Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science
Programming Systems Chair

Diploma Thesis

Domain approximations for finite set constraint variables An integrated approach

submitted by

Patrick Pekczynski

on

31.01.2007

Supervisor:
Prof. Dr. Gert Smolka

Advisor:
Dipl.-Inf. Guido Tack

Reviewers:
Prof. Dr. Gert Smolka
Prof. Dr. Kurt Mehlhorn

Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, 29.04.2007

Patrick Pekczynski

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbrücken, 29.04.2007

Patrick Pekczynski

Ne nega illa!

Abstract

Constraint programming (CP) is a powerful state-of-the-art approach to solve hard combinatorial problems. To achieve this CP needs to represent *constraint variables*, their associated *domains* and *propagators* encoding the respective *constraints* on those variables in a constraint solver. This thesis presents the application, implementation and empirical evaluation of different representations for finite set constraint variables, their respective domains and finite set propagators on them. The corresponding implementation using the *Gecode* C++-library [39] is the first to fully integrate a convex cardinality set bounds representation provided by the *Gecode* -library and the ROBDD-based complete domain representation proposed by Hawkins *et al.*[22] in a single constraint solver. Presenting the concepts of *domain approximation*, *variable views* and *propagator generation* and applying them to the computation domain of finite sets this thesis allows a free combination of both variable representations and their respective propagators across the boundaries of each variable representation.

Acknowledgement

First of all I want to thank professor Gert Smolka, who encouraged us after the “Seminar für Programmiersprachen” to do a FoPra thesis (Fortgeschrittenenpraktikum) [27] at the Programming Systems Lab. It finally was my FoPra thesis about the implementation of advanced propagation algorithms for global constrains that led me to my diploma thesis at the same chair.

Especially, I want to thank Professor Kurt Mehlhorn for spending some of his precious time on doing the second revision of this thesis and I hope that I did not waste it.

I am deeply indebted to my advisor, Guido Tack, who with his seemingly never-ending patience already helped my through my FoPra and now has guided me safely through the dangerous paths of huge search tress, always helping me out of all the failed nodes I have encountered during this thesis. He led me safely back on the path to this hopefully feasible and satisfiable solution. I also want to thank all the other members of the Programming Systems Lab who not only created a very nice working atmosphere but also provide help and support whenever needed.

Moreover I thank professor Smolka and Dr. Christian Schulte, head of the *Gecode* project [39], to give me the chance of being a part of this project, which provided the base for all the implementations during my diploma thesis.

I am also grateful to Mats Carlsson who encouraged me to “keep up with the good work”, what I have hopefully done, to Peter Stuckey who helped me out with some implementation hints with respect to the ROBDD implementation and to François Puget for providing me some of his reports about set representation I could not retrieve elsewhere and for giving me some hints on set representation.

Last but not least I am deeply indebted to the love and support of my family and friends without whose help this thesis could have never been written. Especially, I thank my parents Dieter and Ursula, my sister Nadja as well as my friends Christina, Mirko and Aline whose nerves I unduly stressed whilst writing this thesis. I tried my best such that it has not been in vain.

Contents

1 Introduction

Constraint programming (CP) is a powerful state-of-the-art approach to solve hard combinatorial problems. To achieve this CP needs to represent *constraint variables*, their associated *domains* and *propagators* encoding the respective *constraints* on those variables in a constraint solver. This thesis presents the application, implementation and empirical evaluation of different representations for finite set constraint variables, their respective domains and finite set propagators on them. The corresponding implementation using the *Gecode* C++-library [39] is the first to fully integrate a convex cardinality set bounds representation provided by the *Gecode* -library and the ROBDD-based complete domain representation proposed by Hawkins *et al.*[22] in a single constraint solver. Presenting the concepts of *domain approximation*, *variable views* and *propagator generation* and applying them to the computation domain of finite sets this thesis allows a free combination of both variable representations and their respective propagators across the boundaries of each variable representation.

1.1 From integer to set variables

The essential purpose of constraint programming is the formulation and the solution of specified problems involving problem variables and constraints on them[4] where the variables take their values in problem-specific domains. In this context most of those problems arising in different fields such as combinatorial mathematics, scheduling or operations research require the problem variables to take finite integers as possible values. Those variables are referred to as *finite domain integer variables* or simply *integer variables*. Nevertheless, Gervet emphasizes in [17] that the use of integer variables has its limitations if it comes to real-world combinatorial problems "based on the search for sets or mapping objects"[17].

Finite sets as computation domain In this context Puget considers the computation domain of finite sets over integers as "very useful" [28], that is a finite set of finite sets of integers. In contrast to the integer variables ranging over a single set of finite integers, those variables are referred to as *finite domain set variables* or *set variables*.

New possibilities As witnessed by Gervet in [18] and a dedicated workshop on behalf of "Beyond finite domain programming"[7] this usefulness results in the availability of set constraint solvers based on finite sets as computation domain. This in turn leads to the design of new models and solutions to problems from combinatorial mathematics [8], VLSI circuit verification and warehouse location [5]. Even more so, there are classes of real-world problems, where the use of set variables instead of integer variables remarkably reduces the total number of variables in a problem. For instance Puget points out in [29, sect.5] that modeling a schedule of 3000 crews for an airline with a pool of 2000 people available takes only 3000 set variables instead of $6 \cdot 10^6$ Boolean variables. Apart from reducing the number of problem variables, the use of set variables can also help to eliminate symmetries in the problem specification. For instance Frisch *et al.* indicate in [13] that the use of set variables removes a structural symmetry in scheduling problems like the *Social Golfer* problem.

A new problem However, Puget indicates in [28] that representing set variables in a constraint solver is problematic because the domain size of those variables easily becomes exponential. Since the typical design of a constraint programming system involves both, variables and constraint on them, we have to reassure that the time spent on operations on the data structure implementing a set variable representation does not cut the above mentioned advantages of set variables. Consequently Frisch and Jefferson conclude in [12] that the representation of set variables is a critical aspect in the design of a set constraint solver. This is exactly the point, that we investigate in the remaining chapters of this thesis.

1.2 Contributions

Based on the theoretical background of finite domain constraint programming over set variables this thesis aims at the following goals:

Setting the stage We apply Benhamou's framework of domain approximation to the domain of finite set variables such that all representations for finite set variables in literature can be encoded in this framework. Moreover, we show formally what constraint propagation in this framework means and how constraint propagation is applied on approximate domains as elements of the respective domain approximations under consideration. Based on this framework this thesis presents recent approaches in literature to represent finite set variables like the standard set bounds approximation (Puget[28], Gervet[16]), the set bounds approximation augmented with cardinality information (Gervet, Azevedo [15, 5]) a lexicographic bounds approach (Sadler and Gervet[30]) as well as a full approximation using ROBDDs (Hawkins *et al.*[22]) and encodes them in terms of domain approximations.

Getting practical As this thesis has a practical orientation one main task consists in comparing the set bounds domain approximation with cardinality information as presented in the *Gecode* -library against a different implementation for the same approximation. Another task was the implementation of a ROBDD-based solver in *Gecode* based on the full domain approximation as Hawkins *et al.* presents in [22]. In this context, this thesis discusses how the concept of variable views that Schulte and Tack introduce in [35] can be used to integrate this implementation into an already existing set solver. As practical instance we choose the *Gecode* -library[39] that already provides the set bounds domain approximation augmented with cardinality information. Further, we show how variable views allow the implementation of different consistency techniques presented in [22] with only one domain approximation. Moreover we show how variable views change the orthogonal implementation of two different set solvers into a single generic interface that is able to cross domain approximations and consistency levels. Thus this thesis is the first to integrate two completely different variable representations into one set constraint solver providing the possibility of representation comprehensive constraint propagation. In addition to the abstraction of domain approximations we also apply the concept of a formal modeling language for set constraints that Tack *et al.* present in [37] to implement a prototype of an intensional set constraint representation. This representation not only allows us to translate constraints into propagators for the cardinality set bounds representation available in *Gecode* [39] but also to translate them into propagators for the ROBDD-based representation topping off the integration of two different set solvers into a single generic interface.

Empirical evaluation Apart from the introduction into the formal framework of domain approximations and the integration of the ROBDD-based set solver as new domain approximation into *Gecode* we also provide a thorough analysis and evaluation of the presented domain approximations, their underlying data structures, the variable views as propagation interface between the different domain approximations as wells as the propagators themselves working on those domain representations.

Programming framework The tasks as described in the preceding paragraphs have all been implemented using *Gecode* , a C++-based constraint programming library [39] that is jointly developed at the Royal Technical Institute of Technology in Stockholm (KTH) and the Programming System Chair (PS-LAB) at the Saarland University in Saarbrücken.

1.3 Related Work

As a major aspect of this thesis consists in the practical application and implementation of available frameworks and concepts, this section clarifies what scientific work in the area of finite domain constraint programming provides the fundamental theoretical background for the contributions of this thesis.

1.3.1 Domain Approximation

An essential point of the introductory paragraph is the size issue of a representation for set variables, that is the domain of a set variable possibly has exponential size.

Encode available variable representations In this context, a central aspect in this thesis is the approach of approximating a variable domain using the concept of a *domain approximation* as formalized by Benhamou in [9]. Provided this concept of domain approximations as central framework for our thesis, it allows us to encode the domain approximation used by most state-of-the-art constraint solvers supporting constraint solving over finite sets like *ILOG* [1], *ECL^{i}PS^e* [42], *Conjunto* [15], *Mozart* [40] and *Gecode* [39]. The finite set component of all these solvers essentially bases on the standard *set bounds approximation* scheme, that denotes a domain approximation involving lower and upper bounds on the domain of a set variable as developed by Puget in [28] and presented in detail in the dissertation of Gervet [16]. Additionally, Benhamou' s framework of domain approximations also allows us to talk about a "radically different approach"[18] developed by Lagoon and Stuckey who show in [24] how we can use reduced ordered binary decision diagrams (ROBDDs) to literally model the complete domain of a set variable.

Integration through variable views Apart from those two different domain approximations that represent state-of-the-art cornerstones in the area of set variable representation, we will furthermore concentrate on the concept of *variable views* or *views* as introduced by Schulte and Tack in [35] which yields our main tool to integrate the ROBDD domain representation into the *Gecode* -framework that already comes with an implementation of the above mentioned set bounds domain representation.

Generating propagators for finite set variables Concluding the central aspects of this thesis we will also take up the work of Tack *et al.* in [37] presenting an intensional representation of finite set constraints Puget aims at in [28]. In this context we discuss how they use this intensional representation to generate constraints in both domain approximations, the set bounds approximation and the full domain approximation.

1.4 Outline

The remaining chapters of this thesis are structured according to figure 1.1 providing an overview from variables to constraints. In chapter 2 we discuss the basic framework for constraint programming and recapitulate the theoretical background of domain approximations as introduced by [9]. Subsequently, chapter 3 presents an overview of the most popular domain approximations and discusses the implementation of the set bounds domain approximation as well as the ROBDD-based full domain approximation in the *Gecode* - framework. Chapter 4 introduces variable views for constraint propagation across different domain approximations. Chapter 5 deals with the questions how propagators on the presented domain approximations look like and how they can be generated using a uniform representation. Chapter 6 finally provides an empirical evaluation and discussion of the domain approximations in focus. Chapter 7 concludes the thesis and provides a short outlook on future work related to the aspect of set variable representations in constraint programming.

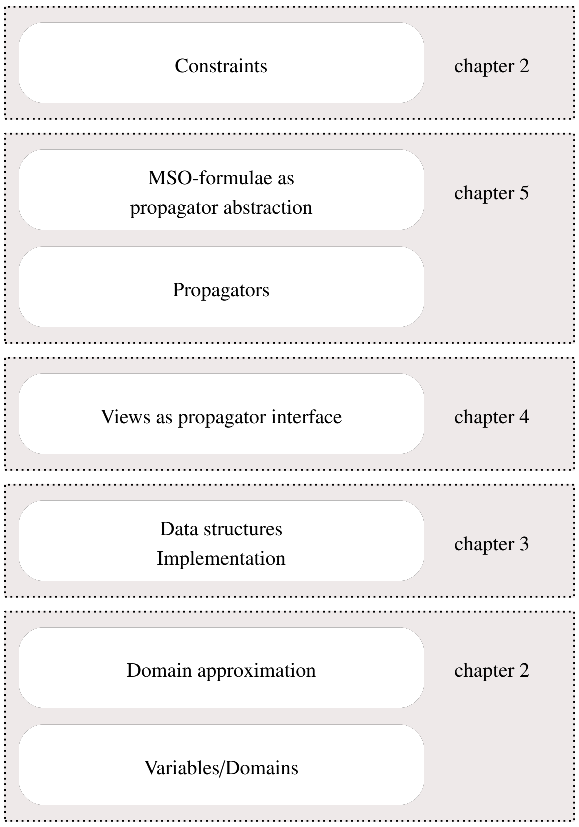

Figure 1.1: Constraint solver vertical cut

2 Constraint Programming - A Framework

THE STORY SO FAR: IN THE BEGINNING THE UNIVERSE WAS CREATED. THIS HAS MADE A LOT OF PEOPLE VERY ANGRY AND HAS BEEN WIDELY REGARDED AS A BAD MOVE [3].

— ADAMS

During the course of this chapter we take glance at *Constraint Programming (CP)* , providing the theoretical framework of this thesis. Moreover, we give a survey of the relevant definitions and notations used in the remaining chapters.

Constraint Satisfaction As Apt underlines in his “Rough Guide to Constraint Propagation”[4], “Constraint programming [..] consists of formulating and solving so-called constraint satisfaction problems”. Obviously, this slogan is twofold and indicates that CP is concerned with a modeling aspect (formulating) and a solution aspect (solving).

2.1 Modeling

In order to model a given problem as a *constraint satisfaction problem (CSP)* we have to state what a CSP consists of. Considering the formal definition of a CSP as provided by Tack *et al.* in [37] a CSP informally splits up into two essential components: a set of *variables* we denote with $\mathcal{X}$ and a set $\mathcal{C}$ of requirements on those variables, which are referred to as *constraints*.

2.1.1 Domains and Variables

According to [23] we assume that the set of variables $\mathcal{X}$ is typed and define:

Definition 1 (Universe and variable type). *Let $x \in \mathcal{X}$ be a constraint variable. We call a set* $\Sigma ::= \mathbb{B} \mid \mathbb{Z}$ *the* universe *of x and the set $\mathcal{T} ::= \Sigma \mid \mathcal{P}(\Sigma)$ the* type *of x, where $\mathcal{P}(\Sigma)$* denotes the power set over the universe Σ. Moreover we want to point out that the universe and hence the variable type are possibly (countable) infinite sets.

Apart from the variable type $\mathcal{T}$, each variable $x \in \mathcal{X}$ is also associated with a set $D \subset \mathcal{T}$, called its *domain*, specifying exactly what possible values the variable $x \in \mathcal{X}$ can be assigned to.

CP on finite domains Clearly, this understanding of a variable domain admits possibly infinite domains. Since we are only interested in variables associated with *finite* domains we restrict the term of CP for all remaining chapters to CP over *finite domains* denoted as CP(FD). Moreover, we also assume a finite set $\mathtt{Var} \subset \mathcal{X}$ of typed constraint variables and a finite universe $\mathcal{U} \subset \Sigma$. From the finiteness of $\mathcal{U}$ we immediately obtain a finite set of values $\mathtt{Val} \subset \mathcal{T}$ for those variables, where $\mathtt{Val} = \mathcal{U}$ if $\mathcal{T} = \Sigma$ and $\mathtt{Val} = \mathcal{P}(\mathcal{U})$ if $\mathcal{T} = \mathcal{P}(\Sigma)$. Finally we also obtain a finite set $Dom \subset \mathcal{P}(\mathtt{Val})$ denoting the power set of possible finite domains. Provided these restrictions, we are now able to define the first component of a constraint satisfaction problem, namely the set of constraint variables $\mathcal{X}$, more concisely.

Definition 2 (Finite domain variable)**.** *A variable $x \in \mathtt{Var}$ is called a* finite domain variable *if it is associated with a finite domain $D \subset \mathtt{Val}$. As D specifies which values x can take we say that x* ranges *over D, written $x : D$ or $x \in D$.*

Remark 1 (n-ary extension)**.** *Extending the notion of a finite domain to the n-ary cartesian product of n finite domains we write $\overrightarrow{D} \overset{def}{=} (D_1, \ldots, D_n)$. Analogously, we overload set intersection ($\cap$), set union ($\cup$) as well as the subset relation ($\subseteq$) to the n-ary case such that: $\overrightarrow{D} \diamond \overrightarrow{E} \overset{def}{=} \forall i \in \{1, \ldots, n\} : D_i \diamond E_i$, where $\diamond \in \{\subseteq, \cap, \cup\}$. Moreover, if $x : E$ denotes a variable with associated domain E we write $\overrightarrow{D}.x$ for the* x-component *of $\overrightarrow{D}$ such that $\overrightarrow{D}.x = E$.*

If we instantiate the previously defined variable type of a finite domain variable according to the grammar specified in definition 1 we obtain the two most popular variable types in CP: *integer variables* and *set variables*.

Definition 3 (Integer variable and set variable)**.** *Let $x : D \subset \mathtt{Val} \subset \mathcal{T}$ be a finite domain variable as defined previously. If $\mathcal{T} = \mathbb{Z}$ such that x ranges over integer values we call x an* integer variable. *In case that $\mathcal{T} = \mathcal{P}(\mathbb{Z})$ such that x ranges over finite subsets of integers we call x a* set variable.

With this concise definition of the first constituent of a CSP, and given the restricted finite sets $\mathtt{Var}$ and $\mathtt{Val}$, we are now able to define how values $v \in \mathtt{Val}$ are assigned to variables $x \in \mathcal{X}$.

Definition 4 (Assignment)**.** *An* assignment *is a mapping $\alpha :\in \mathtt{Asn} = \mathtt{Var} \rightarrow \mathtt{Val}$ from variables to values according to their respective domain D, that is $\forall (x : D) \in \mathtt{Var} : \alpha(x) = v \in D$. Thus, we say that a variable $x \in \mathtt{Var}$ with corresponding domain D is* assigned *if and only if $|D| = 1$, that is its domain D is a singleton set $D = \{v\}$.*

Provided this definition, we can now formalize the notion of a constraint as:

Definition 5 (Constraint)**.** *A* constraint *is a set of assignments $c \in \mathcal{C} = \mathcal{P}(\mathtt{Asn})$.*

Having defined the notion of a constraint and the notion of finite domain variables we have defined both components we need to give a formal definition of a CSP according to [37]:

Definition 6 (Constraint satisfaction problem (CSP))**.** *We define a* constraint satisfaction problem (CSP) *as a tuple* $C = \langle \mathcal{X}, \mathcal{C} \rangle$ *of a finite set of finite domain variables* $\mathcal{X}$ and a set of constraints $\mathcal{C}$ on them.

Let us consider an example for the encoding of a given problem into a constraint satisfaction problem. Assume that we are given an instance of the popular *Sudoku* puzzle (see [36] by Simonis) as depicted in table 2.1.

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

Table 2.1: Sudoku of order $n = 3$ with nine 3×3-squares on a 9×9-grid

According to the *Gecode*-implementation of the above instance and the CP-Tutorial for the *Alice* language ([38]) we can encode a Sudoku problem of order n as follows: Assume that we are given sets $I = \left\{1, \ldots, n^2\right\}$ and $I^2 = \left\{1, \ldots, n^4\right\}$ with respect to the order n of the specified Sudoku problem. At first we choose the set $\mathtt{Var} = \left\{x_i : \mathcal{P}\left(I^2\right) \mid\ i \in I\right\}$ where each of the n^2 set variables x_i represents the set of indices of boxes for number i. Hence, the partial assignment of the Sudoku grid as shown in table 2.1 represents the set $PA \subset \mathtt{Asn}$ of variable assignments $\alpha \in \mathtt{Asn}$ such that

$$\begin{aligned} PA = \{\alpha \in \mathtt{Asn} \quad | \quad & \{17, 56, 76\} \subset \alpha(x_1), \{34, 60\} \subset \alpha(x_2), \{6, 48\} \subset \alpha(x_3), \\ & \{43, 65\} \subset \alpha(x_4), \{22, 74\} \subset \alpha(x_5), \{8, 50\} \subset \alpha(x_6), \\ & \{21, 41\} \subset \alpha(x_7), \{26, 39, 61\} \subset \alpha(x_8), \{20, 32, 62\} \subset \alpha(x_9)\} \end{aligned}$$

In order to obtain a valid solution to the grid in table 2.1 we specify the set $\mathcal{C}$ of constraints as follows. At first, there are exactly n^2 occurrences for each number $i \in I$, that is

$$card \overset{\text{def}}{=} \{\alpha \in \mathtt{Asn} \mid \forall i \in I : |\alpha(x_i)| = n^2\}$$

. Moreover, a box of the grid can only hold one value that is:

$$disj \overset{\text{def}}{=} \{\alpha \in \mathtt{Asn} \mid \forall i, j \in I, i \neq j : \alpha(x_i) \cap \alpha(x_j) = \emptyset\}$$

Additionally, each number may occur at most once per row $i \in I$:

$$row_i \overset{\text{def}}{=} \{\alpha \in \mathtt{Asn} \mid \forall k, j \in I : \left|\{z \in I^2 \mid (i-1) \cdot n^2 + 1 \leq z \leq i \cdot n^2\} \cap x_k\right| = 1\}$$

at most once per column $j \in I$:

$$col_j \overset{\text{def}}{=} \{\alpha \in \texttt{Asn} \mid \forall k, i \in I : \left|\{z \in I^2 \mid z = j + (i-1) \cdot n^2\} \cap x_k\right| = 1\}$$

and at most once per $n \times n$-block $b \in I$:

$$block_b \overset{\text{def}}{=} \left\{\alpha \in \texttt{Asn} \mid \forall i, j \in \{1, \ldots, 3\}, \forall k \in I : \left|\left\{z \in I^2 \mid z = o + (i-1) \cdot n^2 + j\right\} \cap x_k\right| = 1\right\}$$

where $o = n^3 \cdot \left\lfloor \frac{b-1}{n} \right\rfloor + n \cdot ((b-1) \mod n)$ denotes the number of blocks in the Sudoku grid before block b. Using the partial assignment PA and the constraints on the occurrences of the numbers we just defined, we can now express the set of all constraints as

$$\mathcal{C} = PA \cup \{card, disj\} \cup \bigcup_{k=1}^{n^2} \{row_k, col_k, block_k\}$$

and can encode the Sudoku problem as a CSP $\mathcal{SP} = \langle \texttt{Var}, \mathcal{C} \rangle$.

2.2 Solving

In order to solve a CSP like the above Sudoku-instance $\mathcal{SP}$ we have to represent the constraint variables with their associated domain information as well as the constraints in a constraint solver. However, the domain information coupled with the constraint variables might be of exponential size if they are set variables. Consider for example the $\mathcal{SP} = \langle \texttt{Var}, \mathcal{C} \rangle$ problem as described above. The set variables $x : D = \mathcal{P}\left(I^2\right) \in \texttt{Var}$ reason about the power set of I^2 that is, the size of an initial single domain D of such a variable is exponential, since $\left|\mathcal{P}\left(I^2\right)\right| = 2^{|I^2|} = 2^{n^4}$. Additionally, the representability of the problem encoding for a constraint solver decreases with increasing variable size. Furthermore, it is not tractable to implement the constraints in $\mathcal{C}$ extensionally because of the set of possibly exponentially many combinations of variable-value-tuples they represent (see Schulte and Carlsson[32, chap 1.1]). Consider for example the constraint *card* from the above CSP with order $n = 3$. Implementing this constraint literally into the system would imply encoding all $\alpha \in card$ where the cardinality of *card* is computed as follows:

$$|card| = \binom{n^4}{n^2}^{n^2} \overset{n=3}{=} \binom{81}{9}^9 \approx \left(2.6 \cdot 10^{11}\right)^9 > 10^{80} \tag{2.1}$$

where $|card|$ denotes the number of possibilities to assign subsets $S \subset \mathcal{P}\left(I^2\right)$ to the n^2 problem variables such that $|S| = n^2$. Obviously, in a CSP modeling a Sudoku of order $n = 3$ using set variables as we did above, the number of assignments $\alpha \in \texttt{Asn}$ mapping the problem variables to subsets of $\mathcal{P}\left(I^2\right)$ already exceeds a lower bound estimation on the number of atoms in the observable universe.

Based on the work of Tack *et al.* in [37] and Benhamou in [9], the remaining part of this chapter introduces the terms *domain approximation* and *propagators*. Hence, the next level of abstraction on our way from a problem specification via a CSP-encoding $C = \langle \mathcal{X}, \mathcal{C} \rangle$ towards a problem representation tractable for a constraint solver is the abstraction by a *constraint system* $\mathcal{S} = \langle \overrightarrow{D_{\mathcal{A}}}, P \rangle$. Analogously to the CSP-encoding, a *constraint system* consists of two parts, namely a (reasonable) variable representation $\overrightarrow{D_{\mathcal{A}}}$ and the set of *propagators* P. In this context, P represents the computational analog to the CSP component $\mathcal{C}$, which denotes the set of all constraints on the problem variables. Note that this definition differs from Benhamou' s definition in the use of the set of propagators P instead of the set of constraints $\mathcal{C}$ as second component. Consequently we define a *domain approximation* as stated below:

Definition 7 (Domain approximation)**.** *A* domain approximation *$\mathcal{A}$ for Dom is defined as finite subset $\mathcal{A} \subseteq Dom$ such that $\mathcal{A}$ is closed under intersection, that is $\forall A, B \in \mathcal{A} : (A \cap B) \in \mathcal{A}$. The minimum requirements for $\mathcal{A}$ according to Benhamou [9] are*

$$\forall v \in \{\emptyset, \mathtt{Val}\} \cup \{D \in Dom | |D| = 1\} : v \in \mathcal{A}$$

Thus, a domain approximation has to contain at least the empty set, the universe of a variable and all singleton domains over this universe since those are the sets a constraint solver requires to perform constraint propagation. Elements $D \in \mathcal{A}$ are called approximate domains *of $\mathcal{A}$.*

Remark 2 (Full approximation)**.** *Although Benhamou puts the emphasis on $\mathcal{A}$ as a proper "subset" [9, sect. 2.2, def. 1], the above definition does not exclude the possibility of approximating the set Dom by the set itself. Hence we can choose $\mathcal{A} = Dom$ as a domain approximation for Dom.*

Though the above definition of a *domain approximation* theoretically enables us to represent finite domains of variables via an approximation, we still lack a connector between $\mathcal{A}$ and *Dom* we may use as variable domain representation on the operational level. Here the concept of *variable views* as Schulte and Tack define in [35] comes in handy, where a variable view is used as an abstraction over the underlying data structure of a variable, that is the operational representation of the variable's domain. In our setting we can exploit the *adaptor* functionality of the described views and similarly define:

Definition 8 (Variable View)**.** *Let Dom denote the finite set of possible domains and $\mathcal{A}$, $\mathcal{B}$* domain approximations of *Dom* as defined previously. A mapping $v_{\mathcal{B}} : \mathcal{A} \rightarrow \mathcal{B}$ transforming an approximate domain $D_A \in \mathcal{A}$ into an approximate domain $D_B \in \mathcal{B}$ such that

$$v_{\mathcal{B}}(D_A) = min_{\subseteq}\{D_B \in \mathcal{B} \mid D_A \subseteq D_B\}$$

is called a *variable view*. Hence, a view maps an approximate domain $D_A \in \mathcal{A}$ to the smallest approximate domain $D_B \in \mathcal{B}$ containing D_A.

Clearly, the adaptor functionally of such a view, that is transferring a domain $D \in \mathit{Dom}$ into an approximation $\mathcal{A}$, is obtained by choosing a view $v_{\mathcal{A}} : \mathit{Dom} \to \mathcal{A}$ such that $v_{\mathcal{A}}(D) \in \mathcal{A}$ yields the desired approximate domain.

Remark 3 (n-ary extension)**.** *Since we extended the notion of an approximate domain $D \in \mathcal{A}$ to an n-ary cartesian product $\overrightarrow{D} \in \mathcal{A}^n$ of n finite domains we also write*

$$\overrightarrow{v}_{\mathcal{B}}\left(\overrightarrow{D}\right) \overset{def}{=} (v_{\mathcal{B}}(D_1), \ldots, v_{\mathcal{B}}(D_n))$$

for the lifted application of a variable view $v_{\mathcal{B}}$ mapping each domain in the n-ary product from a domain approximation $\mathcal{A}$ to another approximation $\mathcal{B}$.

2.2.1 Constraint Propagation

As in a propagation based constraint solver a *propagator* is the operational analog of a constraint and we just defined a constraint to reason about all variables in `Var` we define a propagator similarly as:

Definition 9 (Propagator)**.** *Let $\mathcal{A}$ be a domain approximation as defined in 7. A* propagator *is a function $p : \mathcal{A}^n \to \mathcal{A}^n$ which is* contracting

$$\forall \overrightarrow{D} \in \mathcal{A}^n : p\left(\overrightarrow{D}\right) \subseteq \overrightarrow{D} \tag{2.2}$$

and monotone

$$\forall \overrightarrow{D}, \overrightarrow{E} \in \mathcal{A}^n : (\overrightarrow{D} \subseteq \overrightarrow{E}) \Rightarrow p\left(\overrightarrow{D}\right) \subseteq p\left(\overrightarrow{E}\right) \tag{2.3}$$

Moreover, we call a propagator p sound *for a constraint c if and only if*

$$\forall \alpha \in \texttt{Asn} : c \cap \{\alpha\} = p\left(\{\alpha\}\right)$$

and complete *if and only if*

$$\forall \overrightarrow{D} \in \mathcal{A}^n : c \cap \overrightarrow{D} = p\left(\overrightarrow{D}\right)$$

Proposition 1 (Fixpoint computation)**.** *Applying a propagator p to variable domains $\overrightarrow{D}$ is equivalent to computing its fixpoint on $\overrightarrow{D}$, which we denote as $\bigsqcap_{\{p\}} \overrightarrow{D}$.*

Proof. We recursively define the application of a propagator p to input domains $\overrightarrow{D}$ as follows:

$$p^i\left(\overrightarrow{D}\right) = \begin{cases} p\left(p^{i-1}\left(\overrightarrow{D}\right)\right) & \text{if } i > 0 \\ \overrightarrow{D} & \text{else} \end{cases}$$

Let $\forall n \in \mathbb{N} : p^{n+1}\left(\overrightarrow{D}\right) \subseteq p^n\left(\overrightarrow{D}\right)$ be the induction hypothesis. For $n = 0$ the base case trivially holds because p is contracting. As $n + 2 > n + 1$ the induction hypothesis holds for all $m < n + 2$

and by monotonicity of p we obtain:

$$\begin{aligned} p^{n+1}\left(\overrightarrow{D}\right) \subseteq p^{n}\left(\overrightarrow{D}\right) \quad &\overset{p\ monotone}{\Rightarrow} \quad p\left(p^{n+1}\left(\overrightarrow{D}\right)\right) \subseteq p\left(p^{n}\left(\overrightarrow{D}\right)\right) \\ &\Leftrightarrow \quad p^{n+2}\left(\overrightarrow{D}\right) \subseteq p^{n+1}\left(\overrightarrow{D}\right) \end{aligned}$$

As $\mathcal{A}^n$ is finite we know that there exists $m \in \mathbb{N}$ such that the m-fold application of p to $\overrightarrow{D}$ results in the descending chain

$$\overrightarrow{D} \supseteq p\left(\overrightarrow{D}\right) \supseteq \quad \dots \quad \supseteq p^{m-1}\left(\overrightarrow{D}\right) \supseteq p^{m}\left(\overrightarrow{D}\right) \tag{2.4}$$

Moreover, we deduce from $\mathcal{A}^n$ being finite that this chain is maximal, that is

$$\neg\exists k \in \{k \in \mathbb{N} \mid k > m\} : p^{k}\left(\overrightarrow{D}\right) \subset p^{m}\left(\overrightarrow{D}\right) \tag{2.5}$$

$$\Leftrightarrow \quad \forall k \in \{k \in \mathbb{N} \mid k > m\} : p^{m}\left(\overrightarrow{D}\right) \subseteq p^{k}\left(\overrightarrow{D}\right) \tag{2.6}$$

Let $\mu \overset{\text{def}}{=} p^{m}\left(\overrightarrow{D}\right)$. As from equation (2.2) we know that $p(\mu) \subseteq \mu$ and as equation (2.6) implies that $\mu = p^{m}\left(\overrightarrow{D}\right) \subseteq p^{m+1}\left(\overrightarrow{D}\right) = p\left(p^{m}\left(\overrightarrow{D}\right)\right) = p(\mu)$ it follows that $p(\mu) = \mu$ and hence that μ is a *fixpoint* of p. Consider the set

$$R \overset{\text{def}}{=} \left\{p^{n}\left(\overrightarrow{D}\right) \mid n \in \mathbb{N} \wedge \mu \subseteq p^{n}\left(\overrightarrow{D}\right)\right\} \subseteq A^{n} \tag{2.7}$$

containing every element of the above maximal chain and which is closed under intersection (compare remark 1). Hence, μ is the greatest lower bound of R that is contained in R, since

$$\mu = p^{m}\left(\overrightarrow{D}\right) = \bigcap_{i=0}^{m} p^{i}\left(\overrightarrow{D}\right) \tag{2.8}$$

Thus, μ is not only a fixpoint of p but also the unique greatest fixpoint of p. Therefore we conclude that the application of a single propagator p to variable domains $\overrightarrow{D}$ in this setting always yields its unique greatest fixpoint μ as a result and we write $\mu = \bigsqcap_{\{p\}} \overrightarrow{D}$. q.e.d

Propagation results Provided that the iterative application of a propagator p on an approximate domain $\overrightarrow{D}$ results in a fixpoint $\mu \overset{\text{def}}{=} \bigsqcap_{\{p\}} \overrightarrow{D}$ one of the following statements holds:

- If $\mu = \emptyset$ it follows that $\exists x \in \texttt{Var} : p\left(\overrightarrow{D}\right).x = \emptyset$, that is p has detected *failure* because one of the variable domains is inconsistent with the operational semantics of propagator p and thus inconsistent with the constraint c implemented by p.

- If in turn $\mu \neq \emptyset$, we know that μ is a stable state for the application of p on the input domains $\overrightarrow{D}$.

Typically, in a constraint solver, the set of constraints C comprises more than just a single constraint. Consequently, the common case deals with a hole set of propagators P. However, we show that constraint propagation of the set P behaves analogously like a singleton set p with only one propagator as presented above.
Let $|P| = n$ and $\pi : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ be a permutation on $\{0, \ldots, n-1\}$ denoting the order according to which the $p_i \in P$ are applied. Given some initial variable domains $\overrightarrow{D}$ and the usual definition of function composition $(f \circ g)(D) \overset{\text{def}}{=} f(g(d))$ we define:

$$
\begin{aligned}
N_\pi\left(\overrightarrow{D}\right) &\overset{\text{def}}{=} \left(\bigcirc_{i=1}^{n} p_{\pi(i)}\right)\left(\overrightarrow{D}\right) && (2.9)\\
&\overset{\text{def}}{=} \left(p_{\pi(1)} \circ (\cdots \circ p_{\pi(n)})\right)\left(\overrightarrow{D}\right) && (2.10)\\
&\overset{\text{def}}{=} p_{\pi(1)}\left(\ldots\left(p_{\pi(n)}\left(\overrightarrow{D}\right)\right)\right) && (2.11)
\end{aligned}
$$

as the family of all compositions N of all propagators p_i in the set of propagators P.

Proposition 2. *A composition N_π of propagators $p_i \in P$ is contracting.*

Proof by induction. Let $|P| = n$ and

$$\left(\bigcirc_{i=1}^{n} p_{\pi(i)}\right)\left(\overrightarrow{D}\right) \subseteq \overrightarrow{D} \qquad (2.12)$$

be the inductive hypothesis. By induction over n the base case for $P = \{p_{\pi(1)}\}$ immediately follows from $p_{\pi(1)}$ being contracting (eq. 2.2). Let $P' = \bigcup_{i=2}^{n} p_{\pi(i)}$ and $P = P' \cup \{p_{\pi(1)}\}$ be such that $N'_\pi\left(\overrightarrow{D}\right) \overset{\text{def}}{=} \left(\bigcirc_{i=2}^{n} p_{\pi(i)}\right)\left(\overrightarrow{D}\right)$ and $N_\pi\left(\overrightarrow{D}\right) \overset{\text{def}}{=} (p_{\pi(1)} \circ N'_\pi)\left(\overrightarrow{D}\right)$ are the corresponding compositions for P and P' respectively. Since $|P'| = n-1 < |P| = n$, N'_π is contracting by (2.12) and we obtain:

$$
\begin{aligned}
& N_\pi\left(\overrightarrow{D}\right) = (p_{\pi(1)} \circ N'_\pi)\left(\overrightarrow{D}\right) \overset{(2.9)}{=} p_{\pi(1)}\left(\left(\bigcirc_{i=2}^{n} p_{\pi(i)}\right)\left(\overrightarrow{D}\right)\right) && (2.13)\\
\overset{N'_\pi contracting}{\subseteq} \quad & p_{\pi(1)}\left(\overrightarrow{D}\right) \overset{p_{\pi(1)} contracting}{\subseteq} \overrightarrow{D} && (2.14)\\
\Rightarrow \quad & N_\pi\left(\overrightarrow{D}\right) \subseteq \overrightarrow{D} && (2.15)
\end{aligned}
$$

q.e.d

Proposition 3 (Monotonicity of a composition)**.** *A composition N_π of propagators $p_i \in P$ is monotone.*

Proof. Let

$$\forall \overrightarrow{D}, \overrightarrow{E} \in \mathcal{A}^n : (\overrightarrow{D} \subseteq \overrightarrow{E}) \Rightarrow \left(\bigcirc_{i=1}^{n} p_{\pi(i)}\right)\left(\overrightarrow{D}\right) \subseteq \left(\bigcirc_{i=1}^{n} p_{\pi(i)}\right)\left(\overrightarrow{E}\right) \tag{2.16}$$

be the inductive hypothesis. By induction over $n = |P|$ the base case for $P = \{p_{\pi(1)}\}$ is again straightforward by monotonicity of $p_{\pi(1)}$. Let $P' = \bigcup_{i=2}^{n} p_{\pi(i)}$ and $P = P' \cup \{p_{\pi(1)}\}$ again be such that $N'_\pi\left(\overrightarrow{D}\right) = \left(\bigcirc_{i=2}^{n} p_{\pi(i)}\right)\left(\overrightarrow{D}\right)$ and $N_\pi\left(\overrightarrow{D}\right) = (p_{\pi(1)} \circ N'_\pi)\left(\overrightarrow{D}\right)$ are the corresponding compositions for P and P'. Since $|P'| < |P|$, N'_π is monotone by (2.16) and we obtain for all $\overrightarrow{D}, \overrightarrow{E} \in \mathcal{A}^n$:

$$(\overrightarrow{D} \subseteq \overrightarrow{E}) \overset{N'_\pi monotone}{\Rightarrow} \overrightarrow{D}' = N'_\pi\left(\overrightarrow{D}\right) \subseteq \overrightarrow{E}' = N'_\pi\left(\overrightarrow{E}\right) \tag{2.17}$$

$$\overset{p_{\pi(1)} monotone}{\Rightarrow} \quad p_{\pi(1)}\left(\overrightarrow{D}'\right) \subseteq p_{\pi(1)}\left(\overrightarrow{E}'\right) \tag{2.18}$$

$$\Leftrightarrow \quad (p_{\pi(1)} \circ N'_\pi)\left(\overrightarrow{D}\right) \subseteq (p_{\pi(1)} \circ N'_\pi)\left(\overrightarrow{E}\right) \tag{2.19}$$

$$\Leftrightarrow \quad N_\pi\left(\overrightarrow{D}\right) \subseteq N_\pi\left(\overrightarrow{E}\right) \tag{2.20}$$

q.e.d

By definition 9 and propositions (2-3) it follows:

Corollary 1 (Composition as propagator). *A composition N_π of propagators $p_i \in P$ is again a propagator.*

Given initial variable domains $\overrightarrow{D}$ we know by the above corollary and the definition of a propagator (sec. 2.2.1) that there exists a unique greatest fixpoint $\mu = \bigsqcap_P \overrightarrow{D}$ such that $N_\pi^m\left(\overrightarrow{D}\right) = \mu$.

Proposition 4 (Mutual fixpoint). *Let $\overrightarrow{D}$ denote initial variable domains. Then the unique greatest fixpoint μ of a composition N_π of propagators $p_i \in P$ applied to $\overrightarrow{D}$ is mutual fixpoint of all p_i.*

Proof.

$$N_\pi(\mu) = \mu \quad \overset{(2.9)}{\Leftrightarrow} \quad \left(\bigcirc_{i=1}^{n} p_{\pi(i)}\right)(\mu) = \mu \tag{2.21}$$

$$\overset{(2.9)}{\Leftrightarrow} \quad \forall p_i \in P : p_{\pi(i)}(\mu) = \mu \Leftrightarrow \forall p_i \in P : p_i(\mu) = \mu \tag{2.22}$$

Hence, from equation (2.22) it follows that the fixpoint μ of the composition N_π is not only the greatest fixpoint of N_π but also a *common* fixpoint of all $p_i \in P$. q.e.d

Proposition 5 (Order independence)**.** *The fixpoint $\mu = \bigsqcap_P \overrightarrow{D}$ of a composition N_π and input domains $\overrightarrow{D}$ is independent of the application order π.*

Proof. Assume by proposition 1 that $\mu = \bigsqcap_P \overrightarrow{D}$ is the unique greatest fixpoint of N_π on $\overrightarrow{D}$. Let further $\eta \overset{\text{def}}{=} M_\tau^k\left(\overrightarrow{D}\right)$ denote the unique greatest fixpoint of another composition M_τ and corresponding permutation $\tau : \{0, \ldots, n-1\} \to \{0, \ldots, n-1\}$ of the propagators in P such that M_τ only differs from N_π with respect to the order in which the p_i's are applied. Consequently, by proposition 4

$$\forall p_i \in P : p_{\tau(i)}(\eta) = \eta \quad \Leftrightarrow \quad \forall p_i \in P : p_i(\eta) = \eta \tag{2.23}$$

also holds for M_τ. Assuming without loss of generality that $\mu \subseteq \eta$ we can conclude that:

$$\begin{aligned}
\mu \subseteq \eta \quad &\overset{N_\pi \text{ monotone}}{\Rightarrow} \quad N_\pi(\mu) \subseteq N_\pi(\eta) && (2.24)\\
&\overset{(2.21)}{\Leftrightarrow} \quad \mu \subseteq N_\pi(\eta) \overset{(2.9)}{=} \left(\bigcirc_{i=1}^{n} p_{\pi(i)}\right)(\eta) \overset{(2.23)}{=} \eta && (2.25)\\
& && (2.26)
\end{aligned}$$

Obviously, η is a fixpoint of M_τ and as a such $\eta \subset \overrightarrow{D}$. Moreover, η is also a fixpoint of N_π, that is η has to be a member of the descending chain resulting from the application of N_π (see eq. (2.4)). Hence, η has to be in the set R representing this chain as defined in equation (2.7). Consequently, we can conclude the proof as follows:

$$\begin{aligned}
(2.25) \quad &\overset{(2.7)}{\Leftrightarrow} \quad \mu \subseteq \eta \in \left\{N_\pi^t\left(\overrightarrow{D}\right), t \in \mathbb{N} \mid \mu \subseteq N_\pi^t\left(\overrightarrow{D}\right)\right\} && (2.27)\\
&\overset{(2.8)}{\Rightarrow} \quad \mu = \eta && (2.28)
\end{aligned}$$

Hence equation 2.28 underlines that every possible composition N_π of propagators $p_i \in P$ applied to variable domains $\overrightarrow{D}$ has the same unique fixpoint $\mu = \bigsqcap_P \overrightarrow{D}$ no matter what propagation order π we choose. q.e.d

Therefore, we generalize the notion of *constraint propagation* to the computation of the *greatest mutual fixpoint* $\bigsqcap_P \overrightarrow{D}$ of a set of propagators P which is independent of the propagation order π.

Definition 10 (Constraint Propagation)**.** *Given a set of propagators P, a domain approximation $\mathcal{A}$ and initial approximate domains $\overrightarrow{D}$ we define* constraint propagation *as the computation of the greatest mutual fixpoint of all $p_i \in P$ on $\overrightarrow{D}$, namely $\mu = \bigsqcap_P \overrightarrow{D}$.*

Analogously to definition 9 we finally define *soundness* and *completeness* for a set of propagators P as follows:

Definition 11 (Soundness)**.** *A set of propagators P is called* sound *for a set of constraints $\mathcal{C}$* if and only if $\forall \alpha \in \texttt{Asn} : \bigcap\limits_{c \in \mathcal{C}} \cap\{\alpha\} = \bigsqcap\limits_{P} \{\alpha\}$.

Definition 12 (Completeness)**.** *A set of propagators P is called* complete *for a set of constraints $\mathcal{C}$* if and only if $\forall \vec{D} \in \mathcal{A}^n : \bigcap\limits_{c \in \mathcal{C}} \cap\vec{D} = \bigsqcap\limits_{P} \vec{D}$.

Variable Modification In fact, different propagation orders p_i representing the order in which the propagators $p_i \in P$ are applied to given approximate domains $\vec{\in}\mathcal{A}$ may lead to different lengths of the propagation chain as depicted in equation (2.4). Hence, a practical implementation of constraint propagation is not a static order we know in advance, but instead is computed dynamically by the modifications a propagator p_i possibly performs on the current variable domains. In order to keep track of those domain modifications and to improve the application of the propagators $p_i \in P$ Schulte and Stuckey present in [33] the concept of *modification events* to determine the propagation order. Modification events for constraint variables essentially differ in two aspects: On the one hand there are modification events that do not depend on the domain approximation. The following events arising after execution of a propagator are common for any non-trivial domain approximation $\mathcal{A}$ such that $\mathcal{A} \supset \{\emptyset, \texttt{Val}\} \cup \{D \in Dom | |D| = 1\}$:

- $val(x)$ stating that the variable x has been assigned
- $fail(x)$ denoting that the variable domain is inconsistent with the propagator's operational semantics
- $dom(x)$ indicating that the domain of x has changed

Obviously, in the trivial case that $\mathcal{A} =\supset \{\emptyset, \texttt{Val}\} \cup \{D \in Dom | |D| = 1\}$ the modification events $dom(x)$ and $val(x)$ coincide as every domain change in the trivial approximation implies that the domain has become a singleton. On the other hand there are also modification events which only become available through the current domain approximation. Examples for those events involve modifications on domain bounds or changes on the cardinality of a variable domain.

2.2.2 Search

As constraint propagation alone is incomplete with respect to solving a constraint system (see 2.1, a constraint solver also needs a search component. However, this search component is again two-fold. Thereby, the first component consists in *branching* or *distribution* (see [31, sec. 2.2]) which, together with constraint propagation, determines the shape of the search-tree that is created during interleaved application of constraint propagation and search. Given a constraint system $S = \left(\vec{D}, P\right)$ such that $\vec{D}$ is already a fixpoint of P branching splits S into k smaller constraint systems $S_1 = \left(\vec{D}_1, P\right), \ldots, S_k = \left(\vec{D}_k, P\right)$ by choosing a variable $x : D$ according to a

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

Table 2.2: Sudoku instance after constraint propagation

specified *branching strategy* such that its corresponding domain $\overrightarrow{D}.x$ is reduced in size, that is $\overrightarrow{D}.x = \biguplus_{j=1}^{k} \overrightarrow{D}_k.x$. The traversal of a search tree is performed in the second component of the search part, namely the tree exploration. Here the tree is traversed according to a given *search strategy* like for example depth-first-search (DFS) and leads after the exploration of failures to a first solution, which is represented in table 2.3

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

Table 2.3: Solution to the Sudoku-instance from 2.1

In this chapter we provided a brief introduction to the topic of constraint programming as application area for this thesis. Moreover we set up basic definitions and notations the following chapters will use. Now, we have seen a brief glimpse on how a given problem specification is solved using the constraint programming paradigm. It uses the modeling and solution phases divided into the interleaved processing of iterated inferences on partial domain information and search. Consequently, we conclude that the representation of constraint variables in a constraint solver is crucial to the tractability of a problem and to the efficiency concerned with the time spent in falsifying the problem or finding a solution for it. In this context, tractability corresponds to the question, whether the problem is representable in a constraint solver or not. Even more, the variable representation is an important factor in constraint programming if we focus on constraint variables with larger domains like set variables. Therefore, the remaining chapters of this thesis focus on domain approximations as variable representations and discuss the questions of what domain approximations there are, how such a domain approximation can be implemented and finally, how it interacts with propagators working those approximations.

3 Approximating finite sets

As we have seen in the Sudoku example from chapter 2, the use of set variables for solving a given problem in CP comes with the possibility of exponential domain sizes depending on the universe of discourse the set variables range over. As a consequence, a crucial aspect of constraint solvers supporting constraint programming with finite set variables is the availability of variable representations that are able to cope with exponentially large finite set domains. Since the previous chapter introduced the concept of domain approximations as a viable solution to the variable representation problem for set variables, the first part of this chapter presents a survey of existing domain approximations for the domain of finite set variables. Subsequently, we will discuss implementations for two of the presented domain approximations.

3.1 Domain Approximations - A survey

The definition of a domain approximation $\mathcal{A}$ (see definition 7) in the previous chapter states that there are two possible choices for $\mathcal{A}$: Either *Dom* is approximated by a representative subset, that is $\mathcal{A} \subset Dom$, or we choose *Dom* itself as an approximation, that is $\mathcal{A} = Dom$. At first we take a closer look at approximations $\mathcal{A}$ forming a proper subset of *Dom*.

3.1.1 Interval Reasoning

As we will see in this section, the majority of domain approximations for the set variable domains as presented in literature and used in most constraint solvers reasoning about set variables exploit the same common principle of *interval reasoning*. Since we restricted our framework to CP(FD) in section 2.1.1 it follows that the domain of a set variable $x : D$ is an element of the set *Dom*, in explicit $D \in Dom = \mathcal{P}(\mathtt{Val}) = \mathcal{P}(\mathcal{P}(\mathcal{U}))$, where $\mathcal{U} \subset \mathbb{Z}$ is the finite universe of discourse. In this context, the major insight of the above mentioned approximations is to impose an ordering relation $\sqsubseteq$ on the set $\mathtt{Val}$ such that $\forall D \in Dom : \emptyset \sqsubseteq D \sqsubseteq \mathtt{Val}$, where $\sqsubseteq$ is either a *partial* or a *total* order as stated in the following order-theoretical definitions according to Gratzer[20]:

Definition 13 (Partial and total order)**.** *Let S be a set and $\sqsubseteq S$ a binary relation on S. Provided the following properties*

P_1	transitivity	$\forall x, y, z \in S : (x \sqsubseteq y \land y \sqsubseteq z) \Rightarrow x \sqsubseteq z$
P_2	reflexivity	$\forall x \in S : x \sqsubseteq x$
P_3	antisymmetry	$\forall x, y \in S : (x \sqsubseteq y \land y \sqsubseteq x) \Rightarrow x = y$
P_4	linearity	$\forall x, y \in S : a \sqsubseteq b \lor b \sqsubseteq a$

we call $\sqsubseteq$ *a* partial order *if it satisfies properties* P_1*,* P_2 *and* P_3 *and we call* $\sqsubseteq$ *a* total order *if it satisfies* P_1*,* P_3 *and* P_4*, where* totality (P_4) *already implies* reflexivity (P_2).
Equipping S *with* $\sqsubseteq$ *we call* S *a* partially ordered set *if* $\sqsubseteq$ *is a partial order and a* totally ordered set *if* $\sqsubseteq$ *is a total order, written* $\langle S, \sqsubseteq \rangle$.

Hence, by choosing a partial or a total order $\sqsubseteq$ on `Val`, `Val` becomes either a partially or a totally ordered set respectively and we are able to define the *lower* and *upper bound* of a subset $D \subseteq S$ according to Davey and Priestley[11] and Gervet in [17, sec. 2.1] as:

Definition 14 (Lower and upper bound)**.** *Let* $\langle S, \sqsubseteq \rangle$ *be an ordered set and let* $D \subseteq S$*. If* $\forall d \in D : x \sqsubseteq d$ *then we call* x *a lower bound of* D*, written* $x = \uparrow D$*. Similarly, if* $\forall d \in D : d \sqsubseteq y$ *then we call* y *an upper bound of* D*, written* $y = \downarrow D$*.*

The main result from this order-theoretical approach is the following: On the one hand we know that we can establish at least a partial order on `Val` and on the other hand we know that due to finiteness `Val` has a supremum and an infimum, namely $\uparrow$ `Val` $= \emptyset$ and $\downarrow$ `Val` $= \mathcal{U}$. As this argumentation carries over to the set $Dom = \mathcal{P}(\texttt{Val})$ we also know that Dom is bounded by $\uparrow Dom = \emptyset$ and $\downarrow Dom =$ `Val`. Consequently, it follows that for every non-empty finite subset $D \in Dom$ we are able to compute its greatest lower bound (infimum) and its least upper bound (supremum) as presented in [11]:

Definition 15 (Infimum and supremum)**.** *Let* $\langle S, \sqsubseteq \rangle$ *be an ordered set and let* $D \subseteq S$*. If there exists* $y \in S$*, such that* $y = \uparrow D$ *and* $\forall s \in S : ((\forall t \in D : s \sqsubseteq t) \Leftrightarrow s \sqsubseteq y)$ *then we call* y *the greatest lower bound of* D*, written* $y = \inf(D)$ *or* $y = \mathrm{glb}(D)$*. Analogously, if* $x \in S$*, such that* $x = \downarrow D$ *and* $\forall s \in S : ((\forall t \in D : t \sqsubseteq s) \Leftrightarrow x \sqsubseteq s)$ *then we call* x *the least upper bound of* D*, written* $x = \sup(D)$ *or* $x = \mathrm{lub}(D)$*.*

Using the above lattice- and order-theoretical definitions, we define an interval over an ordered set $\langle S, \sqsubseteq \rangle$ as follows:

Definition 16 (Interval)**.** *Given a partially or totally ordered set* $\langle S, \sqsubseteq \rangle$ *and elements* $a, b \in S$*, we call the set*

$$[a..b]_{\sqsubseteq} \overset{def}{=} \{s \in S \mid a \sqsubseteq s \sqsubseteq b\}$$

interval *from* a *to* b*. For the sake of simplicity we write* $[a..b]$ *if it becomes clear from the context what partially (totally) ordered set we focus on. Given an interval* $I = [a..b]_{\sqsubseteq}$ *we abbreviate* $\lfloor I \rfloor \overset{def}{=} \min(I) = a$ *and* $\lceil I \rceil \overset{def}{=} \max(I) = b$*. Reasoning about an ordinary integer interval* $[a..b]_<$ *we write* $\underline{I} = \min(I) = a$ *and* $\overline{I} = \max(I) = b$*.*

Provided a partial or total order $\sqsubseteq$ we know that $\langle Dom, \sqsubseteq\rangle$ and $\langle \texttt{Val}, \sqsubseteq\rangle$ are at least partially ordered sets. Thus, the domain approximations presented in this section fix a partial or total order $\sqsubseteq$ and approximate a domain $D \in Dom$ by the interval $[\inf(D) .. \sup(D)]_{\sqsubseteq}$. More formally, we define the *convexity* of a set T according to Hawkins *et al.*[22, sec. 2.1] as

Definition 17 (Convexity)**.** *Let $L = \langle S, \sqsubseteq\rangle$ be a partially (totally) ordered set. A subset $T \subseteq S$ is* convex (with respect to $\sqsubseteq$) *if $\forall t \in S : a \sqsubseteq t \wedge t \sqsubseteq b \Rightarrow t \in T$. More precisely, T is convex if and only if T forms an interval in S (see definition 16).*

and notice that given a set variable $x : D \in \texttt{Var}$, the interval $I = [\inf(D) .. \sup(D)]_{\sqsubseteq}$ is exactly the *convex hull* of D, that is I is the smallest convex subset of $\langle \texttt{Val}, \sqsubseteq\rangle$ that contains D. Hence, we can characterize the domain approximations focusing on interval reasoning as *convex approximations*. The advantage of this approach is obvious: those approximations only need to represent the two sets $\inf(D)$ and $\sup(D)$ in a constraint solver and are able to compute the whole domain D. In the ensuing paragraphs we will discuss in detail how those convex domain approximations define the ordering relation $\sqsubseteq$ and what approximate domains result from this ordering.

Set Bounds

The state-of-the-art domain approximation for set variables in CP is the *set bounds* approximation $\left(\mathcal{S}\right)$, first introduced in CP by Puget in [28] and described in full detail in the dissertation of Gervet, [16, chp. 4]. In this context Gervet defines the approximation using the subset inclusion relation ($\subseteq$) as partial order over Val, that is the partially ordered set $\langle \texttt{Val}, \subseteq\rangle$. Therefore, the set bounds approximation $\left(\mathcal{S}\right)$ of Dom is defined as:

$$\left(\mathcal{S}\right) \overset{\text{def}}{=} \{d \in Dom \mid d \text{ convex with respect to } \subseteq\} \tag{3.1}$$

Given a set $D \in Dom$ the greatest lower bound of D, $\inf(D)$, and the least upper bound of D, $\sup(D)$, are computed via set intersection ($\cap$) and set union ($\cup$) respectively. Thus, a domain $D \in Dom$ is represented by its approximated domain E, such that:

$$E = [\inf(D) .. \sup(D)] \overset{\text{def}}{=} \left[\bigcap_{d \in D} d .. \bigcup_{d \in D} d\right] \tag{3.2}$$

Since by the above equation and by definition 16 we know that $\emptyset = [\emptyset..\emptyset]$, $\texttt{Val} = [\emptyset..\mathcal{U}]$ and $\texttt{Val} \ni \{u\} = [\{u\} .. \{u\}]$ for all $u \in \mathcal{U}$ it consequently follows that

$$\forall v \in \{\emptyset, \texttt{Val}\} \cup \{D \in \texttt{Val} \mid |D| = 1\} : v \in \left(\mathcal{S}\right) \tag{3.3}$$

Hence, by definition 7 $\left(\mathcal{S}\right)$ is a domain approximation. Note however that, due to $\subseteq$ being a partial order, for a given domain D $\inf(D)$ and $\sup(D)$ are not necessarily elements of D itself as it is the case for the sets mentioned in equation (3.3).

Cardinality Information Most constraint solvers using $\left(\mathcal{S}\right)$ as domain approximation for set variables like *Conjunto* [15], *Mozart* [40], *Gecode* [39], *ILOG* [1], *ECL^{i}PS^e* [42] and *Cardinal* [5] use a more fine-grained instance of the above set bounds approximation including some extra cardinality information as stated in [16]. In this case, a given domain $D \in Dom$ of a set variable $x : D$ is not only represented by an approximate domain $E \in \left(\mathcal{S}\right)$ but is also equipped with an additional integer interval $[l..r] \subset \mathbb{N}$ approximating the set of cardinalities $C = \{c \in \{0,\dots,|\mathcal{U}|\} \mid \exists d \in D : c = |d|\}$, where $[l..r] = [\min_{d\in D}\{|d|\}..\max_{d\in D}\{|d|\}]$. Hence, we can define the *cardinality set bounds approximation* $\left(\mathcal{C}\right)$ as

$$\left(\mathcal{C}\right)_l^r \overset{\text{def}}{=} \left(\mathcal{S}\right) \cap \left\{s \in \left(\mathcal{S}\right) \mid |\lfloor s \rfloor| \geq l \land |\lceil s \rceil| \leq r\right\}$$

Thus, the proper set bounds approximation $\left(\mathcal{S}\right)$ can be expressed in terms of the cardinality set bounds approximation as $\left(\mathcal{S}\right) = \left(\mathcal{C}\right)_0^{|\mathcal{U}|}$. The obvious gain from this more fine-grained representation of the set bounds approximation is the possibility to restrict $\left(\mathcal{C}\right)$ to subsets of $\left(\mathcal{S}\right)$.

Example 1. *Consider a set variable $x : D = [\emptyset..[1..4]]$ whose elements are restricted to have at least cardinality 2 and at most cardinality 3, that is $2 \leq |x| \leq 3$. Then the Hesse diagram in figure 3.1 gives us a graphical representation of all values in D with respect to cardinality restrictions, where an edge $e(v_1, v_2)$ connects to vertices if $v_1 \subseteq v_2$:*

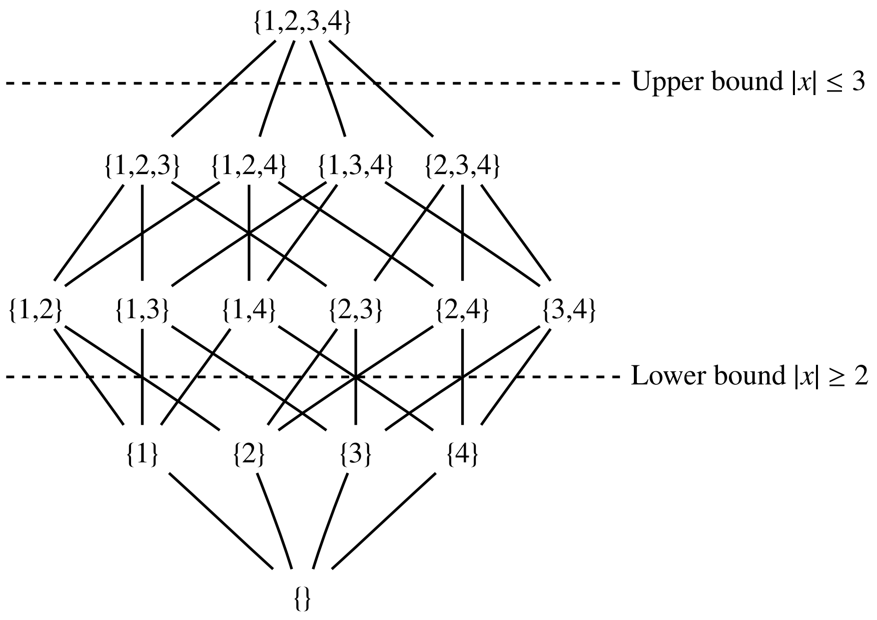

Figure 3.1: Partial order $\langle \mathcal{P}(\{1,\dots,4\}), \subseteq\rangle$ restricted to cardinality $[2,3]$.

Main Advantage The reason why the set bounds approximation is predominant in current constraint solvers is the way it is defined. In contrast to the other convex approximations we come across in this section the set bounds interval $[a..b]_{\subseteq}$ guarantees the following implication. Consider a set variable $x : D$. For all variable assignment $\alpha \in \texttt{Asn}$ and the set bounds interval $E = [\inf(D)..\sup(D)]_{\subseteq}$ we obtain the following implications according to the *extension*-property (C_1) Gervet identifies in [16, sect.4.2.3 p.45]:

$$\forall v \in \lfloor E \rfloor \quad \Rightarrow \quad v \in \alpha(x) \tag{3.4}$$

$$\forall v \notin \lceil E \rceil \quad \Rightarrow \quad v \notin \alpha(x) \tag{3.5}$$

Hence, the set $\lfloor E \rfloor = \inf(D)$ denotes the set of all values definitively belonging to the final value $\alpha(x)$ assigned to the variable (see [18]). Contrarily, values that do not belong to the set $\lceil E \rceil = \sup(D)$ cannot be part of the value $\alpha(x)$ assigned to the variable. Thus, by representing the approximate domain D of a set variable $x : D$, the set bounds approximation $\left(\mathcal{S}\right)$ guarantees that we already obtain a a partial variable assignment for x just by keeping track of the two interval bounds $\lfloor E \rfloor$ and $\lceil E \rceil$.

Lexicographic Bounds

Apart from the set bounds and the cardinality set bounds approximation that rely on a partial order on the set `Val` there is another convex domain approximation for set domains, namely the *lexicographic bounds approximation* $\left(\mathcal{L}\right)$ as presented by Sadler and Gervet in [30]. In contrast to the set bounds approximations $\left(\mathcal{S}\right)$ and $\left(\mathcal{C}\right)$ that based on the partial subset inclusion order ($\subseteq$) the lexicographic bounds approximation uses the following lexicographic order $\preceq$ as formalized in [30, chap. 4]:

Definition 18 (Lexicographic ordering)**.** *Let* $x, y \in \mathit{Dom}$, $\overline{x} = \max(x)$ *and* $\overline{y} = \max(y)$. *Then*

$$x \preceq y \overset{def}{=} x = \emptyset \vee \overline{x} < \overline{y} \vee (\overline{x} = \overline{y} \wedge (x \setminus \{\overline{x}\}) \preceq (y \setminus \{\overline{y}\}))$$

Since the lexicographic order $\preceq$ on the set `Val` is a total order and thus yields a totally ordered set $\langle \texttt{Val}, \preceq \rangle$ we define the $\left(\mathcal{L}\right)$ similarly to the set bounds definition above as

$$\left(\mathcal{L}\right) \overset{\text{def}}{=} \{d \in \mathit{Dom} \mid d \text{ convex with respect to } \preceq\} \tag{3.6}$$

Since the lexicographic order $\preceq$ imposes a total order on `Val` the values of a given domain $D \in \mathit{Dom}$ form a finite chain ordered by $\preceq$. Hence, the infimum and the supremum for D are themselves contained in D and consequently denote the minimum and the maximum of the chain represented by D. Since minimum and maximum of D with respect to $\preceq$ are defined via the lower and upper bound of D we obtain the following approximate domain E for D:

$$E = [\inf(D)..\sup(D)] = [\min(D)..\max(D)] \quad \overset{\text{def}}{=} \quad [\uparrow D..\downarrow D] \tag{3.7}$$

Drawback Though this lexicographic approximation yields totally ordered approximate domains for the set variable domains the major drawback of this domain approximation consists in the fact that the implications from equations (3.4) and (3.5) do not hold anymore for $\left(\mathcal{L}\right)$. Consider a set variable $x : D$, with domain $D \in Dom$ and a variable assignment $\alpha \in \texttt{Asn}$ for x. Approximating D by E as defined in (3.6) still $\lfloor E \rfloor \preceq \alpha(x)$ and $\alpha(x) \preceq \lceil E \rceil$ hold, but in contrast to the set bounds approximation not all the values in the set $\lfloor E \rfloor$ necessarily belong to the set $\alpha(x)$ assigned to the variable. Compared to the set bounds approximation $\left(\mathcal{S}\right)$, a further drawback of this approximation is a loss of cardinality information. Assume that we approximate a domain $D \in Dom$ with a set bounds interval $F = [a..b]_{\subseteq}$ and a lexicographic bounds interval $E = [a..b]_{\preceq}$. Whilst for F it follows that $\forall v \in F : |\lfloor F \rfloor| \leq |v| \leq |\lceil F \rceil|$ the lexicographic interval E may contain sets that are even smaller than $\lfloor E \rfloor$. However we have to underline that $\left(\mathcal{L}\right)$ was originally designed to be used as a hybrid domain together with a set bounds interval and these drawbacks only appear if we only consider $\left(\mathcal{L}\right)$ as domain approximation without further set bounds or cardinality information.

Cardinality Information As an approach to obtain a lexicographic domain approximation that is able to cope with cardinality information without the support of an additional set bounds interval Gervet and Hentenryck formalize in [19] a domain approximation that is convex with respect to a length-lex ordering $\ll$ they define as follows:

Definition 19 (Length-lex ordering)**.** *Let $x, y \in Dom$, $\underline{x} = \min(x)$ and $\underline{y} = \min(y)$. Then*

$$x \ll y \Leftrightarrow x = \emptyset \vee |x| < |y| \vee \left(|x| = |y| \wedge \left(\underline{x} < \underline{y} \vee \underline{x} = \underline{y} \wedge x \setminus \underline{x} \ll y \setminus \underline{y}\right)\right)$$

Like the lexicographic order $\preceq$, the length-lex order also imposes a total order on the set $\texttt{Val}$. Hence we define the *length-lexicographic bounds approximation* $\left(\mathcal{LL}\right)$ as

$$\left(\mathcal{LL}\right) \overset{\text{def}}{=} \{d \in Dom \mid d \text{ convex with respect to } \ll\} \tag{3.8}$$

Although the length-lex order ($\ll$) is a more fine-grained approach to the lexicographic order, that is $\left(\mathcal{LL}\right) \subseteq \left(\mathcal{L}\right)$ and although it avoids the loss of cardinality information by including the cardinality of the compared sets into the order, it still does not satisfy the implications for the set bounds approximations (compare equations (3.4-3.5)).

3.1.2 Characteristic function encoding

Apart from these convex approximations $\mathcal{A} \in \left\{\left(\mathcal{S}\right), \left(\mathcal{C}\right), \left(\mathcal{L}\right), \left(\mathcal{LL}\right)\right\}$ where the domain approximation $\mathcal{A}$ is chosen such that $\mathcal{A} \subset Dom$ we also consider the full approximation $\left(\mathcal{F}\right)$ which we obtain by choosing $\left(\mathcal{F}\right) = Dom$. Since from theorem 1 in [30, chap. 4] it follows that $\left(\mathcal{S}\right) \subseteq \left(\mathcal{L}\right)$ there is the following relation between the convex approximations and the full domain approximation:

$$\left(\mathcal{C}\right) \subseteq \left(\mathcal{S}\right) \subseteq \left(\mathcal{LL}\right) \subseteq \left(\mathcal{L}\right) \subseteq \left(\mathcal{F}\right) \tag{3.9}$$

From the previous section we know that convex approximations are able to represent domains $D \in Dom$ by smaller approximate domains $D_{\mathcal{A}}$, namely convex intervals $I = [a..b]_{\sqsubseteq}$ with respect to a partial or total order $\sqsubseteq$ and consequently only need to keep track of the interval bounds $\lfloor I \rfloor$ and $\lceil I \rceil$ in a constraint solver. Contrarily, the full domain approximation has to represent the complete variable domain $D \in Dom$ of a set variable $x : D$ in a constraint solver. Hence, if a set variable's domain $D \in Dom$ has exponential size, so does the approximate domain $F = D$ and consequently, a representation of $\left(\mathcal{F}\right)$ in a constraint solver has to keep track of possibly exponentially many values $v \in F$. Therefore $\left(\mathcal{F}\right)$ exploits the fact, that we can represent a set D by its characteristic function χ_D as Hawkins *et al.* underline in [22, sect. 1] and Gervet states in [17], such that

$$\chi_D : \mathcal{U} \to \mathbb{B}, \chi_D(v) = \begin{cases} 1 & \text{if } v \in D \\ 0 & \text{else} \end{cases}$$

Moreover, since $D \in Dom$ we also can encode all values $v \in D$ by their respective characteristic functions χ_v. Comparing the convex domain approximations as presented in section 3.1.1 and the full domain approximation as presented in section 3.1.2 we conclude that, provided a set variable $x : D$ with respective domain $D \in Dom$ with approximate domain E, the only determining factor for the efficiency of the convex approximations consists in the representation of the bounds $\lfloor E \rfloor$ and $\lceil E \rceil$, whereas the efficiency of the full domain approximation comes with an efficient representation of the characteristic function χ_D of D. For instance, we will see an implementation based on reduced ordered binary decision diagrams in section 3.2.3.

Conclusion In this section we presented a survey of domain approximations based on either a convex or on a complete bounds reasoning. Moreover we noticed, that not all domain approximations are consistent in the strong sense, that they meet Gervet's *extension*-properties stated in equations (3.4) and (3.5). However, introducing a convex and a complete domain approximation is exactly what adds different *consistency* notions to the constraint solver, namely, *set bounds($\mathcal{A}$)-consistency* and the notion of *domain consistency*. Analogously to Schulte and Stuckey in [34, sect. 2.2-2.3] we define these notions as follows:

Definition 20 (Bounds($\mathcal{A}$)-consistency)**.** *Let $x : D \in$ `Var` be a set variable,* $\mathcal{A}$ a domain approximation, $\langle \mathcal{A}, \sqsubseteq \rangle$ a partially or totally ordered set and $E \in \mathcal{A}$ an approximate domain for D. If for a given constraint c the equation

$$\forall \alpha \in c : \alpha \in E \overset{\text{def}}{=} [\inf(D) .. \sup(D)]_{\sqsubseteq} = \left[\bigcap_{\alpha \in c} \alpha(x) .. \bigcup_{\alpha \in c} \alpha(x) \right]$$

holds, such that $\inf(D)$ is the intersection of all possible assignments $\alpha(x)$ for x and $\sup(D)$ is the union of all possible assignments $\alpha(x)$ for x, we say that E is bounds($\mathcal{A}$)-consistent, written $\mathtt{bc}_{\mathcal{A}}(E)$. Let $\overrightarrow{D} \in \mathcal{A}^n$ denote a tuple of initial approximate domains for a propagator $p : \mathcal{A}^n \to \mathcal{A}^n$. If the equation

$$\forall x \in \mathtt{Var} : \mathtt{bc}_{\mathcal{A}}\left(p\left(\overrightarrow{D}\right).x\right)$$

holds, we say that p is bounds($\mathcal{A}$)-consistent, written $\mathtt{bc}_{\mathcal{A}}(p)$.

Remark 4 (Strength of bounds($\mathcal{A}$)-consistency)**.** *Note however, that approximate domains $D \in \left(\mathcal{S}\right)$ or $D \in \left(\mathcal{C}\right)$ have a higher degree of consistency than approximate domains from other convex approximations presented above, because $\left(\mathcal{S}\right)$ and $\left(\mathcal{C}\right)$ are per definition the only convex domain approximations satisfying Gervet's* extension-*properties.*

Definition 21 (Domain consistency)**.** *Let $x : D \in \mathtt{Var}$ be a set variable. We say that D is* domain-consistent *for a constraint c, written $\mathtt{dc}(D)$ if*

$$D = \bigcup_{\alpha \in c} \alpha(x) \cap D$$

that is D is the least set containing all values for x that are consistent with c. Let $\overrightarrow{D} \in \mathcal{A}^n$ be initial domains for a given propagator $p : \mathcal{A}^n \to \mathcal{A}^n$. If the equation

$$\forall x \in \mathtt{Var} : \mathtt{dc}\left(p(\overrightarrow{D}).x\right)$$

holds, we say that p is domain-consistent, written $\mathtt{dc}(p)$.

Remark 5 (Domain consistency)**.** *Obviously, the only domain-consistent approximation we discussed is the approximation $\left(\mathcal{F}\right)$. Hence, we conclude that only propagators $p : \left(\mathcal{F}\right) \to \left(\mathcal{F}\right)$ can be domain-consistent.*

3.2 Implementing domain approximations

Over the course of the next two sections we discuss the implementation of the cardinality set bounds approximation $\left(\mathcal{C}\right)$ as provided in the *Gecode* -library [39] and the implementation for the full domain approximation $\left(\mathcal{F}\right)$ as proposed by Hawkins *et al.* in [22, 21] and Lagoon and Stuckey in [24] using the *Gecode* -framework. In section 2.2.1 we defined propagators as functions from tuples of approximate domains to tuples of approximate domains. These tuples are in turn composed of variable views as explained in section 7. Moreover, these views depend on *propagation services* as coined by Schulte and Carlsson in [32], where the term of propagation services implies *update* and *lookup* operations on the underlying domain representation as well as the *detection* of inconsistent domains. As the operational semantics of propagators are based on the current domain information, *domain lookup* especially comprises *iterating* over a set variable's domain as formalized by Schulte and Tack in [35, sect. 5], that is access the very elements of a domain. As a consequence the efficiency of a propagator operating on set variables and hence the efficiency of the related constraint solver clearly depend on the time needed to perform domain update and domain lookup operations. Taking a closer look on these operations a set variable domain has to provide, the subsequent paragraphs discuss the representation of the different approximate domains as implemented in the *Gecode* constraint library.

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

According to section 3.1.1 the major issue concerning the implementation of the cardinality subset bounds domain approximation $\left(\mathcal{C}\right)$ is the representation of the lower and upper set bounds $\lfloor I \rfloor$ and $\lceil I \rceil$ of the interval $I = [a..b]$ approximating the domain D of a set variable $x : D \in \mathtt{Var}$. As Gervet points out in [18] there are numerous ways of implementing those bounds either using sorted lists representing the values of those bounds or arrays of integer variables $v : D$ restricted to Boolean domains $D \subseteq \mathbb{B}$ or bitmaps encoding the respective characteristic functions of the sets $\lfloor I \rfloor$ and $\lceil I \rceil$.

Single set representation The *Gecode* -library we are using as practical framework represents a finite set $T \subset \mathcal{U} \subset \mathbb{Z}$ using so-called *range sequences* as defined by Schulte and Carlsson in [32, sect.2.2] and by Schulte and Tack in [35] as follows:

Definition 22 (Range sequence)**.** *Given a finite set $T \subset \mathcal{U} \subset \mathbb{Z}$, a* range sequence *is the shortest sequence $s = \{[n_1..m_1], \ldots, [n_k..m_k]\}$ such that $T = \bigcup_{i=1}^{k} [n_i..m_i]$.*

Clearly, the definition of a range sequence implies that a range sequence s for a given finite set $T \subset \mathcal{U}$ is a partition of T into disjoint maximal intervals $s_i \in s$. The underlying data structure for a range sequence in *Gecode* consists in a doubly-linked list, where a list item corresponds to a maximal interval $s_i \in s$. Considering a range sequence s of length $|s| = n$, each item $s_k \in s, k \in \{1, \ldots, n-1\}$ maintains a pointer to the preceding item s_{k-1} (`prev`) and a pointer to its succeeding item s_{k+1}(`next`). The list items s_0 and s_n maintain a pointer to their immediate successor or predecessor respectively and are marked by a `first` (s_0) and a `last` pointer (s_n) such that the interval bounds of s_0, $\underline{s_0}$ and $\overline{s_0}$, as well as the interval bounds of s_n, $\underline{s_n}$ and $\overline{s_n}$, can be accessed in constant time $\mathcal{O}(1)$.
Additionally, the list maintains a counter `size` summing up the cardinalities of all its items, such that the set size $|s| = n$ can also be queried in constant time $\mathcal{O}(1)$. Keeping in mind the doubly-linked list representation of a range sequence s we also refer to s as *range-list* and write $RL(T)$ for the range-list $r = RL(T)$ representing the set T.

Example 2 (A simple range-list)**.** *Consider the set $T = \{-10, -4, -3, 1, 4, 7, 14, 19\}$. The corresponding range sequence $s = RL(T)$ for the set T is $s = \{[-10..-4], [-3..1], [4..7], [14..19]\}$. The respective range-list constructed from s is depicted in figure 3.2.*

Interval representation Provided the range-list representation of a single finite set T described in the previous paragraph the data structure for the approximate cardinality set bounds domain $E \in \left(\mathcal{C}\right)$ of a set variable $x : E$ is consequently assembled by a tuple

$$SV(x) = \langle \mathtt{glb}, \mathtt{lub}, \mathtt{C} = [l..r] \rangle$$

where `glb` and `lub` are two range lists denoting the greatest lower bound $\lfloor E \rfloor$ and $\lceil E \rceil$ respectively and an additional integer interval $\mathtt{C} = [l..r]$ restricting the cardinality of the domain such that $\underline{\mathtt{C}} = l \leq |\lfloor E \rfloor|$ and $|\lceil E \rceil| \leq \overline{\mathtt{C}} = r$.

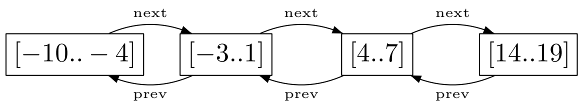

Figure 3.2: range-list representing $\{-10, -4, -3, 1, 4, 7, 14, 19\}$

Example 3. *Consider a set variable* $x : E[\{4,5\}..\{1,2,4,5,7,8,9\}]$. *The corresponding set variable representation* $SV(x) = \langle\{[4..5]\}, \{[1..2], [4..5], [7..9]\}, [2,6]\rangle$ *is shown in figure 3.4.*

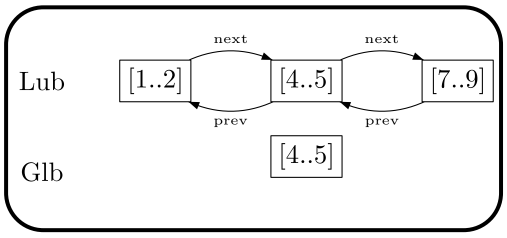

Figure 3.3: Representing set variable $x : E = [\{4,5\}..\{1,2,4,5,7,8,9\}]$

In the following paragraphs we discuss the propagation services that the representation $SV(x)$ of a set variable $x : E$ with corresponding approximate domain $E \in \left(\mathcal{C}\right)$ is supposed to provide.

Domain lookup Concerning domain lookup operations on $SV(x)$, that is accessing information provided by the underlying data structure, we can obviously query the size of the bounds $\|\lfloor E\rfloor\|$,$\|\lceil E\rceil\|$ as well as their smallest and largest elements in constant time $\mathcal{O}(1)$. Moreover, the computation for the size of the set difference $\left|\Delta \overset{\text{def}}{=} \lfloor E\rfloor \setminus \lceil E\rceil\right|$, accessing cardinality restrictions for the set variable $\underline{\mathsf{C}}$, $\overline{\mathsf{C}}$ as well as checking whether the variable x is assigned are also constant time operations, where testing for x being assigned is performed by testing whether $|E| = 1$ which is exactly the case if $\lfloor E\rfloor$ and $\lceil E\rceil$ coincide, that is if $\|\lfloor E\rfloor\| = \|\lceil E\rceil\|$ holds since $\lfloor E\rfloor \subseteq \lceil E\rceil$.However, testing whether a value $v \in \mathbb{Z}$ definitively belongs to the domain of x ($v \in \lfloor E\rfloor$, compare (3.4)) or is definitively excluded from the variables domain ($x \notin \lceil E\rceil$, compare (3.5)) clearly depend on the size of the bounds, that is $\mathcal{O}(\|\lfloor E\rfloor\|)$ for the former and $\mathcal{O}(\|\lceil E\rceil\|)$ for the latter case. Finally, the iterator operations needed to iterate over $\lfloor E\rfloor$ and $\lceil E\rceil$ as presented in the range-iterator architecture introduced in [35], namely testing whether we have already iterated the whole list (referred to as `done`) and following one item in the range-list to its successor (referred to as `incr`) are performed in $\mathcal{O}(1)$.

Domain update Reconsidering the process from the model of a constraint problem to its solution as sketched in chapter 2 domain update operations on $SV(x)$ are the result of domain modifications either caused by constraint propagation or by branching. More explicitly, domain update operations are performed by basic constraints as highlighted by Schulte in [31, chap. 2.1], where we denote the approximate domain resulting from updating $E = [\text{glb}..\text{lub}]$ as $F = [\text{glb}'..\text{lub}']$. Assuming a finite set $G \in \mathcal{U}$ we focus on $G \subseteq x$ and $x \subseteq G$ as basic set constraints. According to Gervet [16, chap.4.2.4] we compute $G \subseteq x$ as $\lfloor E' \rfloor \overset{\text{def}}{=} \lfloor E \rfloor \cup G$ and $x \subseteq G$ as $\lceil E' \rceil \overset{\text{def}}{=} \lceil E \rceil \cap G$. These updates are in turn computed using the *range iterator* architecture as proposed by Schulte and Tack in [35]. Obviously, the worst case for these updates occurs in case $|RL(G)| > 1$. In this case, the update of glb $= RL(\lfloor E \rfloor)$ is essentially achieved by setting glb′ to glb′ $= RL(\lfloor E \rfloor) \cup RL(G)$ where the time needed to perform this update is $\mathcal{O}(|RL(\lfloor E \rfloor)| + |RL(G)|)$ if $\lfloor E \rfloor \parallel G$, that is $\lfloor E \rfloor$ and G are disjoint. Updating lub to lub′, such that lub′ $= RL(\lceil E \rceil) \cap RL(G)$ is performed in at most $\mathcal{O}(|RL(\lceil E \rceil)| + |RL(G)|)$, where the worst case can for example occur if we have that $\lceil E \rceil \parallel G$ and additionally either $\min(E) \leq \min(G) \wedge \max(G) \leq \max(E)$ or $\min(G) \leq \min(E) \wedge \max(E) \leq \max(G)$. However, we note that in case $|RL(G)| = 1$ the time needed to update glb already shrinks to $\mathcal{O}(|RL(\lfloor E \rfloor)|)$. Similarly the worst case complexity for the update of lub decreases to $\mathcal{O}(|RL(\lceil E \rceil)|)$.

Caveat Though the just mentioned worst case complexity is not affected the chosen representation of two separate bounds glb $= RL(\lfloor E \rfloor)$ and lub $= RL(\lceil E \rceil)$ where $\lfloor E \rfloor \subseteq \lceil E \rceil$ leads to the following caveat concerning domain update operations: in addition to the update operations $G \subset x$ and $x \subset G$ we always have to assert that the updated bounds $\lfloor E' \rfloor$ and $\lceil E' \rceil$ still satisfy the partial order that $\lfloor E' \rfloor \subseteq \lceil E \rceil$ or $\lfloor E \rfloor \subseteq \lceil E' \rceil$, that is whether values to be included in $\lfloor E \rfloor$ are allowed by $\lceil E \rceil$ and that values to be excluded from $\lceil E \rceil$ are not required by $\lfloor E \rfloor$. In terms of complexity, each update operation requires additional time $o(|RL(\lfloor E \rfloor)| + |RL(\lceil E \rceil)|)$ which can immediately decrease to $o(|RL(\lceil E \rceil)|)$ in case that either $|RL(\lfloor E \rfloor)| = 1$ or $|RL(\lfloor E \rfloor')| = 1$.

3.2.2 Disjoint bounds

An implementation avoiding the subset test issue mentioned in section 3.2.1 can be found in the set variable implementation of *Cardinal* [6] which, given an approximate domain $E \in \left(\mathcal{C}\right)$ as state above, represents $SV(x)$ using the sets $\lfloor E \rfloor$ and $\Delta \overset{\text{def}}{=} \lceil E \rceil \setminus \lfloor E \rfloor$ instead of representing $\lfloor E \rfloor$ and $\lceil E \rceil$ as explained in the previous section. For the remainder of this section we refer to this representation as *disjoint bounds* representation abbreviated as DB representation in contrast to the *included bounds* representation abbreviated as IB representation, where IB refers to the representation we discussed above involving $\lfloor E \rfloor$ and $\lceil E \rceil$. However, from [6] it does not become clear, whether the underlying data structure for the representation of the set bounds builds upon sorted value lists, some bitmaps or bit vectors or even something similar to the range-list data structure as introduced in the previous section. Another alternative presented by Gervet and Hentenryck in [19] exploits the finiteness of the universe in discourse, $\mathcal{U} \subset \mathbb{Z}$, and represents

Operations	IB representation		DB representation	
Domain lookup				
	best case	worst case	best case	worst case
$\lvert\lfloor E\rfloor\rvert$,$\lvert\lceil E\rceil\rvert$,$\lvert\lceil E\rceil \setminus \lfloor E\rfloor\rvert$		$\mathcal{O}(1)$		$\mathcal{O}(1)$
min($\lfloor E\rfloor$),max($\lfloor E\rfloor$)		$\mathcal{O}(1)$		$\mathcal{O}(1)$
min($\lceil E\rceil$),max($\lceil E\rceil$)		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$
$l \leq \lvert x\rvert$,$\lvert x\rvert \leq r$		$\mathcal{O}(1)$		$\mathcal{O}(1)$
$\lvert\lfloor E\rfloor\rvert = \lvert\lceil E\rceil\rvert$?		$\mathcal{O}(1)$		$\mathcal{O}(1)$
$v \in \lfloor E\rfloor$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lfloor E\rfloor\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$
$v \in \lceil E\rceil$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$
$RL(\lfloor E\rfloor).incr()$		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$
$RL(\lceil E\rceil).incr()$		$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$
Domain update				
	best case	worst case	best case	worst case
$\lfloor F\rfloor \overset{\text{def}}{=} \lfloor E\rfloor \cup G$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lfloor E\rfloor\rvert + \lvert RL(G)\rvert + \lvert\lceil E\rceil\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$
$\lceil F\rceil \overset{\text{def}}{=} \lceil E\rceil \cap G$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert + \lvert RL(G)\rvert + \lvert\lceil E\rceil\rvert)$	$\mathcal{O}(1)$	$\mathcal{O}(\lvert\lceil E\rceil\rvert)$

Table 3.1: Comparison complexities domain lookup and update for IB and DB

the set variable domain using $\lfloor E\rfloor$ and $R \overset{\text{def}}{=} \overline{\lceil E\rceil} = \mathcal{U} \setminus \lceil E\rceil$. We refer to this latter approach as the *complement bounds* representation which we abbreviate as CB representation. Since one task of this thesis was to investigate whether the IB approach 3.2.1 used in *Gecode* can be optimized by changing the variable representation we discuss in this section the differences between IB and DB , which has been implemented in *Gecode* and compared to the current IB representation. In contrast to the IB representation (see 3.2.1) we now represent the approximate domain $E \in \left(\mathcal{C}\right)$ of a set variable $x : E$ by a tuple

$$SV(x) = \langle \texttt{cdb}, \texttt{C} = [l..r]\rangle$$

where `cdb` is a single range-list denoting the disjunctive union $\lfloor E\rfloor \uplus \Delta$, where $\Delta \overset{\text{def}}{=} \lceil E\rceil \setminus \lfloor E\rfloor$, that is the lower interval bound $\lfloor E\rfloor$ is only represented once and the union of $\lfloor E\rfloor$ and Δ again yields the upper interval bound $\lceil E\rceil = \lfloor E\rfloor \cup \Delta$. Additionally, the implementation of `cdb` uses a range-lists, where each interval contains an additional flag `inglb`, denoting, whether the range $[n_i..m_i]$ in the list belongs to the lower bound $\lfloor E\rfloor$ or to the upper bound $\lceil E\rceil$. Obviously, the change of the representation comes along with a relaxation of the range sequence definition (see definition 22) as a range-list or a range sequence is not necessarily of minimal size if it encodes both, the lower and the upper interval bound as clarified in the following example:

Example 4. *Consider the set variable* $x : E\,[\{4,5\}..\{1,2,3,4,5,6,7,8,9\}]$. *The corresponding set variable representation using the DB representation is* $SV(x) = \langle\{[1..3], [4..5], [6..9]\}, [2, 6]\rangle$ *is shown in figure 3.4.*

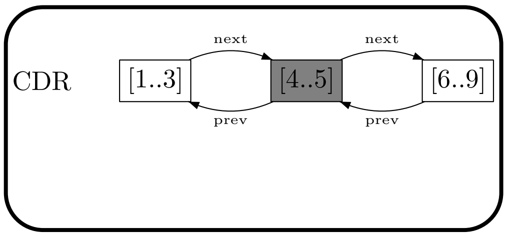

Figure 3.4: Representing set variable $x : E = [\{4,5\}..\{1,2,3,4,5,6,7,8,9\}]$. The gray-colored item denotes $\lfloor E \rfloor$ and the non-colored items denote $\lceil E \rceil$

Domain lookup Taking a glance at the domain lookup operations on $SV(x)$, that is accessing information provided by the underlying data structure, we still can query the size of the bounds $\left|\text{glb}\right|$,$|\text{lub}| \overset{\text{def}}{=} \left|\text{glb}\right| + |\Delta|$ in constant time $\mathcal{O}(1)$. As we fixed the lookup algorithms to start with the first item in the list `cdr` querying for the smallest and largest elements of $\lceil E \rceil$ is still a constant time operation. Finding the smallest and largest element of the lower bound, $\lfloor E \rfloor$, however has a worst case of $\mathcal{O}(|RL(\lceil E \rceil)| - 1)$. As for the computation of the size of the set difference $|\Delta|$ this is a straightforward constant time operation as $SV(x)$ keeps track of $||\lfloor E \rfloor||$ and $|\Delta|$. Moreover, accessing cardinality restrictions for the set variable $\underline{\mathtt{C}}$, $\overline{\mathtt{C}}$ as well as checking whether the variable x is assigned are still constant time operations, where testing for x being assigned is performed by testing whether $|\Delta| = 0$ which is exactly the case if $\lfloor E \rfloor$ and $\lceil E \rceil$ coincide. However, testing whether a value $v \in \mathbb{Z}$ definitively belongs to the domain of x ($v \in \text{glb}$, compare (3.4)) or is definitively excluded from the variables domain ($x \notin \text{lub}$, compare (3.5)) clearly depend on the size of the bounds, that is $\mathcal{O}(|\mathtt{cdb}|)$ for the former and also for the latter case. Finally, the iterator operations needed to iterate over lub, namely testing whether have already iterated the whole list (referred to as `done`) and following one item in the range-list to its successor (referred to as `incr`) are performed in $\mathcal{O}(1)$. Iterating over glb however the test whether iteration has finished as well as the increment operation for the iterator are in $\mathcal{O}(|\mathtt{cdb}|)$.

Domain update Analogously to section 3.2.1 we denote the approximate domain resulting from updating $E = [\text{glb}..\Delta]$ as $F = [\text{glb}'..\Delta']$ and we focus on the same basic constraints. Obviously, the worst case for these updates occurs in case $|RL(G)| > 1$. In this case, the update of $\text{glb} = RL(\lfloor E \rfloor)$ is essentially achieved by setting glb′ to $\text{glb}' = RL(\lfloor E \rfloor) \cup RL(G)$ where the time needed to perform this update is $\mathcal{O}(|RL(\lceil E \rceil)| + |RL(G)|)$, where every value to be included in the lower bound has to be an element of Δ. Updating lub to lub′, such that $\text{lub}' = RL(\lceil E \rceil) \cap RL(G)$ is computed in at most $\mathcal{O}(|RL(\lceil E \rceil)| + |RL(G)|)$. But as recently noticed, we can check in constant time $\mathcal{O}(1)$, whether $G \subset \lfloor E \rfloor$ or not and if so only need to compute the intersection with intervals contained in $\lceil E \rceil$, where iteration skips items in $\lfloor E \rfloor$ in constant time. Thus, we obtain a worst case complexity of $\mathcal{O}(|RL(\Delta)|) = \mathcal{O}(|RL(\lceil E \rceil)|)$. However, we note that in case $|RL(G)| = 1$ the time needed to update glb already shrinks to $\mathcal{O}(|RL(\lceil E \rceil)|)$. Similarly the worst case complexity for the update of lub decreases to $\mathcal{O}(|RL(\lceil E \rceil)|)$.

Caveat The change from IB to DB representation comes along with a loss of constant time operations among the provided domain lookup operations as we can see in table 3.1. Moreover, we spend clearly more time on domain iteration than in the IB case. This will become clear from the evaluation of those two implementations in section 6.4.

3.2.3 Full domain representation $(\mathcal{F})$

The main principle behind the full domain approximation $(\mathcal{F})$ consists in the equivalence of a domain D and its characteristic function χ_D as underlined in section 3.1.2. In the remainder of this section we discuss the approach formulated by Lagoon and Stuckey in [24] to use reduced and ordered binary decision diagrams as abstract representation of a set variable domain. Hence, we first of all make a digression to the area of *binary decision diagrams*.

Binary Decision Diagrams

Recapitulating the notion of a binary decision diagram we provide a definition as presented by Bryant in [10]:

Definition 23 (Binary Decision Diagram (BDD))**.** *A* binary decision diagram (BDD) *represents a Boolean function by a directed acyclic graph* $\overrightarrow{G} := \langle V, E \subseteq V \times V \rangle$ *with one vertex* $r \in V$ *called* root *that has only two outgoing edges such that all vertices* $v \in S \subset V \setminus \{r\}$ *that have no outgoing edge are labeled with* 0 $(\bot)$ *or* $1(\top)$ *and all vertices* $v \in V \setminus S$ *are labeled with a Boolean variable* x_v *having exactly two outgoing edges* $e_1 = (v, t)$ *and* $e_2 = (v, f)$ *labeled with* 1 *and* 0 *respectively. We will refer to the latter vertices as internal vertices, written* $v = (x_v, t, f)$*. Note that for the remainder of this section edges labeled with* 1 *are depicted using* solid edges *and edges labeled with* 0 *are represented by* dashed edges.

A path p from the root node r to a vertex $v \in S$ determines a variable assignment for the Boolean variables along the path, such that the function value for the assignment on the path p is represented in the node v. Furthermore, we call a BDD b is *reduced* according to Hawkins *et al.*[22, chap 2.4] if there are no identical nodes. Let BV denote the set of Boolean variables the nodes $v \in VS$ use as label set. Then a BDD b is *ordered* with respect to a given variable order $\preceq \subseteq BV \times BV$ if $\forall v_1, v_2 \in BV : (\exists e = (v_1, v_2)) \Leftrightarrow v_1 \preceq v_2$. Given a reduced and ordered binary decision diagram b Bryant[10] and Hawkins *et al.*[22] state that b is a canonical function representation up to reordering of BV.

Single set representation In order to represent a finite set $T \subset \mathcal{U} \subset \mathbb{Z}$, $|T| = n$ using the ROBDD-based approach by Hawkins *et al.*, T is associated with a vector v of Boolean variables. Each Boolean variable v_i,$i \in \{l = \min(T) - \min(T), \ldots, r = \max(T) - \min(T)\}$

represents an element of the set T such that we obtain a Boolean variable v_i for every value $v \in [\min(T)..\max(T)]$. Following Hawkins *et al.* the caracteristic function of the set models yields an assignment for the Boolean variables such that $(\chi_T(v) = 1 \Leftrightarrow v \in D \Leftrightarrow x_{v-\min(T)} = 1) \wedge (\chi_T(v) = 0 \Leftrightarrow v \notin D \Leftrightarrow x_{v-\min(T)} = 0)$. Thus we can represent the set T by the following Boolean formula Γ[22, chap. 3.1]:

$$\Gamma \overset{\text{def}}{=} \bigwedge_{i=0}^{n-1} = \begin{cases} x_{v-\min(T)} & \text{if } v \in T \\ \neg x_{v-\min(T)} & \text{else} \end{cases}$$

Example 5 (A single set)**.** *Consider the set $T = \{1, 3, 5\}$. The corresponding Boolean vector $b = \langle v_0, \dots, v_4 \rangle$ together with the characteristic function χ_T yields the formula $\Gamma = v_0 \wedge \neg v_1 \wedge v_2 \wedge \neg v_3 \wedge v_4$ which is represented by the ROBDD in figure 3.5.*

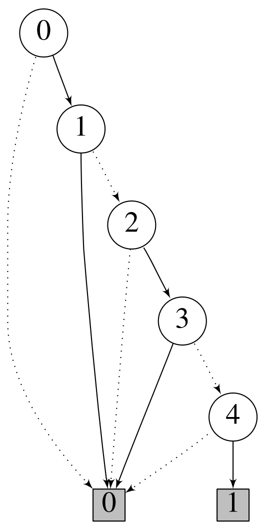

Figure 3.5: ROBDD for set {1,3,5}

Domain representation Apparently an approximate domain $E \in \left(\mathcal{F}\right)$ of a set variable $x : E$ denotes the set of all sets $e \in \mathtt{Val} = \mathcal{P}(\mathcal{U})$ the variable x can take. As stated in [22, sect. 3.1] E is consequently captured as disjunction of the corresponding formulae for all $e \in E$. Thus, we obtain the following data structure for a set variable x:

$$SV(x) = \langle v \overset{\text{def}}{=} \langle v_0, \dots, v_{\max(\lceil E \rceil)-\min(\lceil E \rceil)} \rangle, \mathtt{domain} \rangle$$

where v denotes the vector of Boolean variables for all elements $e \in [\min(\lceil E \rceil)..\max(\lceil E \rceil)]$ and `domain` represents a ROBDD over the Boolean variables contained in v reflecting the current domain of the variable x.

Modeling Advantage The major gain from the ROBDD presentation as developed by Hawkins *et al.* in [22] is the fact, that the ROBDD representation is able to encode any Boolean function. As we are not only able to model the domains of set variables as functions over Boolean formulae using the respective characteristic functions of the domains but also to model constraints themselves as Boolean formulae, constraints can be encoded as ROBDDs. Consequently, the complexity of domain lookup and domain update operations are uniquely determined by the complexity of ROBDD operations as every operation can be expressed in terms of a conjunction of logical formulae. As highlighted in [22, chap. 2.5] for two ROBDDs R_1 and R_2, the worst case complexity for an operation $R_1 \diamond R_2$, such that $\diamond \in \{\wedge, \vee, \Leftrightarrow\}$ are $\mathcal{O}(|R_1| \cdot |R_2|)$, where $|R_i|$ denotes the number of vertices in the ROBDD R_i. Negating an ROBDD R has linear worst case complexity $\mathcal{O}(|R|)$ in the size of the ROBDD and existential quantification $\exists v.R$ has even a quadratic worst case complexity $\mathcal{O}\left(\left|R^2\right|\right)$. However, the test whether R_1 and R_2 are equivalent, that is $R_1 \Leftrightarrow R2$, can be done in $\mathcal{O}(1)$.

Order Issue Obviously, the above listed complexities all depend on the size of a single ROBDD or of two ROBBDs. However, as Bryant puts it in [10], the "form and size of the [R]OBDD representing a function depends on the variable ordering"(sic) which was the requirement for an ROBDD to be ordered (compare sec. 3.2.3). The same fact is underlined by Stuckey et. al. in [22]. First they assume that we are given a set of set variables $\mathcal{V} = \{v_1, \ldots, v_m\}$ with associated vectors of Boolean variables $\langle v_{i,1}, \ldots, v_{i,N} \rangle$, where $i \in \{1, \ldots, m\}$ and $N \overset{\text{def}}{=} |\mathcal{U}|$. Then they fix the variable order ($\prec$) for as

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

. This order *guarantees* a linear representation for the ROBDDS resulting from the propagation of certain constraints [24] hence classify as primitive except constraints restricting the cardinality of the set variable x which is also directly encoded into `domain` in terms of an ROBDD.

Domain lookup Concerning domain lookup operations on $SV(x)$ for a set variable $x : E$, that is accessing information provided by the underlying data structure, we can query the size of the bounds $|\lfloor E \rfloor|$,$|\lceil E \rceil|$ as well as their smallest and largest elements in time proportional to the size of the ROBDD `domain`, $\mathcal{O}(|\texttt{domain}|)$. Similarly, the computation for the size of the set difference $\left|\Delta \overset{\text{def}}{=} \text{lub} \setminus \text{glb}\right|$, accessing cardinality restrictions for the set variable $\underline{\mathtt{C}}$, $\overline{\mathtt{C}}$ as well as checking whether the variable x is assigned depend also on the size of the represented ROBDD `domain` operations, where testing for x being assigned is performed by testing whether there is only one path p in the ROBDD `domain`, such that all Boolean variables from the associated vector b are contained on p. However, testing whether a value $v \in \mathbb{Z}$ definitively belongs to the domain of x ($\texttt{domain} \wedge \neg v \Leftrightarrow \bot$) or is definitively excluded from the variables domain ($\texttt{domain} \wedge v \Leftrightarrow \bot$) clearly depend on the size of the represented ROBDD `domain`, $\mathcal{O}(|\texttt{domain}|)$.

4 Cross-domain propagation using variable views

If it looks like a duck, and quacks like a duck, we have at least to consider the possibility that we have a small aquatic bird of the family anatidae on our hands.
— Adams

Based on Benhamou's work in [9] chapter 2 introduced a formal model of domain approximations as appropriate representations for domains of finite domain variables in a constraint solver. Applying this framework, we furthermore surveyed existing set variable representations as suggested in literature (see chap. 3) and the theories behind them. Recalling the possible implementations for the cardinality set bounds approximation $\left(\mathcal{C}\right)$ and the full domain approximation $\left(\mathcal{F}\right)$ we sketched in the previous section (compare sec. 3.2) this chapter discusses how the propagation services offered by the implemented domain approximations are made available for the propagators requiring them through the concept of *variable views* as introduced in [35], which we simply refer to as *views*.

4.1 Abstraction through encapsulation

As the figure 4.1 clarifies, the notion of a view in our framework is twofold. Assume that we are encoding a problem into a CSP whose specification involves some set variable $x : D$, that is $D \in Dom$. On the one hand we use a view $V_{Dom \to A}$ as an adaptor (see def. 8) to map the domain D to an approximate domain $V_{Dom \to A}(D) = E \in A$, where E is encapsulated in the domain implementation Imp_A in the constraint solver. Hence, a view $V_{Dom \to A}$ is used as mapping prescribing the internal representation of a set variable's domain. Apart from mapping a domain to its internal representation we further exploit the adaptor functionality of a view $V_{A \to B}$ in order to forward the propagation services from the implementation Imp_A to a propagator $p_B : B^n \to B^n$ by providing an interface $F = V_{A \to B}(E)$ to p_B. According to [35, sect.3] the simplest example for views is the *identity*-view $V_{id} \overset{\text{def}}{=} V_{A \to A}$ obtained by instantiating the view B in the above figure 4.1 to A forwarding propagation services from a domain $E = V_{Dom \to A}(D)$ to a propagator p_A reasoning over the same domain approximation A through the propagation interface $F = V_{A \to A}(E)$. Recapitulating the definition of a domain approximation as presented in chapter 3.1 we conclude that, provided domain approximations $A, B \subseteq Dom$, we obtain the appropriate view $\Gamma_B : A \to B$ mapping a domain $D \in A$ to an approximate domain $E \in B$ by

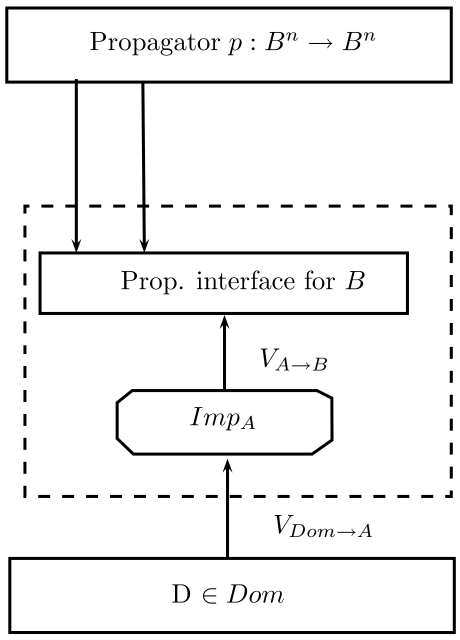

Figure 4.1: Views as propagation interface

choosing B as follows:

$$B = \left(\mathcal{S}\right) \qquad \Gamma_{\left(\mathcal{S}\right)}(D) \overset{\text{def}}{=} \left[\bigcap_{d \in D} d .. \bigcup_{d \in D} d\right]_{\subseteq} \tag{4.1}$$

$$B = \left(\mathcal{C}\right)_l^r \qquad \Gamma_{\left(\mathcal{C}\right)_l^r}(D) \overset{\text{def}}{=} \Gamma_{\left(\mathcal{S}\right)}(D) \cap \left\{ s \in \left(\mathcal{S}\right) \mid l \le \|\lfloor s \rfloor\| \wedge \|\lceil s \rceil\| \le r \right\} \tag{4.2}$$

$$B = \left(\mathcal{L}\right) \qquad \Gamma_{\left(\mathcal{L}\right)}(D) \overset{\text{def}}{=} [\uparrow D .. \downarrow D]_{\le} \tag{4.3}$$

$$B = \left(\mathcal{LL}\right) \qquad \Gamma_{\left(\mathcal{LL}\right)}(D) \overset{\text{def}}{=} [\uparrow D .. \downarrow D]_{\ll} \tag{4.4}$$

$$B = \left(\mathcal{F}\right) \qquad \Gamma_{\left(\mathcal{F}\right)}(D) \overset{\text{def}}{=} D \tag{4.5}$$

where the view $\Gamma_{\left(\mathcal{S}\right)}$ in equation (4.1) is exactly the *convex closure* operator as defined in [16, Def. 38]. Concluding, this paragraph indicates how variable views are used as propagation interface, abstracting the set variable domains to approximate domains as presented in chapter 3. Thus the presence of views in the architecture of a constraint solver not only enables us to change the underlying implementation of a domain representation without altering the propagation algorithm, but also compensates missing propagators $p : B^n \to B^n$ for a domain approximation B by adapting the propagation interface from $V_{B \to B}$ to $V_{B \to A}$ to fit another propagator $p : A^n \to A^n$

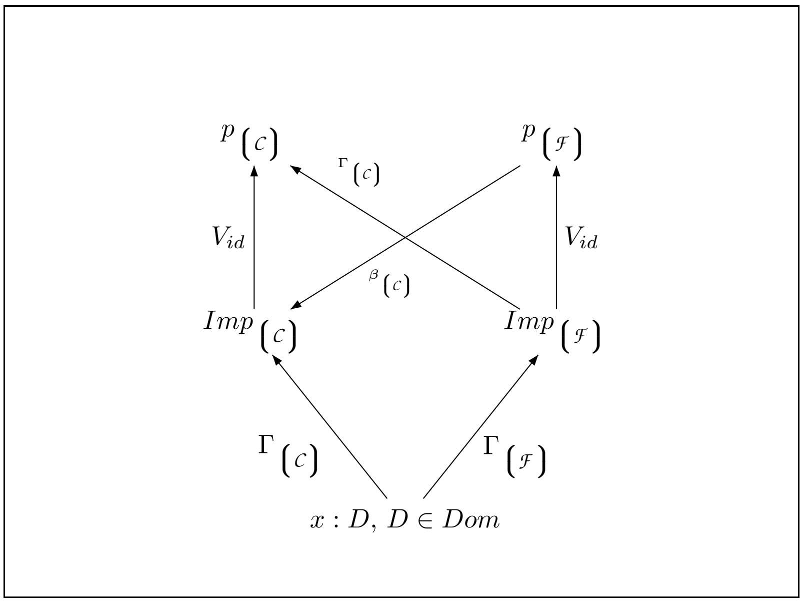

Figure 4.2: Integrating $(\mathcal{C})$ and $(\mathcal{F})$ in a constraint solver with views

serving as replacement. Hence, "variable views [clearly] yield a higher level of propagator abstraction" as Schulte and Tack state in [35].

4.2 One interface to integrate them all

Consequently the main advantage emerging from variable views consists in the possibilty of integrating different levels of consistency, that is different domain approximations, into one constraint solver. Consider the finite set component of the *Gecode* library following the basic scheme as shown on the left-hand side of figure 4.2, where we fix a standard domain representation, namely $(\mathcal{C})$, as set variable representation (see section 3.2) and define propagators $p : (\mathcal{C})^n \rightarrow (\mathcal{C})^n$ working on this approximation through identity views $V_{id} \in (\mathcal{C}) \rightarrow (\mathcal{C})$. Consequently, the *Gecode* component for finite set variables is $bounds((\mathcal{C})) - consistent$ in the strong sense (compare section 3.1.2). If we focus on the other hand on the finite set component of the constraint solver based on the full domain approximation using ROBDDs as proposed by Hawkins *et al.* in [22] the right-hand side of figure 4.2 explains that we can basically model the ROBDD-based finite set component of their constraint solver using the full domain approximation $(\mathcal{F})$ as representation, where identity views $V_{id} \in (\mathcal{F}) \rightarrow (\mathcal{F})$ provide the propagation services for propagators $p : (\mathcal{F})^n \rightarrow (\mathcal{F})^n$ working on this representation. Hence, the ROBDD-approach yields a domain-consistent representation of set variables in our setting.

Obviously, an implementation of both approaches in a single constraint solver would simply coexist without any further integration into the system. However, using the *Gecode* framework for our implementation enables us to use the above discussed views forming one of the integral architectural components of the *Gecode* library.

4.2.1 Crossing borders of consistency

As a consequence we are able to apply the view $\Gamma_{(\mathcal{C})}$ to an approximate domain $F \in (\mathcal{F})$ of a set variable x encoded in $Imp_{(\mathcal{F})}$ and obtain a bounds($(\mathcal{C})$)-consistent interface $G = \Gamma_{(\mathcal{C})}(F)$ allowing bounds($(\mathcal{C})$)-consistent propagators to operate on the domain consistent representation $Imp_{(\mathcal{F})}$ through G as shown in 4.2. As Hawkins *et al.* propose in [22, sect. 6.2] for the ROBDD implementation of F we can use $\Gamma_{(\mathcal{C})}$ to compute the cardinality set bounds of F as follows: Since F is represented as a ROBDD $domain{=}\overrightarrow{G} := \langle V, E \subseteq V \times V\rangle$ with associated boolean vector $b = \langle b_0, \ldots, b_{b-a}\rangle$ ranging over $\mathcal{P}([a..b])$, $\lceil F \rceil \subseteq [a..b]$ values e_i belonging to $\lfloor F \rfloor$ are identified as internal vertices $v_i \in V$, $i \in \{0, \ldots, (b-a)\}$ such that: $v_i = (b_i, t, \bot)$, that is the corresponding boolean variable b_i for the value e_i is true in every assignment represented by *domain* and hence is contained in $\lfloor F \rfloor$. Analogously, the values e_i belonging to $\overline{\lceil F \rceil} \overset{\text{def}}{=} [a..b] \setminus \lceil F \rceil$ are identified as internal vertices $v_i \in V$ such that: $v_i = (b_i, \bot, f)$, that is b_i is false in every assignment and hence $e_i \notin \lceil F \rceil$. Using the initial domain information $[a..b]$, we finally obtain the least upper bound of F as $\lceil F \rceil = [a..b] \setminus \overline{\lceil F \rceil}$. Hawkins *et al.* stress in [22, sect. 6.2] that those internal vertices required to compute the set bounds can be identified in $\mathcal{O}(domain)$, what we achieve by iteration over the ROBDD. Assume that we have identified $k \leq |domain|$ vertices determining the set bounds of the variable x. Computing the conjunction of those k variables we obtain $\Gamma_{(\mathcal{S})}(F)$ in time $\mathcal{O}(k)$, since we insert the identified variables in descending ROBDD order (see 3.2.3). Still we need the cardinality information from the domain to finally arrive at $\Gamma_{(\mathcal{C})}(F)$. We obtain the lacking cardinality information, that is the interval $C = [l, r]$ such that $l \leq |F| \leq r$, using the `bdd_count_cardinality` algorithm of Hawkins *et al.* as stated in [22, sect. 6.4]. Given the representation of F as tuple $\langle b, domain\rangle$ the time needed to extract the cardinality information is denoted as $\mathcal{O}(|b| \cdot |domain|)$. Since k, the number of extracted variables determining $\lfloor F \rfloor$ and $\overline{\lceil F \rceil}$, is bounded from above by $\mathcal{O}(domain)$ the overall time complexity to compute $\Gamma_{(\mathcal{C})}(F)$ is:

$$\mathcal{O}(|domain| + |b| \cdot |domain|) \quad = \quad \mathcal{O}((|b| + 1) \cdot |domain|) \tag{4.6}$$

Note that a variable view not only forwards propagation services from the approximate domain F to a propagator $p : (\mathcal{C})^n \to (\mathcal{C})^n$ but also effects the inverse direction by updating F according to the modifications p performs on the propagation interface $\Gamma_{(\mathcal{C})}(F)$. Hence the above complexity from equation (4.6) not only denotes the worst case complexity for the creation of the propagation interface but also the worst case complexity for every domain modification that occurs on F through this interface during constraint propagation.

4.2.2 Weaken consistency of propagation

The very same view $\Gamma_{(\mathcal{C})}$ that we used in the previous paragraph to wrap a bounds($\left(\mathcal{C}\right)$)-consistent interface around an approximate domain $F \in \left(\mathcal{F}\right)$ can not only be used to interface domain approximations, but also to turn a domain consistent propagator $p : \left(\mathcal{F}\right)^n \to \left(\mathcal{F}\right)^n$ into a bounds($\left(\mathcal{C}\right)$)-consistent propagator $\beta_{(\mathcal{C})} : \left(\mathcal{F}\right)^n \to \left(\mathcal{C}\right)^n$. We achieve this as follows: let F be the x-component of the tuple $\overrightarrow{F}$ denoting the initial domains for the propagator p. Without changing the implementation $Imp_{(\mathcal{F})}$ we can impose a cardinality set bounds view $\Gamma_{(\mathcal{C})}$ as shown in the preceding paragraph. Doing so for all variables $x \in \mathtt{Var}$ we obtain initial domains $\overrightarrow{F}'$ as intermediate result, such that $\overrightarrow{F}'.x = G \overset{\text{def}}{=} \Gamma_{(\mathcal{C})}(F)$ is a cardinality set bounds representation of $\overrightarrow{F}$. Even more so, it follows from (3.9) that $G \in \left(\mathcal{C}\right) \Rightarrow G \in \left(\mathcal{F}\right)$. Hence, we can apply the domain consistent propagator p to $\overrightarrow{F}'$ and get a propagation result $\overrightarrow{G}$. Assuming that all domains are consistent with p we receive a propagation result $\overrightarrow{R} = p\left(\overrightarrow{F}\right)$, which is domain consistent. Instead of updating the variable domains we first apply the view $\Gamma_{(\mathcal{C})}$ again to R. As $\Gamma_{(\mathcal{C})}$ filters exactly the cardinality set bounds information from R, we obtain a bounds($\left(\mathcal{C}\right)$)-consistent propagation result $\overrightarrow{R}' = \Gamma_{(\mathcal{C})}(\overrightarrow{R})$. Thus, given initial approximate domains $\overrightarrow{F} \in \left(\mathcal{F}\right)^n$ and a domain-consistent propagator $p : \left(\mathcal{F}\right)^n \to \left(\mathcal{F}\right)^n$ we can build a bounds($\left(\mathcal{C}\right)$)-consistent propagator $\beta_{(\mathcal{C})} : \left(\mathcal{F}\right)^n \to \left(\mathcal{C}\right)^n$ by:

$$\beta_{(\mathcal{C})}(\overrightarrow{F}) \overset{\text{def}}{=} \Gamma_{(\mathcal{C})}\left(p\left(\Gamma_{(\mathcal{C})}\left(\overrightarrow{F}\right)\right)\right) \tag{4.7}$$

Apart from this bounds($\left(\mathcal{C}\right)$)-consistent propagator $\beta_{(\mathcal{C})}$ Hawkins *et al.* also propose a set bounds-consistent propagator $\beta_{(\mathcal{S})}$ such that

$$\beta_{(\mathcal{S})}(\overrightarrow{F}) \overset{\text{def}}{=} \Gamma_{(\mathcal{S})}\left(p\left(\Gamma_{(\mathcal{S})}\left(\overrightarrow{F}\right)\right)\right) \tag{4.8}$$

and a bounds($\left(\mathcal{L}\right)$)-propagator $\beta_{(\mathcal{L})}$,

$$\beta_{(\mathcal{L})}(\overrightarrow{F}) \overset{\text{def}}{=} \Gamma_{(\mathcal{L})}\left(p\left(\Gamma_{(\mathcal{L})}\left(\overrightarrow{F}\right)\right)\right) \tag{4.9}$$

which we can encode using the views $\Gamma_{(\mathcal{S})}$ and $\Gamma_{(\mathcal{L})}$ as described in section 4.1. Since we do not need to compute the cardinality bounds of the domains in $\overrightarrow{F}$ for the bounds($\left(\mathcal{S}\right)$)-consistent propagator $\beta_{(\mathcal{S})}$ in equation (4.8), the desired propagation interface $G = \Gamma_{(\mathcal{S})}(F)$ for one variable domain F can be obtained in $\mathcal{O}(|domain|)$. Computing the lexicographic bounds $E = [\inf(D)..\sup(D)]_{\leq}$ of a set variable $x : D$ represented as $SV(x) = \langle domain, \langle v_0, \ldots, v_{b-a}\rangle\rangle$ implies computing the conjunction of the two ground sets $\inf(D)$ and $\sup(D)$ where both of them can be encoded as a ROBDD of size $N = b - a + 1$. In order to obtain the corresponding ROBDDS $R(\inf(D))$ and $R(\sup(D))$ we use the algorithms `bdd_lex_lower_bound` and `bdd_lex_upper_bound` presented in [22, sec.6.5] whose complexity is $\Theta(b - a + 1)$.

Completing the picture As sketched in the preceding sections, variable views enable us to use propagators $p : \mathcal{A}^n \to \mathcal{A}^n$ on domain approximations $\mathcal{B}$ different from $\mathcal{A}$. Additionally they allow us to weaken the consistency of a propagator by restricting input and output domains to a propagation interface satisfying a consistency that is weaker than the propagator theoretically yields. Provided these two aspects and through the support of variable views in *Gecode* , we finally obtain a constraint solver for finite set variables that not only offers two different implementations for the underlying variable domain, but also integrates them in the system by allowing propagation across domain approximations, yielding a framework in which we can freely choose whether we want to apply bounds$\left(\mathcal{C}\right)$-consistency or domain-consistency to the variable domains. Thus, in order to complete the integration of both representations, we introduce these views. However, the view transforming a range-list cardinality set bounds approximate domain into a cardinality set bounds ROBDD *domain* such that a domain-consistent ROBDD-propagator could be applied, is not yet implemented.

5 Generating propagators

During the course of the last chapter we focused on the variable part of a constraint solver by turning our attention to different domain approximations in literature along with their definition (see chapter 3) and how they serve as abstract representation for variables in a constraint solver. We showed how constraint propagation is performed by executing propagators as constraint implementations on tuples of approximate domains (see chapter 2.2.1) representing the variables the constraints are defined on. Subsequently chapter 4 highlighted how to utilize variable views as integrating interface to different domain approximations like $\left(\mathcal{C}\right)$ and $\left(\mathcal{F}\right)$ in a single constraint solver forming a cross-approximation platform where propagators with different consistency-levels can operate on. Now, we have a look at propagators.

5.1 The propagator side of life

Concerning the propagator-side of a constraint solver we not only learned in section 2.2.1 what *constraint propagation* consists of but, we also know by now how basic set constraints are directly expressed in terms of domain lookup operations on the data structure implementing a set variable's approximate domain (compare sect. 3.2). This chapter builds on the research of Tack *et al.* who present in [37] how non-basic set constraints can be further abstracted using existential monadic second-order logic (∃MSO) "as a declarative specification language for finite set constraints"[37].

5.1.1 General description language - An intensional representation

As the previous paragraph highlights, the preceding chapters showed how we abstract a set variable $x : D$ in a constraint solver through its associated domain information $D \in Dom$, and how we use a domain approximation $\mathcal{A}$ to represent D by an approximate domain (see definition 7) $E \in \mathcal{A}$ in the constraint solver. Moreover chapter 2 points out an important aspect of the presented framework, namely that finite set constraints on finite set variables are defined extensionally (see definition 5). Due to this issue of representability (see equation (2.1)) a constraint solver uses propagators to implement an intensional representation of constraints. Following Tack *et al.* we discuss in the remainder of this chapter how we obtain this intensional representation using a fragment of ∃MSO as specification language [37], and how we apply this specification language in order to generate bounds($\left(\mathcal{C}\right)$)- and domain-consistent propagators from it. Since a constraint $c \in \mathcal{C}$ is defined in terms of a set of feasible assignments, we know from section 3.1.2 that we can represent c intensionally by its characteristic Boolean function

χ_c. This is exactly the point where the approach of Tack *et al.* comes in to express χ_c via an $\exists$MSO-formula ϕ as "compact representation"[37], where we define ϕ according to the $\exists$MSO-fragment specified in the grammar taken from section 3.1 of [37] augmented with existential first-order quantification as mentioned in [37, sect.6]:

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

Table 5.1: Syntax of a fragment of $\exists$MSO

In fact, a closer look at the grammar in table 5.1 clarifies that any finite set constraint c that we can represent using a formula $\phi_c \in \exists$MSO is a non-basic set constraint which is decomposed into basic element constraint $v \in x : D$ as Schulte points out [31, chap. 2.1]. Hence, this specification language indeed captures the characteristic function χ_c of the constraint yielding an intensional representation for c suitable as modeling language and denotational semantics for a sound propagator p_c a constraint solver uses to represent c intensionally.

Example 6. *Consider for example the constraint $c \equiv x \cap y = z$ on set variables $x, y, z \in \texttt{Var}$ constraining z to equal the intersection of x and y. Hence an extensional definition of c is $c = \{\alpha \in \texttt{Asn} \mid \alpha(x) \cap \alpha(y) = \alpha(z)\}$. Using $\exists$MSO as described in 5.1 we can define c in terms the $\exists$MSO-formula $\phi_c = \forall v.v \in x \wedge y \in y \Leftrightarrow v \in z$*

Limits of expressiveness Though using the presented $\exists$MSO-fragment as proposed by Tack *et al.* we are able to express any set constraint c by constructing the corresponding $\exists$MSO-formula ϕ_c there are still constraints that cannot be expressed in this framework. For instance we cannot express constraints like $c \equiv |x \cap y| = l$ involving some integer $l \in \mathbb{Z}$ as cardinality restriction on a non-basic constraint $x \cap y$. Instead we have to express c as a conjunction of $c' \equiv x \cap y = z$ and $d' \equiv |z| = l$ by using a variable $z \in \texttt{Var}$ which [22] refer to as *intermediate variable*. Then in turn we can abstract c' by its corresponding $\exists$MSO-formula $\phi_{c'}$ as shown in example 6 and immediately apply the basic cardinality constraint d' on z as depicted in **??**.

5.2 Propagators on $\left(\mathcal{C}\right)$

Let $x : D$ be a set variable whose domain is represented by an approximate domain $E = [\lfloor D\rfloor .. \lceil D\rceil]_{\subseteq} \in \left(\mathcal{C}\right)$ using cardinality set bounds domain approximation $\left(\mathcal{C}\right)$. From section 3.1.1 we know that given a ground set $G \subseteq \mathcal{U}$ a propagator p reasoning on set bounds can only perform two basic domain lookup operations, namely stating that G takes part in any assignment α for x or stating that any assignment α for x must be contained in G. Hence we obtain the two basic lookup operations of $G \subseteq x$ and $x \subseteq G$. Obviously, if we want to transform a ∃MSO-formula ϕ_c representing a constraint c over set variables $y \in \mathtt{Var}$, $y : D_y$ into a bounds($\left(\mathcal{C}\right)$)-consistent propagator $p_{\left(\mathcal{C}\right)}$ operating on the tuple of approximate domains $\overrightarrow{E} = \Gamma_{\left(\mathcal{C}\right)}\left(\overrightarrow{D_y}\right)$ we must transform ϕ_c into a set of domain lookup operations specifying for any variable y what values $v \in G$ any of its assignments contain and what values $v \in H$ its assignments are allowed to contain at all, that is every set variable $y \in \mathtt{Var}$ has to be updated such that the updated domain F of y is determined as $p\left(\overrightarrow{D_y}\right).x = F$. Thus, we obtain:

$$\forall y \in \mathtt{Var} : \left(\lfloor D_y \rfloor \cup G \subseteq y \subseteq \lceil D_y \rceil \cap H\right)$$

Therefore, as Tack *et al.* state in [37, sect.5.2], a translation from the ∃MSO-formula ϕ_c of a constraint c to a propagator $p : \left(\mathcal{C}\right)^n \to \left(\mathcal{C}\right)^n$ has not only to consider each variable $x : \mathtt{Var}$ the constraint c ranges over separately but has also to provide two ground sets G_x and L_x such that $G_x \subseteq x \subseteq L_x$.

5.2.1 From ∃MSO to $\left(\mathcal{C}\right)$

In fact Tack *et al.* define the sets G_x and L_x needed to perform inferences on the convex approximate domains of a given set variable $x \in \mathtt{Var}$ in terms of evaluated *range expressions* [37, sect.4.2] where range expressions are defined as follows:

Definition 24 (Range expression and evaluation)**.** *An expression R which is constructed according to the following grammar*

$$R \quad ::= \quad x \in \mathtt{Var} \;\mid\; \langle R\rangle \cup \langle R\rangle \;\mid\; \langle R\rangle \cap \langle R\rangle \;\mid\; \overline{\langle R\rangle} \;\mid\; \emptyset$$

Table 5.2: Grammar for range expressions on set variables

is called a range expression. *A range expression R is evaluated using functions r_{glb} and r_{lub} such that*

$$\begin{array}{ll} r_{glb}(\emptyset) = r_{lub}(\emptyset) = \emptyset & \\ r_{glb}(\overline{R}) = \overline{r_{lub}(R)} & r_{lub}(\overline{R}) = \overline{r_{glb}(R)} \\ r_{glb}(x) = \lfloor x \rfloor & r_{lub}(x) = \lceil x \rceil \\ r_{glb}(R_1 \diamond R_2) = r_{lub}(R_1 \diamond R_2) = r_{lub}(R_1) \diamond r_{glb}(R_2),\ \diamond \in \{\cup, \cap\} & \end{array}$$

Table 5.3: Evaluation of range expressions using r_{glb} and r_{lub}

Since a propagator needs to perform bounds inferences on each variable $x \in \mathtt{Var}$ Tack *et al.* further define *projectors* [37, sect.4.2] using the range expressions as defined above to perform exactly this task of modifying an approximate domain $E = \Gamma_{\left(\mathcal{C}\right)}(D) \in \left(\mathcal{C}\right)$ of a variable $x : D$ by updating the interval bounds of E as specified by the range expressions G_x and L_x.

Definition 25 (Projectors)**.** *We call a propagator* $p : \left(\mathcal{C}\right)^n \to \left(\mathcal{C}\right)^n$ *such that*

$$\forall \overrightarrow{E} \in \left(\mathcal{C}\right)^n : \forall y \in \texttt{Var} : p\left(\overrightarrow{E}\right).y = \begin{cases} \overrightarrow{E}.y & \text{if } y \neq x \\ p\left(\overrightarrow{E}\right).x & \text{else} \end{cases}$$

a projection propagator *or* projector *for x, written* p_x*. If we are given two range expressions* G_x *and* L_x *as shown in definition 24, such that* p_x *is defined in terms of* G_x *and* L_x*, namely* $p_x \equiv (G_x \subseteq x \subseteq L_x)$*, then the projection result* $p_x\left(\overrightarrow{E}\right).x$ *is defined as*

$$p_x\left(\overrightarrow{E}\right).x \overset{def}{=} F \overset{def}{=} \left[r_{glb}(G_x) \cup \lfloor x \rfloor ..r_{lub}(L_x) \cap \lceil x \rceil\right]_{\subseteq}$$

Since we restricted ourselves to finite domain constraint programming over finite sets we know that our universe of discourse $\mathcal{U}$ is a finite one and consequently it follows, that every constraint c we can model by the ∃MSO-fragment in table 5.2 is a relational constraint, that is c states either a relation only between variables, between variables and the universe or between an element $v \in \mathcal{U}$ and a single set variable if it comes to the basic membership constraint ($v \in x'$). The major gain from this insight is the following fact: in the cardinality set bounds approximation $\left(\mathcal{C}\right)$ the only relations the ∃MSO-grammar uses are subset(⊆), equality(=), disequality(≠) and disjointness(||). In fact, any of these relations can be expressed using the subset relation:

$$\begin{aligned} A = B &\overset{\text{def}}{=} A \subseteq B \wedge B \subseteq A \\ A \neq B &\overset{\text{def}}{=} \neg(A = B) = \neg(A \subseteq B) \vee \neg(B \subseteq A) \\ A \parallel B &\overset{\text{def}}{=} \neg(A \subseteq B) \wedge \neg(B \subseteq A) \end{aligned}$$

Evidently this is the major insight Tack *et al.* use to translate an ∃MSO-formula ϕ_c for a constraint c into a bounds($\left(\mathcal{C}\right)$)-consistent propagator p_c, because this guarantees us that we can transform any ∃MSO-formula into the form

$$\rho \overset{\text{def}}{=} \wedge_x \forall v : (\psi_{1_x} \to v \in x) \wedge (v \in x \to \psi_{2_x})$$

as provided in [37, sect.5.2]. Thus, being able to express a formula ϕ_c as formula ρ_c we can express any conjunct of the form $(\psi_{1_x} \to v \in x) \wedge (v \in x \to \psi_{2_x})$ using range expressions G_x and L_x as $(G_x \subseteq x) \wedge (x \subseteq L_x)$ directly corresponding to the input a single projector needs to propagate on the domain of the set variable x (see definition 25). Repeating this process for all conjuncts of the formula ρ_c we thus achieve propagation on all set variables $x \in \texttt{Var}$ occurring in the ∃MSO-formula ϕ_c. Consequently we obtain a propagator p_c for a constraint c by constructing the projector set of all projectors p_x for all set variables $x \in \texttt{Var}$ that occur in the ∃MSO-formula ϕ_c. This procedure exactly describes the two steps in which Tack *et al.* translate an ∃MSO-formula ϕ_c into the corresponding propagator p_c, namely transforming the formula ϕ_c into a formula ρ_c only using implication (⇒) as Boolean connector, and creating the set of projectors for all conjuncts of ρ_c (see [37, sect.5]). Moreover, they also prove that those created propagators are complete as we will see in 6.7.

5.3 Propagators on $\left(\mathcal{F}\right)$

In contrast to bounds($\left(\mathcal{C}\right)$)-consistent propagators, domain-consistent propagators $p : \left(\mathcal{F}\right)^n \rightarrow \left(\mathcal{F}\right)^n$ follow a different principle in order to update the domains of the variables they reason about. From chapter 3.2.3 we know that an approximate domain $D \in \left(\mathcal{F}\right)$ is represented via a tuple of a Boolean vector v and a Boolean function f represented by a ROBDD encoding all feasible assignments for v. If a propagator p_c modifies such a domain it exploits the fact, that the constraint c it implements can be expressed via a Boolean formula itself. Consequently, updating a domain D can be expressed by forming the Boolean conjunction of all variable domains D_y, $y \in \texttt{Var}$ the constraint reasons about and the Boolean expression $\beta(c)$ of the constraint $\epsilon = \bigwedge_{x \in \texttt{Var}} \wedge \beta(c)$. However, in case that c is a non-basic constraint, that is it reasons about more than one single variable $\beta(c)$, the resulting Boolean expression, also reasons about more than one variable. Thus forming only the conjunction ϵ would lead to an updated domain E of D which not only contains information about its corresponding variable $x \in \texttt{Var}$, $x : D$, but also contains information about all other variables $y \in \texttt{Var}$, $y \neq x$ appearing in the constraint c.

5.3.1 From ∃MSO to $\left(\mathcal{F}\right)$: ROBDD-based projectors

In order to avoid this overflow of information a domain-consistent propagator p follows the above mentioned principle of projection by projecting the result ϵ back to the variable x in focus, such that only the domain information the constraint c imposes on x is propagated to the variable domain. Let F be a Boolean formula with existentially quantified Boolean variables $x_1, \ldots x_n$ in it. Then Hawkins *et al.* define the shorthand $\exists_V F \overset{\text{def}}{=} \exists x_1 \ldots \exists x_n F$ and $\overline{\exists}_V F \overset{\text{def}}{=} \exists_{V'} F$ such that $V' \overset{\text{def}}{=} V \setminus \text{vars}(F)$. Assume that the domain D of the variable x is the x-component of the initial tuple of variable domains $\overrightarrow{D}$. Given this shorthand as defined in [22, sect.2.5] Hawkins *et al.* now define the updated variable domain E of x as propagation result $p\left(\overrightarrow{D}\right).x$ of p on the variable $x : D$ such that

$$p\left(\overrightarrow{D}\right).x \overset{\text{def}}{=} \begin{cases} \overline{\exists}_{b(x)} \left(\beta(c) \wedge \bigwedge\limits_{x_i \in \texttt{Var}} D_i\right) & \text{if} x \in \text{vars}(c) \\ D & \text{else} \end{cases} \tag{5.1}$$

where $b(x)$ denotes all Boolean variables in the Boolean vector b of the respective variable x. The point, where the ∃MSO-fragment presented in table 5.1 comes in is exactly the construction of the Boolean expression $\beta(c)$ representing a constraint c. From section 5.2 we know that except basic cardinality constraints we can model any set constraint c using ∃MSO as a formula ϕ_c. Moreover, since the universe of discourse $\mathcal{U}$ is finite, we can replace the universal quantifier $\forall v$ in an ∃MSO-formula by the finite conjunction over all values in the universe $\bigwedge_{v \in \mathcal{U}}$ as Tack *et al.* point out in [37, sect.7]. Consider for example the constraint $c \equiv x \cup y = z$. Encoding c in ∃MSO we obtain an ∃MSO-formula $\phi_c = \forall v.(v \in x \vee v \in y) \Leftrightarrow v \in z$. Replacing the universal quantifier by a conjunction over all values $v \in \mathcal{U}$ we obtain the Boolean formula $\beta(c) = \bigwedge_{v \in \mathcal{U}} x_v \vee y_v \Leftrightarrow z_v$. Exchanging the conjunction over all values v in the universe $\mathcal{U}$ by a conjunction over all components of the Boolean vector $b = \langle v_0, \ldots, v_{N-1} \rangle \wedge N = |\mathcal{U}| - 1$ representing the universe we obtain precisely the Boolean formulas which are used to encode the set variables $x \in \texttt{Var}$ and the constraints c over them as represented in the domain approximation $\left(\mathcal{F} \right)$, namely $\beta(c) = \bigwedge_{i=0}^{N-1} x_i \vee y_i \Leftrightarrow z_i$.

5.3.2 Conclusion

During the course of this chapter we have seen that a non-basic constraint, that is a constraint modifying more than one variable, is decomposed into basic constraints that directly correspond to domain update operations as provided by the underlying data-structures for approximate domains highlighted in section 3.2. Moreover we have seen that Tack *et al.* present in [37] a concept to use these decompositions in order to abstract constraints using logic formulas expressed in a fragment of ∃MSOas depicted in table 5.1. This concept neatly fits into our picture of abstracting the domains of set variables via approximate domains and to access them via views since it enables us to abstract bounds($\left(\mathcal{C} \right)$)- and domain-consistent propagators on these views into a single constraint modeling language using ∃MSO-formulas. Hence we can complete the integration of a ROBDD-based constraint solver into *Gecode* , since we now are not only able to generate the desired variable view on the respective approximate domains crossing consistency levels but also to generate both, the bounds($\left(\mathcal{C} \right)$)- as well as the domain-consistent propagator from a single ∃MSO-formula. Finally, we obtained a general set constraint solver, which is not only able to abstract in its variable component over the respective variable domains but also in the propagator component by allowing the choice between different propagation algorithms implementing different levels of consistency for the modeling of a CSP.

6 Evaluation

See first, think later, then test. But always see first. Otherwise you will only see what you were expecting. Most scientists forget that.

— Adams

The preceding chapters 2 to 5 presented a framework that eases comparison of different representations for set variables including representation of variable domains as well as representation of the propagators operating on those variables. In this chapter we aim at providing a thorough analysis and evaluation of the implementation of this framework using the *Gecode* -library. As it comes to benchmark criteria for the evaluation we mainly focus on the different representations by time and space consumption in terms of runtime and memory usage.

6.1 Aspects of evaluation

The empirical evaluation presented in this chapter will concentrate on the following aspects:

Two implementations for the same domain approximation - IB versus DB First of all we focus on the comparison of two different implementations for the cardinality set bounds domain approximation $(\mathcal{C})$ as discussed in section 3.2.2, namely the standard IB representation where the lower set bound is included in the upper set bound which is the standard *Gecode* -implementation and the DB representation as presented in 3.2.2 which builds on disjoint set bounds.

Different domain approximations - $(\mathcal{C})$ versus $(\mathcal{F})$ Apart from testing different implementations for the same domain approximation $(\mathcal{C})$ we also glance at the question, whether the newly ROBDD implementation of the $(\mathcal{F})$ domain approximation can compete with standard $(\mathcal{C})$ domain approximation that already comes with *Gecode* . In this context we pay peculiar attention to runtime behavior and size of the resulting search tree for the respective benchmark.

Capabilities of variable views - same consistency with different views For this benchmark, we test whether it pays off to use views in order to mimic the $(\mathcal{C})$ domain approximation using the $(\mathcal{F})$ domain approximation plus the $(\mathcal{C})$-view $\Gamma_{(\mathcal{C})}$ as propagation interface. Another aspect of variable views this evaluation will look at is the result of using variable views to weaken the consistency of a domain-consistent ROBDD-propagator to achieve set bounds($\left(\mathcal{S}\right)$), cardinality set bounds($\left(\mathcal{C}\right)$) or even lexicographic bounds($\left(\mathcal{L}\right)$) consistency.

6.2 Set of Benchmarks

For the sake of comparability we choose a standard set of benchmarks for this evaluation which is commonly agreed on in literature: the *Steiner* problem as used in [22, 30, 5, 26, 2], the *Hamming* problem as studied in [30, 2] and the *Social Golfer* problem as stated in [5, 2]. All of the problems used for the evaluation in this section are provided as examples in the *Gecode* -library [39]. Subsequently, this sections provides specifications for the above mentioned benchmarking problems as well as CSP-encodings used in the evaluation presented in the next section.

6.2.1 Social Golfer Problem

The *social golfer problem* is problem number `prob010` in *csplib*: a problem library for constraints [14] which provides the following problem specification:

Specification The coordinator of a local golf club has come to you with the following problem. In her club, there are 32 social golfers, each of whom play golf once a week, and always in groups of 4. She would like you to come up with a schedule of play for these golfers, to last as many weeks as possible, such that no golfer plays in the same group as any other golfer on more than one occasion. Possible variants of the above problem include: finding a 10-week schedule with "maximum socialisation"; that is, as few repeated pairs as possible (this has the same solutions as the original problem if it is possible to have no repeated pairs), and finding a schedule of minimum length such that each golfer plays with every other golfer at least once ("full socialisation"). The problem can easily be generalized to that of scheduling m groups of n golfers over p weeks, such that no golfer plays in the same group as any other golfer twice (i.e. maximum socialisation is achieved).

A CSP Model An instance `golf`-$(\mathtt{w}, \mathtt{g}, \mathtt{s})$ of the social golfer problem specifies that we have to setup a schedule for $p \overset{\text{def}}{=} g \cdot s$ players over w weeks such that s players are in each of the g groups scheduled for one week. For the purpose of evaluation we will compare to different models for this instances that we differentiate by `golf`-$(\mathtt{w}, \mathtt{g}, \mathtt{s})$-`n` denoting the naive model with less constraints and by `golf`-$(\mathtt{w}, \mathtt{g}, \mathtt{s})$-`s` for the smart model using additional symmetry breaking and redundant constraints as stated below. Obviously, the set `Var` of problem variables contains $m \overset{\text{def}}{=} g \cdot w$ set variables $x_{ij} : D_{ij} = [\emptyset..P]$, where $P \overset{\text{def}}{=} \{0, \ldots, p-1\}$ where each set variable x_{ij} denotes the group $j \in G \overset{\text{def}}{=} \{0, \ldots, g-1\}$ in week $i \in W \overset{\text{def}}{=} \{0, \ldots, w-1\}$. Concerning the sect of constraints $\mathcal{C}$ we can obviously model the problem using the following constraints $c \in \mathcal{C}$: The most straightforward constraint for this model consists in restricting a set variable denoting a group to contain exactly s players as required by the problem description:

$$\forall i \in W \forall j \in G : cardgroup_{ij} \equiv \left|x_{ij}\right| = s \tag{6.1}$$

Since every player plays per week, the groups of one week form a disjoint partition of the set P, that is

$$\forall i \in W : part_i \equiv \biguplus_{j \in G} x_{ij} = P \tag{6.2}$$

As the schedule is supposed to achieve a maximum degree of socialisation, no two golfers shall play in the same goup more than once, what is expressed by the equation:

$$\forall i, k \in W \forall j, l \in G, i \neq k, j \neq l : binatmost_{(ij,kl)} \equiv \exists z : x_{ij} \cap x_{kl} = z \wedge |z| \leq 1 \tag{6.3}$$

Let

$$\begin{aligned} Card &= \bigcup_{i \in W} \bigcup_{j \in G} cardgroup_{ij} && (6.4)\\ Part &= \bigcup_{i \in W} part_i && (6.5)\\ BinAtMostOne &= \bigcup_{i,k \in W, i \neq k} \bigcup_{j,l \in G, j \neq l} binatmost_{(ij,kl)} && (6.6) \end{aligned}$$

then we can already formulate the naive model for the social golfer problem as

$$\mathcal{GPN} = \langle \mathtt{Var}_{\mathcal{GPN}}, C_{\mathcal{GPN}} \overset{\text{def}}{=} Card \cup Part \cup BinAtMostOne \rangle$$

Smart Model For the smart model we additionally introduce the following constraints: For each week $i \in W$ we introduce a set of boolean variables B_{ik}, such that $b_j \in B_{ik}$ denotes the fact, that player k belongs in week i to the group j. Using this notation we impose the constraint that in each week $i \in W$, one player $k \in P$ plays in only one group $j \in G$:

$$\forall i \in W \forall k \in P : OnlyOne_{ik} \equiv \left[\left(\forall j \in G : \{k\} \subset x_{ij} \Leftrightarrow b_j \right) \wedge \sum_{j=0}^{g-1} b_j = 1 \right] \tag{6.7}$$

A redundant global *atmostOne* constraint as discussed in 5.3.2 and an additional symmetry breaking constraint to order the groups in each week by introducing integer variables $y_m : P$ such that

$$\begin{aligned} \forall i \in W \forall j \in \{0, \ldots, g-2\} : SymGroupWeek_{ij} &\equiv y_j = \min(x_{ij}) \wedge y_{j+1} = \min(x_{i(j+1)}) && (6.8)\\ &\wedge \; y_j < y_{j+1} \end{aligned}$$

Finally we introduce intermediate set variables $z_i : \mathcal{P}(P)$ and intermediate integer variables $y_i : P$ to obtain a quasi-lexicographic ordering on the first groups in each week by stating:

$$\begin{aligned} \forall i \in \{0, \ldots, w-2\} : SymFstGroup_i &\equiv z_i = x_{i0} \setminus \{0\} \wedge z_{i+1} = x_{(i+1)0} \setminus \{0\} \\ &\wedge\ y_i = \min(z_i) \wedge y_{i+1} = \min(z_{i+1}) \wedge y_i < y_{i+1} \end{aligned} \tag{6.9}$$

Let

$$OnlyOne \overset{\text{def}}{=} \bigcup_{i \in W} \bigcup_{k \in P} OnlyOne_{ik} \tag{6.10}$$

$$SymGroupWeek \overset{\text{def}}{=} \bigcup_{i \in W} \bigcup_{j \in \{0,\ldots,g-2\}} SymGroupWeek_{ij} \tag{6.11}$$

$$SymFstGroup \overset{\text{def}}{=} \bigcup_{i \in W} SymFstGroup_i \tag{6.12}$$

then we obtain the smart model for the social golfers problem as

$$\begin{aligned} \mathcal{GPS} = \langle \mathtt{Var}_{\mathcal{GPN}}, C_{\mathcal{GPS}} &\overset{\text{def}}{=} C_{\mathcal{GPN}} \cup OnlyOne \cup \{atmostOne\} \\ &\cup\ SymGroupWeek \cup SymFstGroup \rangle \end{aligned} \tag{6.13}$$

Branching As for the branching strategy (see section 2.2.2) we first choose the set variable $x_{ij} : D_{ij}$ such that $v \overset{\text{def}}{=} \min(D_{ij})$ is the smallest minimum of all variables and add proceed to the left branch of the search tree by additionally imposing the constraint $v \in x_{ij}$ on x_{ij}.

Example 7 (Golf instance)**.** *A feasible schedule for the instance* `golf-(2, 4, 3)` *of the social golfers problem is provided in table 6.1.*

		groups			
		0	1	2	3
weeks	0	{0,1,2}	{3,4,5}	{6,7,8}	{9,10,11}
	1	{0,3,6}	{1,4,9}	{2,7,10}	{5,8,11}

Table 6.1: Schedule for problem instance `golf-(2, 4, 3)`

6.2.2 General Steiner System

Since the CSP-library only comprises the *ternary steiner problem*, an instance of the *general steiener problem* we will use the following specification for the *general steiner problem* as presented by Hawkins *et al.* in [22].

Specification The *general steiner problem* consists in the problem of computing so-called *Steiner systems* `steiner`-(t, k, n), where Steiner system is a set S with cardinality $|S| = n$ and a collection T of $m = \binom{n}{t} / \binom{k}{t}$ subsets called "blocks" of S such that all subsets $b \in T$ have cardinality $|b| = k$ and such that t elements of S are contained in exactly one subset $b \in T$. Instantiating this general Steiner system to `steiner`-(2, 3, n) we obtain the *ternary Steiner* instance of this problem which is specified as problem number `prob044` in [14]. As Azevedo points out any instance of a ternary Steiner problem `steiner`-(2, 3, n) can be expressed as an instance of the *balanced incomplete block design problem* (`prob028` in [14]) such that `steiner`-(2, 3, n) = `bibd`-(n, m = n · (n − 1)/6, (n − 1)/2, 3, 1) .

A CSP Model For an instance `steiner`-(t, k, n) of the general steiner problem we obtain the set S as $S \overset{\text{def}}{=} \{1, \dots, n\}$ and define the set I of indices to be $I \overset{\text{def}}{=} \{0, \dots, n-2\}$. Furthermore, we can choose the set `Var` of problem variables such that $\forall x_i \in \texttt{Var} : x_i : [\emptyset..\mathcal{P}(S)]$. The evaluation in the following chapters will compare different models for this instances that differ in the kind of symmetry breaking constraints. With `steiner`-(t, k, n)-n we denote the naive model with less elaborate symmetry breaking and with `steiner`-(t, k, n)-s we denote the smart model using a more elaborate symmetry breaking than the naive model. The set $\mathcal{C}$ of constraints for an instance of the general steiner system looks as follows: The most straightforward constraint for this model consists in restricting a set variable x_i representing a block b in the collection T to contain exactly k elements of S using the equation

$$\forall i \in \{0, \dots, n-1\} : Card_i \equiv |x_i| = k \tag{6.14}$$

Restricting the set variables x_i such that they intersect in at most one element $s \in S$ is expressed by the following constraint

$$\forall i, j \in I, i \neq j : BinAtMostOne_{ij} \equiv \exists z : x_i \cap x_j = z \wedge |z| \leq 1 \tag{6.15}$$

In order to eliminate symmetries the naive model introduces $\binom{n}{2}$ sets Y_{ij} and Z_{ij} of k integer variables $y_{ijk} \in Y_{ij}$ and $z_{ijk} \in Z_{ij}$ each such that $y_{ijk} : [1..n]$ and $z_{ijk} : [1..n]$. In the naive model we eliminate possibility of permuting the block alignment using the following constraint:

$$\begin{aligned} \forall i, j \in I, i \neq j : SymBreak_{ij} \quad \equiv \quad & \forall s \in \{0, \dots, k-1\} : \{y_{ijs}\} \subseteq x_i \wedge \{z_{ijs}\} \subseteq x_j \\ \wedge \quad & \left[\forall r, s \in \{0, \dots, k-2\}, r \neq s : z_{ijr} < z_{ijs} \wedge y_{ijr} < y_{ijs}\right] \\ \wedge \quad & \sum_{s=0}^{k-1} (n+1)^i \cdot z_{ijs} - \left(\sum_{s=0}^{k-1} (n+1)^i \cdot y_{ijs}\right) < 0 \end{aligned} \tag{6.16}$$

Defining the sets

$$\begin{aligned} Card &\equiv \bigcup_{i\in I} Card_i && (6.17)\\ BinAtMostOne &\equiv \bigcup_{i,j\in I, i\neq j} BinAtMostOne_{ij} && (6.18)\\ SymBreak &\equiv \bigcup_{i,j\in I, i\neq j} SymBreak_{ij} && (6.19)\end{aligned}$$

we are able to specify the naive CSP model for the general steiner system as

$$\mathcal{GSPN} \overset{\text{def}}{=} \langle \mathtt{Var}_{\mathcal{GSPN}}, C_{\mathcal{GSPN}} \overset{\text{def}}{=} Card \cup BinAtMostOne \cup SymBreak \rangle$$

For the smart version of the general steiner CSP model we modify the symmetry breaking constraint such that

$$\begin{aligned} \forall i,j \in I, i\neq j : SymBreakM_{ij} \equiv\ & match_{ij} && (6.20)\\ \wedge\ & \left[\forall r,s\in\{0,\ldots,k-2\}, r\neq s : z_{ijr} < z_{ijs} \wedge y_{ijr} < y_{ijs}\right] \\ \wedge\ & \sum_{s=0}^{k-1} n^i \cdot z_{ijs} - \left(\sum_{s=0}^{k-1} n^i \cdot y_{ijs}\right) = -1 \end{aligned}$$

where $match_{ij}$ is a global constraint on two set variables x_i, x_j and the introduced integer sets Y_{ij} and Z_{ij} such that

$$\begin{aligned} match_{ij} \equiv\ & \big[Y_{ij} = x_i \wedge Z_{ij} = x_j && (6.21)\\ \wedge\ & \forall r,s\in\{0,\ldots,k-2\}, r\neq s : z_{ijr} < z_{ijs} \wedge y_{ijr} < y_{ijs}\big] \end{aligned}$$

Redefining the set of symmetry breaking constraints $SymBreak$ to the set of all symmetry breaking constraints using the $match_i$ constraint we obtain

$$SymBreak \equiv \bigcup_{i,j\in I, i\neq j} SymBreakM_{ij} \qquad (6.22)$$

we thus obtain the following CSP encoding for the smart model of the general steiner system

$$\mathcal{GSPS} \overset{\text{def}}{=} \langle \mathtt{Var}_{\mathcal{GSPS}}, C_{\mathcal{GSPS}} \overset{\text{def}}{=} Card \cup BinAtMostOne \cup SymBreak \rangle$$

Example 8 (Steiner)**.** *A feasible collection of* $\binom{9}{2}/\binom{3}{2} = 12$ *blocks for the general steiner instance* `steiner-(2, 3, 9)` *is given in table 6.2.*

6.2.3 Hamming Distance

As there is no further specification of the Hamming problem in the CSP-library [14] we stick to the definition of the *Hamming* problem as given in [30] that is provided below

$$12 \text{ blocks} \begin{cases} \{1,2,3\} \\ \{1,4,5\} \\ \{1,6,7\} \\ \{1,8,9\} \\ \{2,4,6\} \\ \{2,5,8\} \\ \{2,7,9\} \\ \{3,4,9\} \\ \{3,5,7\} \\ \{3,6,8\} \\ \{4,7,8\} \\ \{5,6,9\} \end{cases}$$

Table 6.2: Steiner instance `steiner-(2, 3, 9)`

Specification (Binary error correcting codes). A binary error correcting code is a collection of n bit-strings, that is vectors of zeros and ones of length b, called codewords, with the property that the distance between any two codewords is at least some number d. The distance between two codewords is defined to be the number of positions in which the two bit-strings vary. This distance function is called the *Hamming distance*. We will abbreviate an instance of the above specification with `hamming-(n, b, d)`. The evaluation in the next chapter focuses on a `hamming-(32, 20, 3)` instance of the hamming distance problem.

A CSP Model Let $B \overset{\text{def}}{=} \{1, \ldots, b\}$. Then we can specify the set of problem variables `Var` as n set variables $x_i : [\emptyset..\mathcal{P}(B)]$ such that the bit-vector for the codeword c_i represents the characteristic function of x_i, that is $v \in x_i$ if the v-th bit is set in c_i. According to Sadler and Gervet we compute the distance between two codewords c_i and c_j as the cardinality of the symmetric difference of the set variables x_i and x_j representing the codewords[30, sect.6].

Definition 26 (Symmetric difference). *Let $A, B \subset \mathcal{U}$ be two finite sets. Then we define the* symmetric difference *$A \oplus B$ as $A \oplus B \overset{def}{=} (A \cup B) \setminus (A \cap B) = (A \cap \overline{B}) \cup (B \cap \overline{A})$.*

Using the above definition we can encode the distance constraint between two codewords as

$$\forall i, j \in \{0, \ldots, 31\}, i \neq j : HamDist_{ij} \equiv \left|x_i \cap \overline{x_j}\right| + \left|x_j \cap \overline{x_i}\right| \geq d \tag{6.23}$$

Additionally we restrict the cardinality of the x_i to be at least one and at most b, that is :

$$\forall i \in \{0, \ldots, 31\} : Card_i \equiv 1 \leq |x_i| \leq b \tag{6.24}$$

If we define the set of *HamDist* of binary $HamDist_{ij}$ constraint as

$$HamDist \equiv \bigcup_{i,j \in \{0,\ldots,31\}, i \neq j} HamDist_{ij} \tag{6.25}$$

and the set *Card* of all cardinality constraints as

$$Card \equiv \bigcup_{i \in \{0,\ldots,31\}} Card_i \tag{6.26}$$

we obtain a CSP encoding $\mathcal{HD}$ for the hamming distance problem as follows

$$\mathcal{HD} \overset{\text{def}}{=} \langle \mathtt{Var}_{\mathcal{HD}}, \mathcal{C}_{\mathcal{HD}} \overset{\text{def}}{=} Card \cup HamDist \rangle$$

Example 9 (Hamming Instance)**.** *A solution to the problem instance* `hamming`-(6, 20, 3) *for the Hamming Distance problem is provided in the table 6.3.*

[1..20]	⟨11111111111111111111⟩
[1..17]	⟨11111111111111111000⟩
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 18, 19}	⟨11111111111111100110⟩
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 20}	⟨11111111111111100001⟩
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 18, 20}	⟨11111111111111010101⟩
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 19}	⟨11111111111111010010⟩
(a) set representation	(b) corresponding bit vectors

Table 6.3: Feasible solution for the instance `hamming`-(6, 20, 3) of the Hamming Distance problem

6.3 Benchmark platform

All experiments use *Gecode* , a C++-based generic constraint programming library [39]The version used in this paper corresponds to *Gecode* 1.3.0 extended with the following implementations: an alternative disjoint bounds implementation for the $\left(\mathcal{C}\right)$ domain approximation already contained in the *Gecode* library, a ROBDD-implementation interfacing the BuDDyBDD library [25] and the respective implementations of variable views as mentioned in chapter 4. *Gecode* has been compiled with gnu gcc 4.1.0. All examples have been run on a Laptop with a 2 GHz Pentium M CPU and 1024 MB main memory running Gentoo Linux. If not stated otherwise the runtimes as indicated in the evaluation tables are the average of 20 runs with a coefficient of deviation less than 2% for all benchmarks.

6.4 Different implementations for cardinality set bounds $\left(\mathcal{C}\right)$

A major task and contribution of this thesis is the comparison between different implementations for the cardinality set bounds domain approximation $\left(\mathcal{C}\right)$ using the *Gecode* framework.

One of those implementations is the including bounds representation IB already shipped with the *Gecode* -library. Recapitulating section 3.2.1 the IB representation essentially bases on the approximation of a set variable domain D via the bounds $\lfloor D \rfloor$ and $\lceil D \rceil$, such that $\lfloor D \rfloor \subseteq \lceil D \rceil$, and contains additional cardinality information to achieve $\left(\mathcal{C}\right)$ domain approximation. In this section we compare this representation with the disjoint bounds representation DB we looked at in section 3.2.2. Contrarily to the IB representation, the DB implementation bases its approximation of D on the two sets $\lfloor D \rfloor$ and $\Delta \overset{\text{def}}{=} \lceil D \rceil \setminus \lfloor D \rfloor$.

problem instance	IB - $[\lfloor D \rfloor .. \lceil D \rceil]$ absolute time (ms)	mem (KB)	DB - $[\lfloor D \rfloor .. \Delta]$ (diff in %) time	mem
`steiner-(2,3,7)-n`	0.7425	91	+0.1838	+0.2088
`steiner-(2,3,7)-s`	0.4435	58	+0.1353	+0.2759
`steiner-(2,3,9)-n`	81.1500	591	+0.1935	+0.2284
`steiner-(2,3,9)-s`	45.5000	349	+0.1626	+0.2865
average	—	18447	**+0.1688**	**+0.2499**
`hamming-(32,20,3)`	963.1000	8879	+0.3947	+0.1064
`golf-(9,8,4)-n`	184.0000	10794	+0.2826	-0.0072
`golf-(9,8,4)-s`	446.2000	1750	+0.2418	±0.0000
`golf-(7,5,3)-s`	1138.2000	25	+0.2703	+0.1697
`golf-(2,4,3)-n`	25.7350	159	+0.2427	+0.0400
`golf-(2,4,3)-s`	39.4600	86	+0.2619	+0.1047
average	—		**+0.2900**	**+0.0856**
average (all)			**+0.2459**	**+0.1586**

Table 6.4: Comparison: including bounds (IB) vs. disjoint bounds(DB)

6.4.1 Intuition and results

Intuitively, exchanging the IB representation by the DB representation should result in an increase of performance with respect to time and memory consumption since the DB representation does not represent the lower set bound $\lfloor D \rfloor$ twice as it is the case with the IB representation. A comparison between the two implementations based on instances of the *Hamming Distance* problem(6.2.3), the *General Steiner System* problem (6.2.2) as well as the *Social Golfer* problem is summarized in table 6.4. Considering the column "DB -$[\lfloor D \rfloor .. \Delta]$" referring to the disjoint bounds representation DB an entry of the form +0.1838 highlights that the time or memory consumption of the DB implementation is 0.1838 percent larger relative to the performance of the IB representation. Strikingly, table 6.4 underlines that the average runtime performance of the DB representation is approximately $0.2459 \approx 0.3\%$ worse than the respective runtime of the IB representation. In addition to the inferior runtime of the DB representation table 6.4 shows also that the memory consumption of the DB representation is $0.1586 \approx 0.2\%$ worse than the IB representation.

Evaluation of variable modification Obviously now the question arises why these results do not affirm the intuition that the DB representation consumes less time and less memory than the IB representation. As the *Gecode* architecture bases the computation of the propagation order on variable modification and the respective modification events it allows us to keep track of the modification events occurring during execution of a benchmark. This way we can observer more precisely what domain lookup and update operations on the underlying range-list data structure are performed. Since section 2.2.1 shows that the modification events depend on the used domain approximation we base the evaluation on the modification events depending on $\left(\mathcal{C}\right)$. Recalling that a set variable $x : D$ approximated with $\left(\mathcal{C}\right)$ is represented as $SV(x) = \langle \lfloor D \rfloor, \lceil D \rceil, \mathsf{C} = [l..r] \rangle$ this results in the following modification events for the modified domain $SV(x) = \langle \lfloor E \rfloor, \lceil E \rceil, \mathsf{C} = [v..w] \rangle$:

$val(x)$	$\lfloor E \rfloor = \lceil E \rceil$
$glb(x)$	$\lfloor E \rfloor \subset \lfloor D \rfloor$
$lub(x)$	$\lceil E \rceil \subset \lceil D \rceil$
$bb(x)$	$glb(x) \wedge lub(x)$
$dom(x)$	$[\lfloor E \rfloor .. \lceil E \rceil] \subset [\lfloor D \rfloor .. \lceil D \rceil]$

(a) modified domain

$none(x)$	no modification
$cglb(x)$	$card(x) \wedge glb(x)$
$club(x)$	$card(x) \wedge lub(x)$
$cbb(x)$	$card(x) \wedge bb(x)$
$card(x)$	$l < v \vee w < r \wedge \neg dom(x)$

(b) modified cardinality

Table 6.5: Modification events for $\left(\mathcal{C}\right)$

Clearly, the DB representation cannot perform better in case we do not modify the range-list representation of the set bounds interval directly. Hence we can safely exclude the cases from evaluation where the original domain D is already a singleton set($|D| = 1$), where we only modify the cardinality of the representation ($card(x)$) or where we perform a domain update operation ($D \diamond S$) with an empty ground set $S = \emptyset$. After discarding those cases where no performance improvement is possible table 6.6 indicates that for all benchmarks except the instance `steiner-(2, 3, 9)-n` the remaining number of cases where a domain modification ($dom(x)$) could occur still sums up to 20% on average of all detected modification events. However, the lines $none(x)$ referring to the cases in which no domain modification occurs make up from 60% in `steiner-(2, 3, 9)-n` up to 90% of the interesting cases in `golf-(7, 5, 3)-s`. As we have seen in chapter 3.2.2 detecting that no modification occurred ($none(x)$) takes with DB at least as much time as with IB , but is even worse in most cases because of the higher constants the DB representation has to pay for. The major pain for the use of DB consist in the fact that the subset test on the IB representation is only performed in case of modification ($dom(x)$). But since exactly those cases where the DB representation could outperform the IB representation at all do not make up more than 3% in any of the benchmarks there is no way for the DB representation to

	`steiner`-(2, 3, 9)			
	`smart`		`naive`	
		in %		in %
detected	476871	100.00	725850	100.00
remaining	116855	**24.50**	69835	**9.62**
fail(x)	197	0.04	629	0.09
none(x)	104383	21.89	44305	6.10
val(x)	3283	0.69	8352	1.15
lub(x)	8292	1.74	14788	2.04
glb(x)	700	0.15	1761	0.24
dom(x)	12275	**2.57**	24901	**3.43**

(a) Events for `steiner`-(2, 3, 9)

	`hamming`-(32, 20, 3)	
		in %
detected	8104029	100.00
remaining	1651962	**20.38**
fail(x)	68	0.00
none(x)	1437204	17.73
val(x)	46511	0.57
lub(x)	60663	0.75
glb(x)	16368	0.20
club(x)	54326	0.67
cglb(x)	36822	0.45
dom(x)	214690	**2.65**

(b) Events for `hamming`-(32, 20, 3)

	`golf`-(9, 8, 4)-n		`golf`-(7, 5, 3)-s	
		in %		in %
detected	2753601	100.00	9068638	100.00
remaining	564729	**20.51**	2500241	**27.57**
fail(x)	32	0.00	545	0.01
none(x)	497017	18.05	2255813	24.87
val(x)	1877	0.07	26166	0.29
lub(x)	65447	2.38	209445	2.31
glb(x)	356	0.01	5298	0.06
club(x)	—-	—-	1158	0.01
cglb(x)	—-	—-	1816	0.02
dom(x)	67680	**2.46**	243883	**2.69**

(c) Events for `golf` instances

Table 6.6: Profiling modifcation events for $(\mathcal{C})$

yield a better performance neither with respect to time nor with respect to memory.

6.5 Cardinality set bounds $(\mathcal{C})$ versus full domain approximation $(\mathcal{F})$

The aim of this chapter is to provide an empirical comparison of the cardinality set bounds representation $(\mathcal{C})$ against the full domain approximation based on the ROBDD-approach as presented in section 3.2.3. In order to compare the range-list based $(\mathcal{C})$ implementation against the ROBDD based $(\mathcal{F})$ implementation we first of all focus on problem instances for the *General Steiner System* (see 6.2.2) Since this section compares orthogonal domain representations, namely range-list based $(\mathcal{C})$ representation in *Gecode* and the ROBDD based $(\mathcal{F})$ approximation of [22], we have a look at the following table 6.7.

problem	$(\mathcal{C})$ *Gecode*	$(\mathcal{F})$ ROBDD
	time(ms)	
`steiner`-(2, 3, 7)-`s`	0.4535	9.205
`steiner`-(3, 4, 8)-`s`	5.0790	91.016
`steiner`-(2, 3, 9)-`s`	45.8850	92.850
`steiner`-(2, 4, 13)-`s`	3.3150	271.866
`steiner`-(2, 3, 15)-`s`	26.5100	4550.000
`steiner`-(2, 4, 16)-`s`	—	2200.000
`steiner`-(2, 5, 21)-`s`	16.2400	13410.000

Table 6.7: Comparison of the orthogonal domain approximations $(\mathcal{C})$ and $(\mathcal{F})$

Obviously, the $(\mathcal{C})$ approach outperforms the ROBDD based $(\mathcal{F})$ domain approximation in almost every problem instance of the *General Steiner* problem by at least a factor 2 in the instance `steiner`-(2, 3, 9)-`s` and at most by an immense factor of approximately 825 in case of instance `steiner`-(2, 5, 21)-`s`. Although this table suggests that the $(\mathcal{C})$ representation as available in *Gecode* is still the representation of choice with respect to runtime one gains the possibility to solve instances with the $(\mathcal{F})$ representation that cannot be solved by the $(\mathcal{C})$ representation within one hour as it is the case for the problem instance `steiner`-(2, 4, 16)-`s`.

6.6 Using Views

As proposed in chapter 4 this section provides a summary of the discussed views for cross-domain propagation and weakening of propagator consistency.

Weaking propagator consistency At first we take a glance at the views described in section 4.2.1, which allow to turn a domain-consistent propagator for the $\left(\mathcal{F}\right)$ representation into a bounds($\mathcal{A}$) consistent propagator such that $\mathcal{A} \in \{\left(\mathcal{S}\right), \left(\mathcal{C}\right), \left(\mathcal{L}\right)\}$. An overview of the different views using the *General Steiner* problem is provided in table 6.8.

problem instance	$\left(\mathcal{F}\right)$ ROBDD	$\Gamma_{(\mathcal{S})}\left(\left(\mathcal{F}\right)\right)$ ROBDD	$\Gamma_{(\mathcal{C})}\left(\left(\mathcal{F}\right)\right)$ ROBDD	$\Gamma_{(\mathcal{L})}\left(\left(\mathcal{F}\right)\right)$ ROBDD
	time ms			
steiner-(2, 3, 7)-s	9.205	11.030	28.505	23.456
steiner-(3, 4, 8)-s	91.016	72.930	251.950	138.110
steiner-(2, 3, 9)-s	92.850	590.200	2975.300	—–
steiner-(2, 4, 13)-s	271.866	301.450	4967.000	—–
steiner-(2, 3, 15)-s	4550.000	1653.000	26985.000	—–
steiner-(2, 4, 16)-s	2200.000	—-	—-	—–
steiner-(2, 5, 21)-s	13410.000	3541.500	—-	—–

Table 6.8: Comparison of different views for changing propagator consistency

From the small problem instances, steiner-(2, 3, 7)-s and steiner-(3, 4, 8)-s, we learn that full-domain consistency is obviously the best choice since it for the original identity view on $\left(\mathcal{F}\right)$ there is no such overhead as computing the set bounds in the $\left(\mathcal{S}\right)$ case. Neither does it need to extract the additional cardinality $\left(\mathcal{C}\right)$ or lexicographic bounds $\left(\mathcal{L}\right)$. Only the set bounds view $\left(\mathcal{S}\right)$ is a factor 0.2 faster on the second steiner instance. The set bounds consistent view $\left(\mathcal{S}\right)$ also outperforms the domain-consistent view on larger problem instances as steiner-(2, 3, 15)-by a factor of almost 0.7. Even more so the steiner-(2, 5, 21)-case where it outperforms the $\left(\mathcal{F}\right)$ approximation by approximately a factor of 0.8. Strikingly, the cardinality view $\left(\mathcal{C}\right)$ is even slower than the bounds consistency $\left(\mathcal{S}\right)$ what is due to the computation of the cardinality bounds for every domain lookup and every domain modification concerning the cardinality of the set variable. The same holds for the lexicographic bounds $\left(\mathcal{L}\right)$ which is even slower than the cardinality view.

Crossing approximation boundaries In this paragraph we compare the *Gecode* cardinality set bounds domain approximation with a domain $D \in \left(\mathcal{F}\right)$ that is mapped to $\left(\mathcal{C}\right)$ via a $\Gamma_{(\mathcal{C})}$ view as presented in chapter 4.2.1.

As table 6.9 shows mimicking the $\left(\mathcal{C}\right)$ domain approximation with an underlying domain implementation based on ROBDDs is costly. Like in the above case of using views to weaken domain-consistent propagation the computation of the cardinality bounds from the ROBDD information has to be performed for every mapping back and forth between $\left(\mathcal{C}\right)$ and $\left(\mathcal{F}\right)$ and even more so for every domain lookup operation and domain modification involving cardinality.

problem	*Gecode*	$\Gamma_{(\mathcal{C})}\left(\left(\mathcal{F}\right)\right)$ ROBDD
	time ms	
`steiner`-(2, 3, 7)-s	0.4535	94.5
`steiner`-(3, 4, 8)-s	5.0790	1168.5
`steiner`-(2, 3, 9)-s	45.8850	18580.0
`steiner`-(2, 4, 1)-3 s	3.3150	5610.3

Table 6.9: Comparison of the *Gecode* $\left(\mathcal{C}\right)$ approximation and a mimicked $\left(\mathcal{C}\right)$

6.7 Comparison *Gecode* projectors vs. ROBDD projectors

In this section we evaluate the propagator generation for finite set constraints in the *Gecode* and compare them to the generated projectors for the ROBDD $\left(\mathcal{F}\right)$ domain approximation. Similarly to the preceding sections this comparison also bases on the *General Steiner* problem. However, we modify the model from section 6.2.2 and omit the symmetry breaking constraints, which impose an order on the blocks of the collection of subsets in the solution. Thus we obtain the following results as shown in table 6.10.
To highlight the important result we also added runtime and number of fails for the $\left(\mathcal{C}\right)$ domain approximation in *Gecode* and the $\left(\mathcal{C}\right)$ domain approximation using ROBDDs and weakened domain propagation. The most obvious result is the difference in number of detected fails for the problem instance `steiner`-(2, 3, 9). Though its poor performance as witnessed by the preceding sections, the use of $\Gamma_{(\mathcal{C})}$ $\left(\mathcal{F}\right)$ to perform bounds($\left(\mathcal{C}\right)$) consistent propagation on a domain $D \in \left(\mathcal{F}\right)$ has the tremendous advantage of completeness. Although in the ternary *Gecode* intersection constraint $x \cap y = z$ also includes cardinality propagation, this is not domain-consistent for the cardinality information. Hence, the *Gecode* $\left(\mathcal{C}\right)$ representation has less fails in the `steiner`-(2, 3, 9) instance than the corresponding bounds($\left(\mathcal{C}\right)$) consistent projector for *Gecode* but is incomplete with respect to propagation of cardinality information since it produces more fails than $\Gamma_{(\mathcal{C})}\left(\left(\mathcal{F}\right)\right)$. Moreover this example shows, that set bounds consistent generated projectors in *Gecode* are indeed complete as proven by Tack *et al.* in [37] since it has the same number of fails as the corresponding bounds-consistent ROBDD propagator.

	Gecode projector		$\Gamma_{(\mathcal{C})}\left((\mathcal{F})\right)$ ROBDD projector	
problem	time(ms)	fails	time(ms)	fails
`steiner`-$(2,3,7)$	8.35	6	130.50	6
`steiner`-$(3,4,8)$	64.80	60	2460	60
`steiner`-$(2,3,9)$	2036.50	4521	167170	4521
`steiner`-$(2,4,13)$	38.05	19	—-	—-

(a) Comparing projectors for $(\mathcal{S})$

	Gecode $(\mathcal{C})$		$\Gamma_{(\mathcal{C})}\left((\mathcal{F})\right)$	
problem	time(ms)	fails	time(ms)	fails
`steiner`-$(2,3,7)$	0.4535	6	196	6
`steiner`-$(3,4,8)$	5.0790	60	4140	35
`steiner`-$(2,3,9)$	45.8850	4505	101740	3103
`steiner`-$(2,4,13)$	3.3150	19	—-	—-

(b) Comparing normal $(\mathcal{C})$

Table 6.10: Comparison of the *Gecode* $(\mathcal{C})$ approximation and a mimicked $(\mathcal{C})$

7 Conclusion

In the final chapter of this thesis we conclude the preceding chapters and present some ideas on future work.

7.1 Main contributions

In the course of this thesis we presented an application of Benhamou's work on *domain approximations* to finite domain set variables. We showed, how this framework captures common variable representations available in literature and how it defines constraint propagation.

Survey We took a closer a look on how set variable representations define domain approximations by providing a survey on the set bounds domain approximation ($(\mathcal{S})$), the cardinality set bounds domain approximation ($(\mathcal{C})$), the lexicographic bounds consistency ($(\mathcal{L})$), the length-lexicographic bounds domain approximation ($(\mathcal{LL})$) and the full domain approximation ($(\mathcal{F})$).

7.1.1 Implementing domain approximations

Apart from encoding existing variable representations in terms of a domain approximation we also discussed how two of them, namely $(\mathcal{C})$ and $(\mathcal{F})$, can be implemented. Concerning the $(\mathcal{C})$ domain approximation we took a glimpse on a possible implementation with the help of the $(\mathcal{C})$ domain approximation as defined in *Gecode* . As for the $(\mathcal{F})$ domain approximation this thesis shows how Lagoon and Stuckey define and implement a set variable representation using ROBDDs.

Exchanging implementations Due to the open design and architecture of the *Gecode* - library we were able to implement a disjoint bounds representation as $(\mathcal{C})$ implementation in *Gecode* , which we compared with the including bounds representation in chapter 6.

Integrating Implementations On the other hand, this thesis provided the formal framework for our implementation of the ROBDD-based set variable representation by Lagoon and Stuckey in *Gecode* . Taking up the concept of views and iterators (Schulte and Tack[35]) , which forms an integral component of the *Gecode* architecture, we were able to integrate Lagoon and Stuckey's full domain approximation into the cardinality set bounds consistent set constraint solver in *Gecode* .

Generating propagators Moreover this thesis also proposes a way to automatically generate finite set constraints for both domain approximations ($\left(\mathcal{C}\right)$ and $\left(\mathcal{F}\right)$) from a uniform specification language as proposed by Tack *et al.*. A prototype of the automatic propagator generation has been implemented in *Gecode* and empirically evaluated in section 6.7.

Summing up Thus, this thesis is the first to integrate different set variable representations into a single constraint solver such that variable representations and finite set constraints on those variables can be combined freely using the variable view concept. Hence, we can not only implement the ROBDD approach with its benefits of different consistencies for one propagator but also plug them into *Gecode* 's set constraint solver to combine the ROBDD representation with already available set constraints.

7.2 Future Work

Though the integration of different representations into a single framework is a major contribution, still the implementation can be improved as the poor performance of some of the implemented views in section 6.6 indicates.

Bottlenecks As we are not close in touch with Hawkins *et al.* we implemented most of the proposed ROBDD representation Hawkins *et al.* literally as shown in their paper. However, we focus especially on improving our algorithms concerning cardinality reasoning of the ROBDDs since these “bottlenecks” are very time consuming as highlighted in chapter 6. This is already work in progress but not yet implemented in the presented framework.

Change of implementation Unfortunately, the DB representation did not kept what was promised by the intuition behind what we summarized in section 6.4. Since it turned out that the range-list representation in *Gecode* as presented in section 3 cannot be improved further by only changing the represented sets (see section 6.4) it would be worth investigating, whether changing the underlying data structure form range-lists to bit-vectors or similar data structures would do the trick.

Further constraints Since not all the constraints for ROBDDs as presented in [22] have been implemented during this thesis, this could also be worth having a look at.

Multisets Finally, it would be interesting whether the presented framework of this thesis could also cope with multisets by generalizing the obtained results for normal sets.

Bibliography

[1] (2000). *ILOG Solver 5.0 reference Manual*. ILOG Inc., Mountain View, CA, USA.

[2] (tbd). *Problem Solving with Finite Set Constraints in Oz. A Tutorial.* Mozart, tbd.

[3] Adams, D. (2001). *The Restaurant at the End of the Universe (Hitch Hiker's Guide to the Galaxy)*. Pan Macmillan.

[4] Apt, K. R. (1999). The rough guide to constraint propagation. In *CP '99: Proceedings of the 5th International Conference on Principles and Practice of Constraint Programming*, pages 1–23. Springer-Verlag.

[5] Azevedo, F. (2007). Cardinal: a finite sets constraint solver. *Constraints*, **12**(1), n.n.

[6] Azevedo, F. and Barahona, P. (2000). Applications of an extended set constraint solver.

[7] Azevedo, F., Gervet, C., and Pontelli, E., editors (2005). *Constraint Programming: Beyond Finite Integer Domains (BeyondFD 2005) Sitges, Spain,*, Sitges (Spain).

[8] Barnier, N. and Brisset, P. (2002). Solving the kirkman's schoolgirl problem in a few seconds.

[9] Benhamou, F. (1996). Heterogeneous constraint solving. In M. Hanus and M. Rodríguez-Artalejo, editors, *ALP*, volume 1139 of *Lecture Notes in Computer Science*, pages 62–76. Springer.

[10] Bryant, R. E. (1992). Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Comput. Surv.*, **24**(3), 293–318.

[11] Davey, B. A. and Priestley, H. A. (2002). *Introduction to Lattices and Order*. Cambridge University Press.

[12] Frisch, A. M. and Jefferson, C. (2005). Representations of sets and multisets in constraint programming. In *Proceedings of the 4th International Workshop on Modelling and Reformulating Constraint Satisfaction Problems*, pages 102–116.

[13] Frisch, A. M., Jefferson, C., Martinez-Hernandez, B., and Miguel, I. (2007). Symmetry in the generation of constraint models. In *Proceedings of the ?th International Symmetry Conference*.

[14] Gent, I. and Walsh, T. (1999). Csplib: a benchmark library for constraints. Technical report, Technical report APES-09-1999. Available from http://csplib.cs.strath.ac.uk/. A shorter version appears in the Proceedings of the 5th International Conference on Principles and Practices of Constraint Programming (CP-99).

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

[16] Gervet, C. (1995). *Set Intervals in Constraint Logic Programming*. Ph.D. thesis, L'Université de Franche-Comté.

[17] Gervet, C. (1997). Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, **1**(3), 191–244.

[18] Gervet, C. (2006). In *Handbook of Constraint Programming*. *Constraints over Structured Domains*, chapter 17, pages 603–636. Elsevier Science Publishers.

[19] Gervet, C. and Hentenryck, P. V. (2006). Length-lex ordering for set csps. In *AAAI*. AAAI Press.

[20] Gratzer, G. A. (1998). *General Lattice Theory*. Birkhauser.

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

[22] Hawkins, P., Lagoon, V., and Stuckey, P. (2005). Solving set constraint satisfaction problems using ROBDDs. *J. Artif. Intell. Res. (JAIR)*, **24**, 109–156.

[23] Kuhlmann, M. and Tack, G. (2005). Indepth-lecture constraint programming. online, http://www.ps.uni-sb.de/courses/cp-ss05/.

[24] Lagoon, V. and Stuckey, P. J. (2004). Set domain propagation using ROBDDs. In [41], pages 347–361.

[25] Lind-Nielsen, J. (1996). Buddy - a binary decision diagram package. Available from `http://buddy.sourceforge.net`.

[26] Müller, T. and Müller, M. (1997). Finite set constraints in Oz. In F. Bry, B. Freitag, and D. Seipel, editors, *13. Workshop Logische Programmierung*, pages 104–115, Technische Universität München.

[27] Pekczynski, P. (2006). *Implementation and Evaluation of Advanced Propagation Algorithms for Global Constraints*. Fopra thesis (Fortgeschrittenen-Praktikum, Saarland University , Faculty of Natural Sciences and Technology I, Department of Computer Science, Saarbrücken, Germany.

[28] Puget, J.-F. (1992). Pecos a high level constraint programming language. In *Singapore International Conference on Intelligent Systems (SPICIS)*.

[29] Puget, J.-F. (1996). Finite set intervals. In *In Proceedings of the Second International Workshop on Set Constraints, Cambridge, Massachusetts*.

[30] Sadler, A. and Gervet, C. (2004). Hybrid set domains to strengthen constraint propagation and reduce symmetries. In [41], pages 604–618.

[31] Schulte, C. (2000). *Programming Constraint Services*. Doctoral dissertation, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik, Saarbrücken, Germany.

[32] Schulte, C. and Carlsson, M. (2006). In *Handbook of Constraint Programming*. *Finite Domain Constraint Programming Systems*, chapter 14, pages 495–526. Elsevier Science Publishers.

[33] Schulte, C. and Stuckey, P. J. (2004). Speeding up constraint propagation. In M. Wallace, editor, *Tenth International Conference on Principles and Practice of Constraint Programming*, volume 3258 of *Lecture Notes in Computer Science*, pages 619–633, Toronto, Canada. Springer-Verlag.

[34] Schulte, C. and Stuckey, P. J. (2005). When do bounds and domain propagation lead to the same search space? *Transactions on Programming Languages and Systems*, **27**(3), 388–425.

[35] Schulte, C. and Tack, G. (2006). Views and iterators for generic constraint implementations. In M. Carlsson, F. Fages, B. Hnich, and F. Rossi, editors, *Recent Advances in Constraints, 2005*, volume 3978 of *Lecture Notes in Computer Science*, pages 118–132. Springer.

[36] Simonis, H. (2005). Sudoku as a constraint problem. In B. Hnich, P. Prosser, and B. Smith, editors, *Proc. 4th Int. Works. Modelling and Reformulating Constraint Satisfaction Problems*, pages 13–27.

[37] Tack, G., Schulte, C., and Smolka, G. (2006). Generating propagators for finite set constraints. In F. Benhamou, editor, *12th International Conference on Principles and Practice of Constraint Programming*, volume 4204 of *Lecture Notes in Computer Science*, pages 575–589. Springer.

[38] The Alice team (2006). The Alice system. Available from `http://www.ps.uni-sb.de/alice/index.html`.

[39] The Gecode team (2006). Generic constraint development environment. Available from http://www.gecode.org.

[40] The Mozart Consortium (2006). The Mozart programming system. http://www.mozart-oz.org.

[41] Wallace, M., editor (2004). *Principles and Practice of Constraint Programming - CP 2004, 10th International Conference, CP 2004, Toronto, Canada, September 27 - October 1, 2004, Proceedings*, volume 3258 of *Lecture Notes in Computer Science*. Springer.

[42] Wallace, M., Novello, S., and Schimpf, J. (1997). Eclipse: A platform for constraint logic programming. Technical report, IC Parc, Imperial College, London.