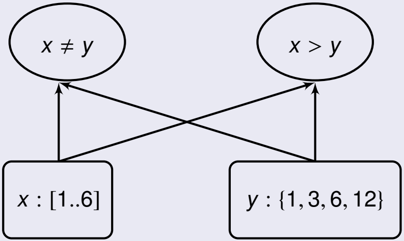
Finite domain set variables (SetVar)

When to use them ?

- reduce number of variables
- focus on collection of elements
- avoid symmetries
  - students in tutorial groups
  - players in a team
  - workers at a shift

Constraint Programming - A quick reminder

Essential components

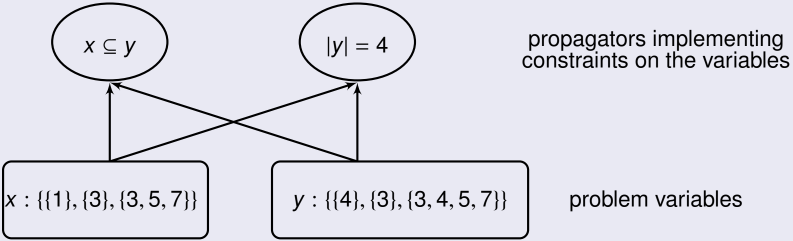


propagators implementing constraints on the variables

problem variables

Constraint Programming - A quick reminder

Essential components



Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $y_i : \{1, \ldots, 9^2\}$

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $y_i : \{1, \ldots, 9^2\}$
- variable $y_1 : \left[\{17, 56, 76\}..\{1, \ldots, 9^2\}\right]$
  $|y_j| = 9$

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $y_i : \{1, \ldots, 9^2\}$
- variable $y_1 : \left[\{17, 56, 76\}..\{1, \ldots, 9^2\}\right]$
  $|y_j| = 9$
- row: $|y_j \cap r_i| \leq 1$

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $y_i : \{1, \ldots, 9^2\}$
- variable $y_1 : \left[\{17, 56, 76\}..\{1, \ldots, 9^2\}\right]$
  $\left|y_j\right| = 9$
- row: $\left|y_j \cap r_i\right| \leq 1$
- column: $\left|y_j \cap c_i\right| \leq 1$

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $y_i : \{1, \ldots, 9^2\}$
- variable $y_1 : \left[\{17, 56, 76\}..\{1, \ldots, 9^2\}\right]$
  $|y_j| = 9$
- row: $|y_j \cap r_i| \leq 1$
- column: $|y_j \cap c_i| \leq 1$
- 3×3-block: $|y_j \cap b_i| \leq 1$

Constraint Programming - Motivation

					3		6	
							1	
	9	7					8	
				9		2		
		8		7		4		
		3		6				
	1				2	8	9	
	4							
	5		1					

- variable $y_i : \{1, \ldots, 9^2\}$
- variable $y_1 : \left[\{17, 56, 76\}..\{1, \ldots, 9^2\}\right]$
  $\left|y_j\right| = 9$
- row: $\left|y_j \cap r_i\right| \leq 1$
- column: $\left|y_j \cap c_i\right| \leq 1$
- 3×3-block: $\left|y_j \cap b_i\right| \leq 1$

Framework

Gecode Constraint Library

- **ge**neric
- **co**nstraint
- **d**evelopment
- **e**nvironment

Gecode [The06], 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)

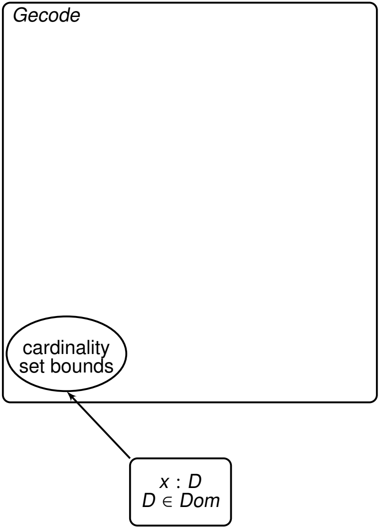
Available architecture in *Gecode*

Gecode

Gecode set solver

$x : D$
$D \in Dom$

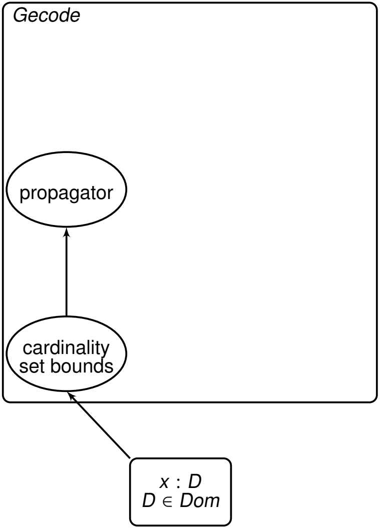
Available architecture in *Gecode*



Gecode set solver

- Representation of set variables: *Cardinality set bounds*

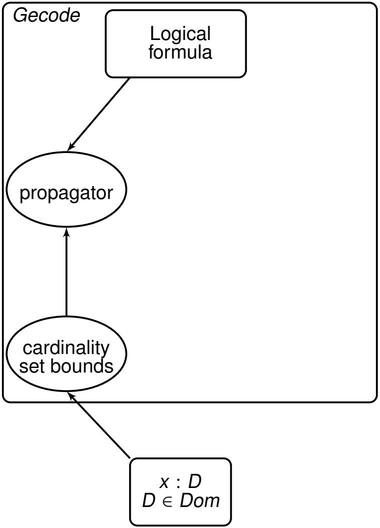
Available architecture in *Gecode*



Gecode set solver

- Representation of set variables: *Cardinality set bounds*
- Propagators for this representation

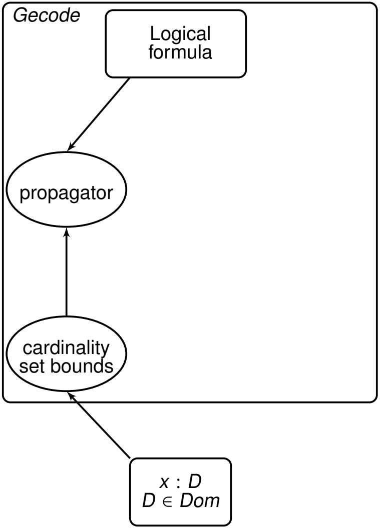
Available architecture in *Gecode*



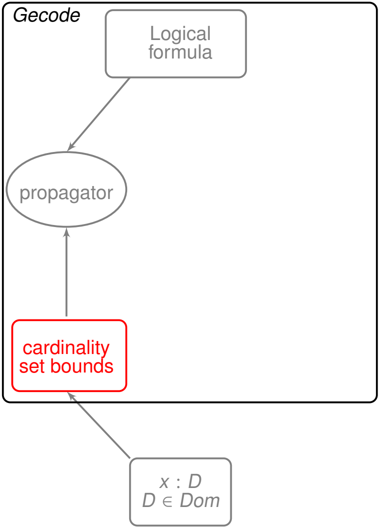
Gecode set solver

- Representation of set variables: *Cardinality set bounds*
- Propagators for this representation
- Automated generator using logical formulae

Contributions



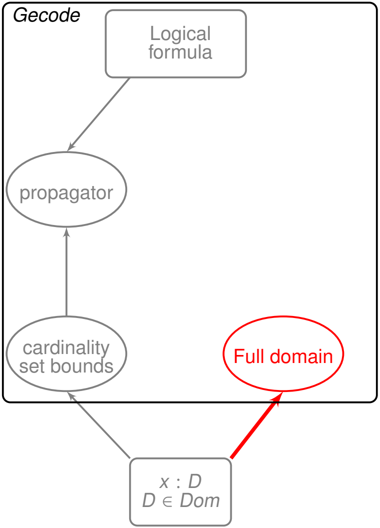
Contributions



Extending the *Gecode* set solver

- Compare different data structure for *Cardinality set bounds* with *Gecode* data structure

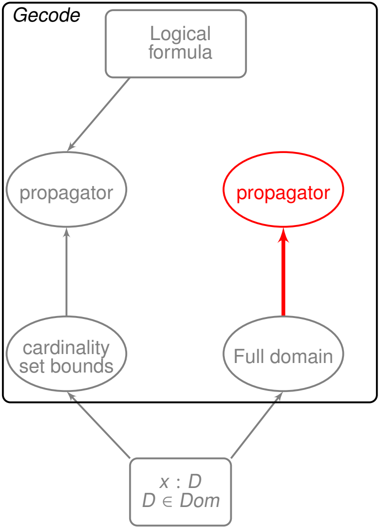
Contributions



Extending the *Gecode* set solver

- Implemented:
  - different representation: *Full domain*

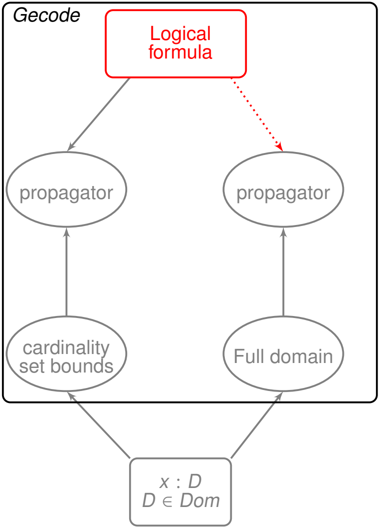
Contributions



Extending the *Gecode* set solver

- Implemented:
    - different representation: *Full domain*
    - propagators for this representation

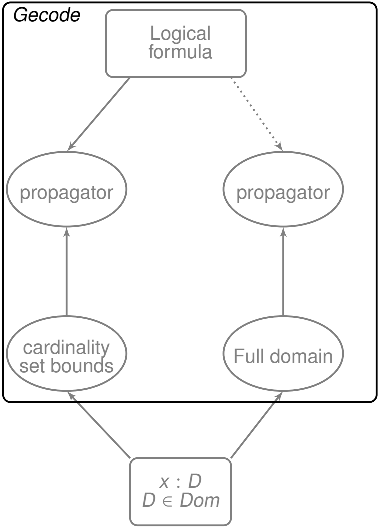
Contributions



Extending the *Gecode* set solver

- Prototypical extension for automated propagator generation

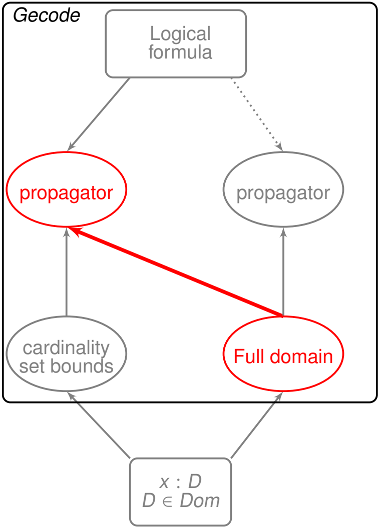
Contributions



Extending the *Gecode* set solver

- Prototypical extension for automated propagator generation
- interfaces for system integration:

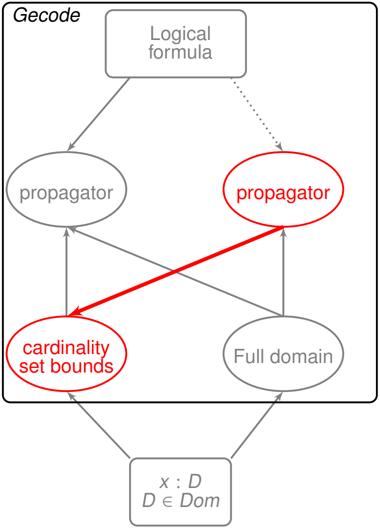
Contributions



Extending the *Gecode* set solver

- Prototypical extension for automated propagator generation
- interfaces for system integration:
    - Simulation of other data structures

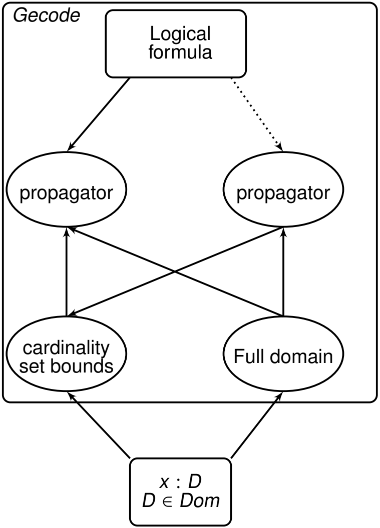
Contributions



Extending the *Gecode* set solver

- Prototypical extension for automated propagator generation
- interfaces for system integration:
  - Simulation of other data structures
  - Simulation of weaker propagation

Contributions



Extending the *Gecode* set solver

- Prototypical extension for automated propagator generation
- interfaces for system integration:
    - Simulation of other data structures
    - Simulation of weaker propagation

Central question

Practical implementation

- How to represent a set variable in the system
- What data structures to use

Central question

Practical implementation

- How to represent a set variable in the system
- What data structures to use

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

Theoretical model - Domain approximation

Theoretical foundations

- introduced by Benhamou[Ben96]
- model all available domain representations for constraint variables

Theoretical model - Domain approximation

Idea: Domain Approximation $\mathcal{A}$

- representative subset ($\mathcal{A} \subseteq Dom$)
- closed under intersection ($\forall A, B \in \mathcal{A} : (A \cap B) \in \mathcal{A}$)
- Elements of $\mathcal{A}$: *approximate domains*

Required elements

$\emptyset$	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

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

Set bounds approximation

Theoretical foundations

- Puget in [Pug92]
  - First introduced it in constraint programming
- Gervet in [Ger95, chp. 4]
  - Described it in full detail
  - *Conjunto* [Ger94] as reference implementation

Set bounds approximation

Convex set bounds

- $\left(\mathcal{S}\right) = \{T \in Dom \mid \inf(D) \subseteq T \subseteq \sup(D)\}$

Example

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

Set bounds approximation

Convex set bounds

- $\left(\mathcal{S}\right) = \{T \in Dom \mid \inf(D) \subseteq T \subseteq \sup(D)\}$
- $E \in \left(\mathcal{S}\right)$ smallest convex interval containing D (w.r.t. $\subseteq$)

Example

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

Set bounds approximation

Convex set bounds

- $\left(\mathcal{S}\right) = \{T \in Dom \mid \inf(D) \subseteq T \subseteq \sup(D)\}$
- $E \in \left(\mathcal{S}\right)$ smallest convex interval containing D (w.r.t. $\subseteq$)
- $E = \left[\bigcap_{d \in D} d .. \bigcup_{d \in D} d\right]_{\subseteq}$

Example

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

Set bounds approximation - Pros and Cons

Pros $(\mathcal{S})$

- guaranteed linear size
- space efficiency:
  - only two sets $\lfloor E \rfloor$, $\lceil E \rceil$ instead of exponentially many
- *extension* property (Gervet[Ger95]):
  - 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 - Pros and Cons

Pros $(\mathcal{S})$

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

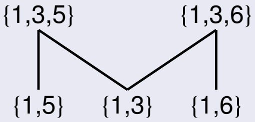
$$\begin{array}{lcl} \forall v \in \lfloor E \rfloor & \Rightarrow & v \in \alpha(x) \\ \forall v \notin \lceil E \rceil & \Rightarrow & v \notin \alpha(x) \end{array}$$

Cons $(\mathcal{S})$

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

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

Hesse-Diagram



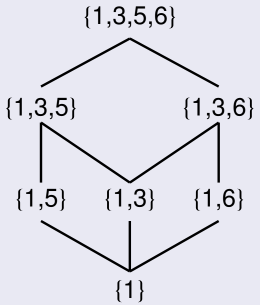
From *Dom* to $\left(\mathcal{C}\right)$

Set variable $x : D$

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

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

Hesse-Diagram



From *Dom* to $(\mathcal{C})$

Set variable $x : D$

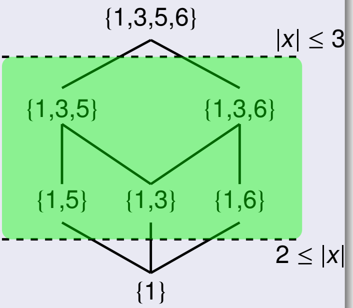
$$D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\},\{1,3,6\}\}$$

Set bounds

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

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

Hesse-Diagram



From *Dom* to $\left(\mathcal{C}\right)$

Set variable $x : D$

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

Set bounds

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

Adding cardinality requirements

$$F = E \cap \left\{T \in \left(\mathcal{S}\right) \mid 2 \leq \lfloor T \rfloor \wedge \lceil T \rceil \leq 3\right\}$$
$$= D$$

Full domain approximation $(\mathcal{F})$

From Dom to $(\mathcal{F})$

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

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

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

- exact representation of the complete domain
- stronger propagation

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

Pros $(\mathcal{F})$

- exact representation of the complete domain
- stronger propagation

Cons $(\mathcal{F})$

- space efficiency depends on data structure implementing formula F
- worst case exponential size

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

Theoretical foundations

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

Reduced Ordered Binary Decision Diagrams (ROBDDS)

Short overview

R educed: no identical nodes

O rdered: respect specified variable order $\prec$

B inary Decision Diagram:
well-known method of modeling Boolean functions
on Boolean variables

Reduced Ordered Binary Decision Diagrams (ROBDDS)

Short overview

- R educed: no identical nodes
- O rdered: respect specified variable order $\prec$
- B inary Decision Diagram:
  well-known method of modeling Boolean functions
  on Boolean variables

ROBDD

- canonical function representation up to reordering
- permits efficient implementation of Boolean function operations

Represent SetVar in full domain approximation

- $D \in \left(\mathcal{F}\right)$ represented as tuple $\langle b, F \rangle$
  1. Boolean vector $b = \langle b_{\min(\lceil D \rceil)}, \ldots, b_{\max(\lceil D \rceil)} \rangle$
  2. represent $d_i \in D$ as formula $f(d_i) = \bigwedge_{j=1}^{|\lceil D \rceil|} a_i \quad a_i = \begin{cases} b_j & \text{if } j \in d_i \\ \neg b_j & \text{else} \end{cases}$
  3. represent D as formula $F = \bigvee_{d_i \in D} f(d_i)$

Reduced Ordered Binary Decision Diagrams (ROBDDS)

- $D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\}\}$
- create Boolean vector
  $b = \langle b_1, b_2, b_3, b_4, b_5, b_6 \rangle$
- resulting formula F

$$F = \bigvee_{d_i \in D} f(d_i)$$

Reduced Ordered Binary Decision Diagrams (ROBDDS)

- $D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\}\}$
- create Boolean vector
  $b = \langle b_1, b_2, b_3, b_4, b_5, b_6 \rangle$
- resulting formula F

$$
\begin{aligned}
F &= f(\{1,3\}) \\
&\vee f(\{1,5\}) \\
&\vee f(\{1,6\}) \\
&\vee f(\{1,3,5\})
\end{aligned}
$$

Reduced Ordered Binary Decision Diagrams (ROBDDS)

- $D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\}\}$
- create Boolean vector
  $b = \langle b_1, b_2, b_3, b_4, b_5, b_6 \rangle$
- resulting formula F

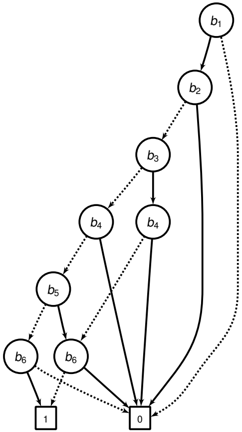
$$
\begin{aligned}
F \; = & \; b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge \neg b_6 \\
\vee & \; b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge b_5 \wedge \neg b_6 \\
\vee & \; b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge b_6 \\
\vee & \; b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge b_5 \wedge \neg b_6
\end{aligned}
$$

Reduced Ordered Binary Decision Diagrams (ROBDDS)

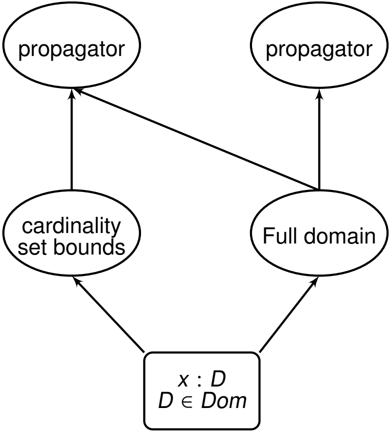
- $D = \{\{1,3\},\{1,5\},\{1,6\},\{1,3,5\}\}$
- create Boolean vector $b = \langle b_1, b_2, b_3, b_4, b_5, b_6 \rangle$
- resulting formula F

$$\begin{aligned} F &= b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge \neg b_6 \\ &\vee b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge b_5 \wedge \neg b_6 \\ &\vee b_1 \wedge \neg b_2 \wedge \neg b_3 \wedge \neg b_4 \wedge \neg b_5 \wedge b_6 \\ &\vee b_1 \wedge \neg b_2 \wedge b_3 \wedge \neg b_4 \wedge b_5 \wedge \neg b_6 \end{aligned}$$

- *ROBDD* representing all valuations of formula F

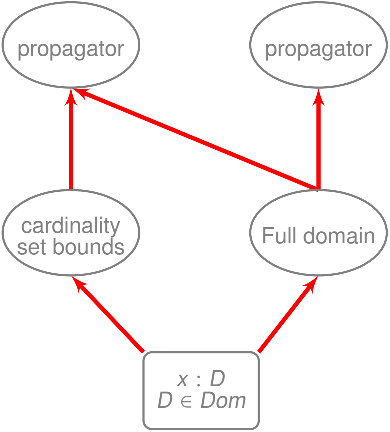


Variable Views as interface



Variable Views [ST06]

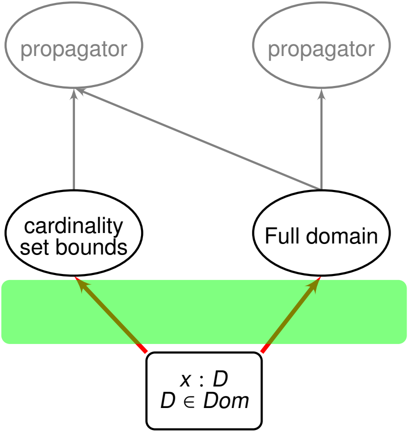
Variable Views as interface



Variable Views [ST06]

1. Mapping $V : \mathcal{A} \rightarrow \mathcal{B}$

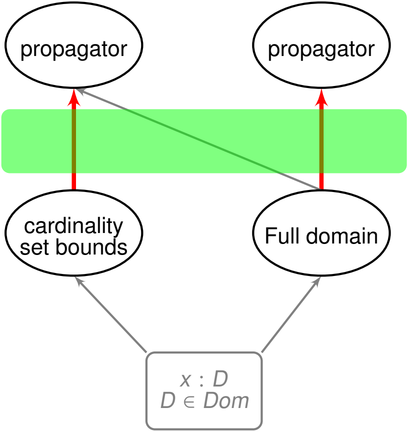
Variable Views as interface



Variable Views [ST06]

1. Mapping $V : \mathcal{A} \rightarrow \mathcal{B}$
2. Adaptor for $\mathcal{A}$, $V_{\mathcal{A}} : Dom \rightarrow \mathcal{A}$
   - map $D \in Dom$ to $A \in \mathcal{A}$
   - prescribe internal representation (data structures)

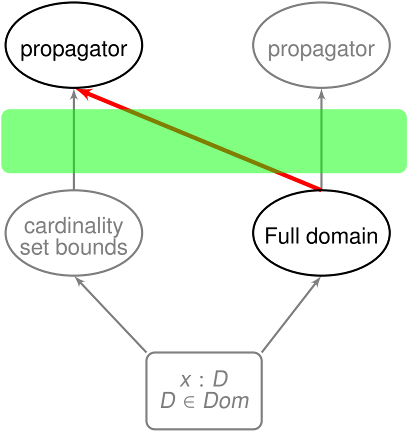
Variable Views as interface



Variable Views [ST06]

1. Mapping $V : \mathcal{A} \rightarrow \mathcal{B}$
2. Adaptor for $\mathcal{A}$, $V_{\mathcal{A}} : Dom \rightarrow \mathcal{A}$
   - map $D \in Dom$ to $A \in \mathcal{A}$
   - prescribe internal representation (data structures)
3. Propagation interface providing propagation services
   - domain lookup
   - domain update

Variable Views as interface



Variable Views [ST06]

1. Mapping $V : \mathcal{A} \to \mathcal{B}$
2. Adaptor for $\mathcal{A}$, $V_{\mathcal{A}} : Dom \to \mathcal{A}$
   - 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 approximations

Using views to connect approximations

Adaptor functionality

- $x : D = \{\{1\}, \{1, 3\}, \{1, 2, 4\}\}$

Using views to connect approximations

Adaptor functionality

- $x : D = \{\{1\}, \{1, 3\}, \{1, 2, 4\}\}$
- Set bounds view $V_{(\mathcal{S})} : Dom \rightarrow (\mathcal{S})$
  $V_{(\mathcal{S})}(D) = [\{1\}..\{1, 2, 3, 4\}]_{\subseteq}$

Using views to connect approximations

Adaptor functionality

- $x : D = \{\{1\}, \{1,3\}, \{1,2,4\}\}$
- Set bounds view $V_{(\mathcal{S})} : Dom \to (\mathcal{S})$
  $V_{(\mathcal{S})}(D) = [\{1\}..\{1,2,3,4\}]_{\subseteq}$
- Cardinality set bounds view $V_{(\mathcal{C})} : Dom \to (\mathcal{C})$

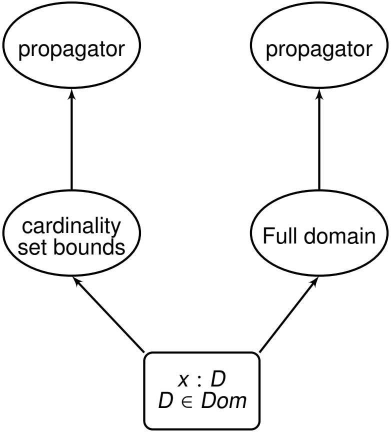
1. add cardinality constraints: $2 \leq |x| \leq 3$
2. $$\begin{aligned} V_{(\mathcal{C})}(D) &= V_{(\mathcal{S})}(D) \cap \left\{T \in (\mathcal{S}) \mid 2 \leq |\lfloor T \rfloor| \wedge |\lceil T \rceil| \leq 3\right\} \\ &= \{\{1,3\}, \{1,2,4\}\} \end{aligned}$$

Using views to connect approximations

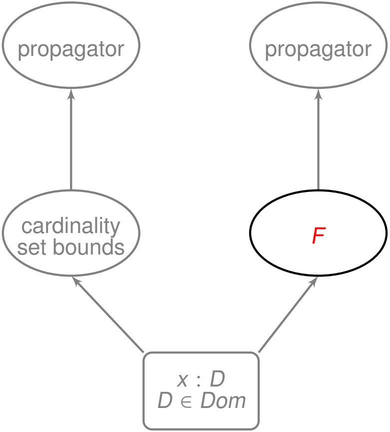
Adaptor functionality

- $x : D = \{\{1\},\{1,3\},\{1,2,4\}\}$
- Set bounds view $V_{(\mathcal{S})} : Dom \rightarrow (\mathcal{S})$
  $V_{(\mathcal{S})}(D) = [\{1\}..\{1,2,3,4\}]_{\subseteq}$
- Cardinality set bounds view $V_{(\mathcal{C})} : Dom \rightarrow (\mathcal{C})$
  1. add cardinality constraints: $2 \leq |x| \leq 3$
  2. $$\begin{aligned} V_{(\mathcal{C})}(D) &= V_{(\mathcal{S})}(D) \cap \left\{T \in (\mathcal{S}) \mid 2 \leq \|\lfloor T \rfloor\| \wedge \|\lceil T \rceil\| \leq 3\right\} \\ &= \{\{1,3\},\{1,2,4\}\} \end{aligned}$$
- Full domain view $V_{(\mathcal{F})} : Dom \rightarrow (\mathcal{F})$
  $\Gamma_{(\mathcal{F})}(D) = D$

Weaken propagation

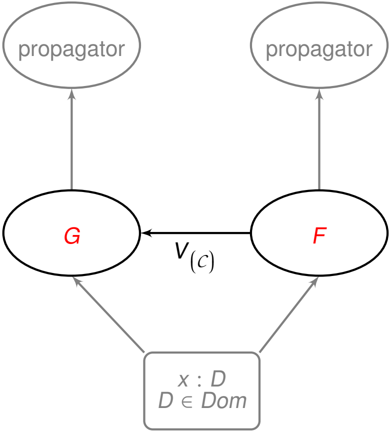


Weaken propagation



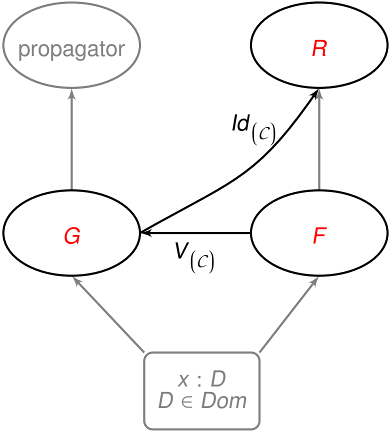
1. $F \in \left(\mathcal{F}\right)$ is x-component of domain tuple $\vec{F}$, $\vec{F}.x = F$

Weaken propagation



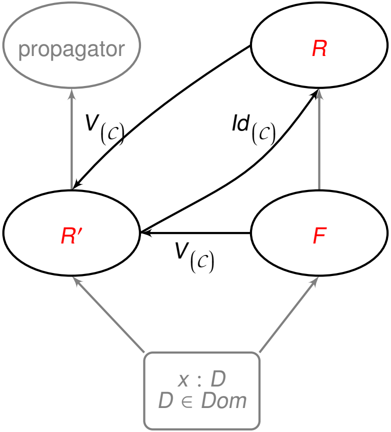
1. $F \in \left(\mathcal{F}\right)$ is x-component of domain tuple $\vec{F}$, $\vec{F}.x = F$
2. Map F to the respective cardinality set bounds $G = V_{(\mathcal{C})}(F)$

Weaken propagation



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

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 $G = V_{(\mathcal{C})}(F)$
3. Since $G \in (\mathcal{C}) \subset (\mathcal{F})$ apply $P_{(\mathcal{F})} : (\mathcal{F})^n \to (\mathcal{F})^n$ Propagation result $R = P_{(\mathcal{F})}(\vec{G}).x$
4. Map result R again to $R' = V_{(\mathcal{C})}(R)$.

Using variable views to weaken propagation

Changing consistency of propagation result

- Use a domain-consistent propagator

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

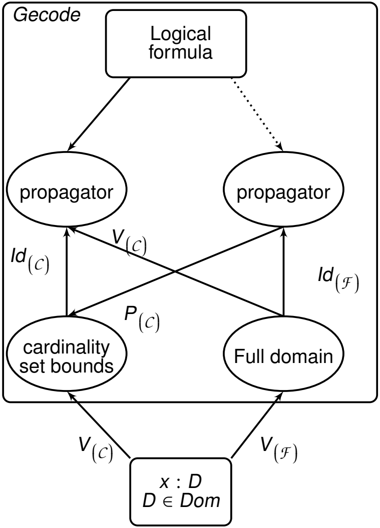
- obtain bounds $(\mathcal{C})$-consistent propagator
  $P_{(\mathcal{C})} : (\mathcal{F})^n \to (\mathcal{C})^n$
  $P_{(\mathcal{C})}\left(\vec{F}\right) = V_{(\mathcal{C})}\left(P_{(\mathcal{F})}\left(V_{(\mathcal{C})}\left(\vec{F}\right)\right)\right)$

Contributions

Summary

1. Implemented presented concepts in *Gecode*
   1. ROBDD set component
   2. Simulation $(\mathcal{C})$ with $(\mathcal{F})$ using view $V_{(\mathcal{C})}$
   3. Propagation across approximations using $P_{(\mathcal{C})}$
   4. Also implemented:
      1. $P_{(\mathcal{S})}$ for proper set bounds
      2. $P_{(\mathcal{L})}$ 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 [TSS06]
4. Implemented and compared different implementations for $(\mathcal{C})$

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
- Finish automated propagator generation for ROBDD component

Thanks

Thank you for your attention!

Questions

Are there any questions?

References I

[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.

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

References II

[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.

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

References III

[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.

[The06] The Gecode team.
Generic constraint development environment.
Available from `http://www.gecode.org`, 2006.

References IV

[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.