View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by AMS Tesi di Laurea

Alma Mater Studiorum · Università di Bologna

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea Magistrale in Informatica

A study on set variable representations in constraint programming

Tesi di Laurea in Algorithms and Data Structures

Relatore:
Prof.ssa Zeynep Kiziltan

Presentata da:
Andrea Rappini

Sessione III
Anno Accademico 2009/2010

Ai miei genitori, Pina e Marcello.

Sommario

Il lavoro presentato in questa tesi si colloca nel contesto della programmazione con vincoli, un paradigma per *modellare* e *risolvere* problemi di ricerca combinatoria che richiedono di trovare soluzioni in presenza di vincoli. Una vasta parte di questi problemi trova naturale formulazione attraverso il linguaggio delle variabili insiemistiche. Dal momento che il dominio di tali variabili può essere esponenziale nel numero di elementi, una rappresentazione esplicita è spesso non praticabile. Recenti studi si sono quindi focalizzati nel trovare modi efficienti per rappresentare tali variabili. Pertanto si è soliti rappresentare questi domini mediante l'uso di approssimazioni definite tramite intervalli (d'ora in poi rappresentazioni), specificati da un limite inferiore e un limite superiore secondo un'appropriata relazione d'ordine.

La recente evoluzione della ricerca sulla programmazione con vincoli sugli insiemi ha chiaramente indicato che la combinazione di diverse rappresentazioni permette di raggiungere prestazioni di ordini di grandezza superiori rispetto alle tradizionali tecniche di codifica. Numerose proposte sono state fatte volgendosi in questa direzione. Questi lavori si differenziano su come è mantenuta la coerenza tra le diverse rappresentazioni e su come i vincoli vengono propagati al fine di ridurre lo spazio di ricerca. Sfortunatamente non esiste alcun strumento formale per paragonare queste combinazioni.

Il principale obiettivo di questo lavoro è quello di fornire tale strumento, nel quale definiamo precisamente la nozione di combinazione di rappresentazioni facendo emergere gli aspetti comuni che hanno caratterizzato i lavori precedenti. In particolare identifichiamo due tipi possibili di combinazioni, una forte ed una debole, definendo le nozioni di coerenza agli estremi sui vincoli e sincronizzazione tra rappresentazioni. Il nostro studio propone alcune interessanti intuizioni sulle combinazioni esistenti, evidenziandone i limiti e svelando alcune sorprese. Inoltre forniamo un'analisi di complessità della sincronizzazione tra *minlex*, una rappresentazione in grado di propagare in maniera ottimale vincoli lessicografici, e le principali rappresentazioni esistenti.

Il primo capitolo è un'introduzione al contesto generale nel quale questo lavoro prende forma dettagliando in modi più precisi le motivazioni e gli scopi di questa tesi.

Nel secondo capitolo vengono introdotti i concetti e le notazioni che saranno necessarie al lettore per comprendere questa tesi. In particolare introdurremo formalmente i problemi di soddisfacimento di vincoli, le nozioni di coerenza locale e le variabili insiemistiche.

Il terzo capitolo offre una panoramica sulla letteratura più rilevante che tratta di rappresentazioni insiemistiche. In particolare ci focalizzeremo su *lengthlex*, una rappresentazione che sarà propedeutica alla comprensione del capitolo riguardante la trattabilità.

Nel quarto capitolo formalizzeremo un *framework* preliminare in grado di gestire la combinazione di due rappresentazioni, una tecnica piuttosto diffusa per ottenere migliori approssimazioni. Definiremo le necessarie nozioni di coerenza agli estremi e sincronizzazione per combinazioni binarie. Introdurremo quindi la rappresentazione *minlex*, in grado di propagare in maniera effettiva vincoli lessicografici. Compareremo la robustezza di differenti combinazioni posizionando le rappresentazioni esistenti all'interno del nostro *framework*.

Nel capitolo quinto forniremo alcuni risultati di complessità per la rappresentazione *minlex*. In particolare mostreremo che qualsiasi vincolo unario e binario trattabile per *lengthlex* è trattabile anche per *minlex*. Mostreremo quindi che la sincronizzazione tra *minlex* e le principali rappresentazioni esistenti può essere ottenuta in tempo polinomiale.

Il capitolo sesto tratterà delle generalizzazione del *framework* proposto nel capitolo terzo allo scopo di gestire un numero arbitrario di rappresentazioni. Continueremo quindi il confronto di alcune combinazioni basate su *minlex* con combinazioni esistenti, posizionandole nel nostro *framework*.

Infine il capitolo settimo conclude la tesi con una discussione dei risultati ottenuti e degli sviluppi futuri.

Contents

List of Figures

Chapter 1

Introduction

There are many constraint satisfaction problems that can be naturally formulated using set variables. They require searching set of sets which satisfy some constraints. Set variables are usually approximated using sets intervals. A powerful approach is to combine together different intervals to achieve the maximum level of pruning. In this thesis, we provide a general framework for combining different intervals in order to give a systematic approach to the study of combined approximations.

The first chapter gives an introduction to the context and highlights the motivations of this thesis.

1.1 Constraint programming

Constraints are everywhere. Many common problems in our lives can be viewed as constraint satisfaction problems (CSP). In university, we face every year the famous timetable problem. A timetabling problem can be defined as the scheduling of a certain number of lectures, which are given by a teacher and attended by a certain number of students. There are many constraints that must be satisfied at the same time: the rooms must be large enough to contain the students, a teacher cannot teach two lectures at the same time, lectures can start after 8am and must finish before 7pm and other constraints. Another typical example of CSP can be the well-known Sudoku puzzle: in such a game the objective is to fill a 9×9 grid with digits so that each column, each row, and each of the nine distinct 3×3 boxes contains all of the digits from 1 to 9, as shown in Figure 1.1.

A constraint satisfaction problem is defined as a set of variables, each one with its domain, and a set of constraints which specify the solutions [16]. A solver implements variables, constraints and a solution procedure which

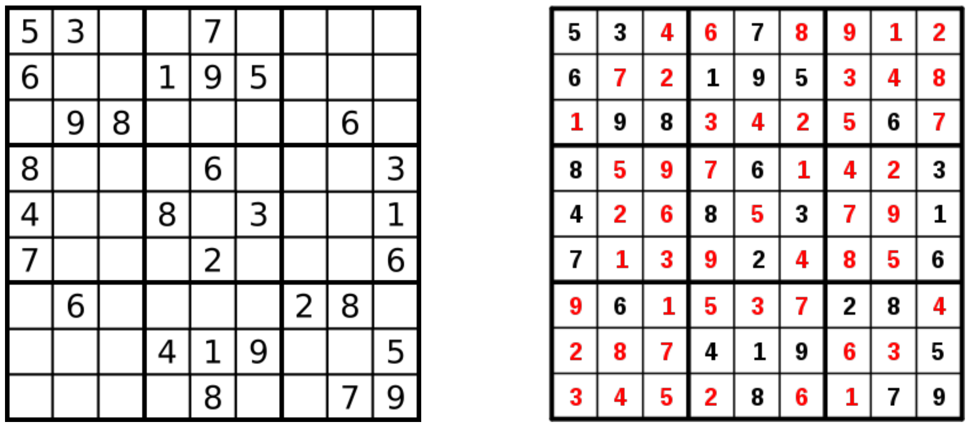

Figure 1.1: A sudoku puzzle (left) and its solution (right).

tries to find an assignment to the variables such that it satisfies all of the constraints. Solvers are usually embedded within a programming language or provided via separate software libraries. A solver exploits two techniques to find solutions: inference and search. Inference involves constraint reasoning and it is able to forbid the assignment of values or combination of values to the variables only by looking at the constraints. On the other hand, search is based on backtracking and it is an exhaustive procedure to find a solution. Search efficiency is improved by inference.

Constraint programming (CP) consists of two phases that are strongly interconnected:

1. modelling;
2. solving.

When a CP user models a problem, he must define the variables, the domains and specify the solutions by posting the constraints. Modelling is a critical aspect because it requires to formalize the human understanding of a problem, choosing the right variables, constraints, etc. A bad model can result in poor performance or it could yield to wrong results.

From the user point of view, constraint programming is a declarative programming paradigm: a programmer must not focus in how to solve the problem, like we normally do with traditional imperative languages, but on how to model it. It will be the solver that will take care of resolving the problem. However constraint programming is also traditional computer programming, as the user often must program a strategy to find a solution in an efficient way.

1.2 Motivations and aim of this thesis

Set variables (i.e. variables whose domain is a set of sets) are natural objects for modelling a wide range of problems. For instance, consider the problem of packing items in a bag. Such a bag can be modelled using a single set variable as the order of the elements is irrelevant.

In general, the real domain of a set variable is exponential in size. Therefore it is often represented with an approximation. An approximation is an interval specified by a lower bound and an upper bound. The first approximation is due to Puget [15] and Gervet [10] and exploits the traditional subset $\subseteq$ ordering. The values in the lower bound are called definite values because each set which belongs to the domain must contain them. The values in the upper bound are called possible element, since a set can contain only those elements. This approximation has some weaknesses: for example it is not able to deal well with cardinality constraints as this information is not explicitly represented. To overcome this weakness Cardinal [1] enhanced subset bounds with cardinality bounds, integrating different approximations. Promising results in this new type of approximation has opened a new research in set variable representation and consequently many interesting combined approximations have been studied [19, 11, 1]. The only work that rejects this assumption is the one by Hawkins, Lagoon and Stuckey [12]. They completely represent a domain using ROBDD (Reduced Ordered Binary Decision Diagram), without any loss of information. However experimental evaluations [28] have shown that combining together different set approximations outperforms the state-of-the-art solver which uses ROBDD. This is because maintaining ROBDD during propagation is expensive.

These previous studies have differed on how approximations are combined, how consistency between intervals is maintained, and how the constraints are propagated in order to reduce the search space. Unfortunately no formal framework exists to compare these combined approximations. Our goal is to provide a framework in which we define the notion of combining approximations, characterize some common issues that naturally arise when two or more approximations are combined (e.g. consistency between domains). We then want to compare the strength of the different combinations and position existing combinations within our framework. Finally we want to give some complexity results on maintaining consistency between domains.

1.3 Overview

In the next chapter, we introduce the concepts and the notations that are needed to the reader to understand this thesis. In particular, we formally introduce constraint satisfaction problems, local consistencies, search and set variables. In Chapter 3 we summarize the most relevant literature regarding set variable representations. In particular, we will focus on *lenghtlex*: a representation which will be studied throughout this thesis. Then the rest of this thesis is divided in 3 chapters with the following contributions.

A framework for combining two representations In Chapter 4, we formalize a framework to combine two representations which is a popular way of combining representations. We characterize two types of combinations and we define bound consistency on them. Then we introduce a new representation, called *minlex*, and we study its various combinations with the most common existing representations. We then compare the strength of these *minlex* based combinations and we position the existing mutually combined representations within our framework.

Tractability In Chapter 5, we provide some complexity results for the *minlex* representation. In particular, we show that every unary and binary constraint that is tractable for *lengthlex*, is tractable also for *minlex*. We also show that consistency between *minlex* and the most common existing representations can be achieved in polynomial time.

Generalizing the framework In Chapter 6, we generalize the framework introduced in Chapter 3 to deal with an arbitrary number of representations, given that the recent is to consider more than two representations at a time. We then focus on the combination of three representations. We further compare the strength of *minlex* based combinations and position in our framework the existing combinations that take into account three representations.

Chapter 7 concludes the thesis, with a discussion of the obtained results and future work. Part of the hereby presented work is submitted to The Twenty-Fifth Conference on Artificial Intelligence (AAAI 2011).[1]

[1]http://www.aaai.org/Conferences/AAAI/aaai11.php

Chapter 2

Background

In this chapter, we introduce the concepts and the notations that are needed to the reader to understand this thesis. In particular, we formally introduce constraint satisfaction problems, local consistencies, search and set variables.

2.1 Constraint satisfaction problem

Following [16], a constraint satisfaction problem (CSP) is defined as follows:

Definition 1 *A CSP is a triple $\langle X, D, C\rangle$ where:*

- *X is a n-tuple of variables $X = \langle X_1, X_2, \ldots, X_n\rangle$;*
- *D is a n-tuple of domains $D = \langle D(X_1), D(X_2), \ldots, D(X_n)\rangle$ where $D(X_i)$ is the domain of the variable X_i;*
- *C is a t-tuple of constraints $\langle c_1, c_2, \ldots, c_t\rangle$ where c_j is a relation defined on a sequence of variables $var(c_j) = \langle X_l, \ldots, X_m\rangle$. c_j is the subset of $D(X_l) \times \ldots \times D(X_m)$ that contains the combinations of values that are allowed by c_j.*

A constraint can be specified extensionally by the list of its satisfying tuples or intensionally by a formula. $|var(c)|$ is the arity of the constraint. Constraints of arity 1 are called unary constraints whilst constraint of arity 2 are called binary constraints.

Example 1 (Modelling sudoku) *The sudoku problem can be formulated as CSP as follows:*

- *$9{\times}9$ variables, one for each cell, $X_{i,j}$ with domains $D(X_{i,j}) = \{1, \ldots, 9\}$;*

- *not equals constraints on the rows, columns and boxes. For example:*
  - *alldifferent(*$X_{1,1}$,$X_{1,2}$,$X_{1,3}$, …, $X_{1,9}$*);*
  - *alldifferent(*$X_{1,1}$,$X_{1,2}$,$X_{1,3}$, …, $X_{1,9}$*);*
  - *alldifferent(*$X_{1,1}$,$X_{1,2}$,$X_{1,3}$,$X_{2,1}$,$X_{2,2}$,$X_{2,3}$,$X_{3,1}$,$X_{3,2}$,$X_{3,3}$*);*

where the alldifferent constraint states that all the variables must be assigned to different values.

A variable *assignment* or *instantiation* is an assignment to a variable X_i of one of the values from $D(X_i)$. A *partial assignment* is an assignment to some but not all $X_i \in X$ whilst a *total assignment* is an assignment to every $X_i \in X$. We denote with A a partial assignment and with $A[\{X_i, \ldots, X_j\}]$ the projection of A on variables $X_i, \ldots, X_j$. A partial assignment A is *consistent* if and only if for all constraints $c(X_l, \ldots, X_m) \in C$, $A[\{X_l, \ldots, X_m\}] \in c(X_l, \ldots, X_m)$. A *solution* to a CSP is a consistent total assignment. We say that a partial assignment A can be *extended* to a solution if and only if it is consistent. An assignment or a partial assignment that it is not consistent is an *unsatisfying assignment.*

Example 2 *Consider again the sudoku puzzle of Figure 1.1.* $X_{1,3} = 4$ *is a variable assignment and it is also a consistent partial assignment since it can be extended to a solution. The assignment induced by the right puzzle of Figure 1.1 is a solution.* $X_{1,3} = X_{1,4} = 4$ *is a failure.*

In general solving CSPs is NP-hard. Despite this general intractability, constraint programming provides a platform to solve CSPs for many real problems [23]. Once a problem is modelled as a CSP a solver solves it by searching the set of solutions. The search space is given by the cartesian product of each domain $D(X_i)$. However the search space is reduced thanks to the inference done by local consistencies. The hope is that the reduction is enough to solve the CSP in polynomial time.

2.2 Local consistencies

Inference is a form of reasoning which eliminates parts of the search space by examining the variables, their domains and the constraints. Clearly inference must preserve the set of solutions.

Example 3 *Take for example the following constraint* $X_1 < X_2$ *on two integer variables* X_1 *and* X_2*. Assume that* $D(X_1) = \{4, 5, 6\}$ *and* $D(X_2) = \{1, 2, 3\}$*. Clearly this constraint has no solution: the smallest element of* $D(X_1)$ *is in fact greater than the greatest element of* $D(X_2)$*.*

Since complete inference is infeasible, we classify the amount of reasoning which a solver is able to do with local consistencies. A local consistency is a property that must be satisfied by a CSP. It is called local because it considers individual constraints. Therefore it is an incomplete inference.

Generalized Arc Consistency As presented in [16], arc consistency (AC) is the oldest and most well-known form of local consistency. Arc consistency guarantees that every value in a domain can be extended to a solution. AC is defined only for binary constraints.

Given a constraint $c(X_l, \ldots, X_m)$, a *support* is a tuple $\langle d_l, \ldots, d_m \rangle \in c$ where $d_i \in D(X_i)$. Now we are ready to give the formal definition of generalized arc consistency (GAC) which is defined for constraints of any arity. This definition includes also the definition of AC.

Definition 2 (Generalized Arc Consistency) *Given a CSP $\langle X, D, C \rangle$, a constraint $c \in C$, c is generalized arc consistent if and only if $\forall X_i \in var(c)$, $\forall v \in D(X_i)$, v belongs to a support.*

A CSP is GAC if and only if all its constraints are GAC.

Example 4 *Let X_1, X_2 and X_3 be three variables. Assume that $D(X_1) = D(X_2) = D(X_3) = \{1, 2, 3\}$. Consider the constraints $c_1(X_1, X_2) \equiv (X_1 = X_2)$ and $c_2(X_2, X_3) \equiv (X_2 < X_3)$. This CSP is not arc consistent. Indeed when we check c_2, we see that $X_2 = 3$ cannot be extended to a solution.*

Note that local consistencies are only properties of a CSP. It is not specified how to enforce these properties. A *propagator*, or *propagation algorithm*, is an algorithm which enforces the desired local consistency for a CSP. The process of examining a constraint is called *propagation*. Propagation may remove values from the domains of the variables when local consistencies are enforced. This removal is also known as *pruning*.

Example 5 *Consider example 4: when we check c_2, we see that $3 \in D(X_2)$ must be pruned because it can not be supported by any value in $D(X_3)$. Removing 3 from $D(X_2)$ causes propagation which in turn removes 3 from $D(X_1)$ because of constraint c_1.*

It has been proved [4] that GAC is in general NP-hard to achieve. During the years many algorithms have been proposed for enforcing AC and GAC. The most well-known propagator is the one proposed by Mackworth under the name AC3 which runs in $O(ed^3)$, where e is the number of constraints and d the domain size. Nowadays the best GAC propagator for any kind of CSP runs in $O(erd^r)$, where r is the largest arity of a constraint. It is possible to obtain better performance exploiting the constraint semantics .

Consistencies weaker than GAC It is possible to define weaker levels of consistencies than GAC. In fact sometimes GAC could be too expensive to maintain. As a result, many proposals have been done during the years. Here we cover only bound consistency since it will be extensively used throughout this thesis.

The main idea behind bound consistency is to exploit the fact that domains are composed of integers. Thus they inherit the traditional total ordering $\leq$ on $\mathcal{Z}$. Given a domain $D(X_i)$, we denote with $lb_{\mathcal{Z}}(X_i)$ and $ub_{\mathcal{Z}}(X_i)$ the smallest and the biggest element of $D(X_i)$ under the ordering of $\mathcal{Z}$. We can relax the domain of $D(X_i)$ as $\{lb_{\mathcal{Z}}(X_i), \dots, ub_{\mathcal{Z}}(X_i)\}$. Given a constraint $c(X_l, \dots, X_m)$, a *bound support* is a tuple $\langle d_l, \dots, d_m \rangle \in c$ where $d_i \in \{lb_{\mathcal{Z}}(X_i), \dots, ub_{\mathcal{Z}}(X_i)\}$. Therefore bound consistency (BC) is formally defined as follows:

Definition 3 (Bound consistency) *Given a CSP $\langle X, D, C \rangle$, a constraint $c \in C$, c is bound consistent if and only if $\forall X_i \in var(c)$, $lb_{\mathcal{Z}(X_i)}$ and $ub_{\mathcal{Z}(X_i)}$ belong to a bound support.*

A CSP is BC if and only if all its constraints are BC.

Note that since bound consistency is obtained by relaxing arc consistency, we have that every CSP that is arc consistent is also bound consistent but not vice versa. That means that arc consistency is stronger than bound consistency.

Example 6 *Take $X_1, \dots, X_6$ variables and assume that $D(X_1) = D(X_2) = \{1, 2\}$, $D(X_3) = D(X_4) = \{2, 3, 5, 6\}$, $D(X_5) = \{5\}$ and $D(X_6) = \{3, 4, 5, 6, 7\}$. Consider the constraint alldifferent($X_1, \dots, X_6$) that imposes that the values assigned to the variables must be different. After the BC propagation on alldifferent we have $D(X_1) = D(X_2) = \{1, 2\}$, $D(X_3) = D(X_4) = \{3, 5, 6\}$, $D(X_5) = \{5\}$ and $D(X_6) = \{3, 4, 5, 6, 7\}$. Clearly this domain is not arc consistent, in fact $5 \in D(X_4)$ does not have a support. If we propagate GAC we obtain $D(X_1) = D(X_2) = \{1, 2\}$, $D(X_3) = D(X_4) = \{3, 6\}$, $D(X_5) = \{5\}$ and $D(X_6) = \{4, 7\}$.*

Consistencies stronger than GAC For the sake of completeness, the reader should know that there exist other forms of local consistencies stronger than AC and for which several propagation algorithms have been developed: just to name a few we remind path consistency by Montanari (more in general the full class of k-consistencies) and triangle based consistencies. However such consistencies are beyond the scope of this thesis. See [16] for an introduction.

2.3 Search

Backtracking search is a fundamental technique for solving constraint satisfaction problems. If a solution exists, eventually it will be found. Traditionally backtracking search is represented with a tree, where each node defines a partial assignment and each branch defines a variable assignment. Backtracking search builds up a partial assignment choosing values for the variables. At each node the consistency of the corresponding partial assignment is checked. In case of inconsistency, a deadend is reached and backtracking search undoes the last choice and tries another. It is different from agnostic brute force since it does not wait a complete instantiation to evaluate the assignment.

As explained in [16], there are many ways to improve backtracking search. Backtracking search performance highly depends on some decisions that have to be taken during the search such as which variable to branch on and which value to assign to the variable. It has been shown that for many problems, the choice of variable and value ordering can be crucial to effectively solve the problem. Another option is to interleave inference with search, called constraint propagation. A further effective technique to improve backtracking search is to add implied constraints. Implied constraints are constraints that do not change the set of the solutions but help pruning the search space. A detailed description can be found in [16].

2.4 Set variables

A set variable is a variable that takes as value a subset of a universe $U = \{1, \dots, n\}$. A set of k elements from the universe is denoted by $\{s_1, s_2, \dots, s_k\}$ where $s_1 < s_2 < \dots < s_k$.

The domain $D(S)$ for a set variable S can potentially contain 2^n values (the size of the power-set $\mathcal{P}(U)$). For example, if $U = \{1, 2\}$ then $\mathcal{P}(U) = \{\{\}, \{1\}, \{1, 2\}, \{2\}\}$. In the rest of this thesis, we will use the letter S when we mean a set variable.

The real domain of a set variable is usually approximated by a representation. A representation R of a set variable is a subset of $\mathcal{P}(U)$ and it is defined by a single lower and upper bound. Representations are usually limited to subsets of $\mathcal{P}(U)$ that form an interval in some partial or total order on $\mathcal{P}(U)$. The *original* domain $D(S)$ of a set variable is approximated by a representation R as $D_R(S)$ which is described by its lower bound $lb_R(S)$ and upper bound $ub_R(S)$. If $D_R(S)$ is the smallest interval containing $D(S)$, we call it the *tightest* approximation of $D(S)$. As we will show in Chapter 3, there exist many representations proposed for set variables.

Set variables are a powerful modelling tool. They can for instance help modelling groups of elements. As an example, consider the problem of packing items in a bag. Such a bag can be modelled using a single set variable as the order of the elements is irrelevant. More examples of CSPs modelled with set variables can be found in [16].

Chapter 3

Related work

In this chapter we cover most of the existing literature for set variable representations. We will present the related work in chronological order, reflecting the various attempts to improve set variable representations.

3.1 Subset bounds representation

The first representation which received a great attention from the CP community is the subset bounds (SB) representation proposed in [10, 15]. This representation is based on the traditional subset partial order. Hence a set variable is represented by an interval whose lower and upper bounds are known sets. The elements that belong to the lower bound are called definite elements because each set in the interval must contain them. The elements that belong to the upper bound are called possible elements because each set in the interval can not contain elements that are not contained in the upper bound. More formally , given a set variable S, $D_{SB}(S) = \{s \mid lb_{SB}(S) \subseteq s \subseteq ub_{SB}(S)\}$. Figure 3.1 shows an example of subset bounds domain. Each line represent a subset relation between two sets. The domain that is represented by the lattice is $\{s \mid \{\} \subseteq s \subseteq \{1, 2, 3, 4\}\}$.

A formal and practical framework, Conjunto, which works with this set representation has been defined [10]. The set of primitive constraints that are supported are:

- domain constraints: $S \in [a, b]$ where S is a set variable and a and b are the domain bounds;
- constraints of type $S_1 \subseteq S_2$, where S_1 and S_2 are set variables;
- constraints of type $f(S_1) \in [m, n]$, where f is a monotonically increasing function from $(D_{SB}(S_1), \subseteq)$ to $\mathcal{N}$.

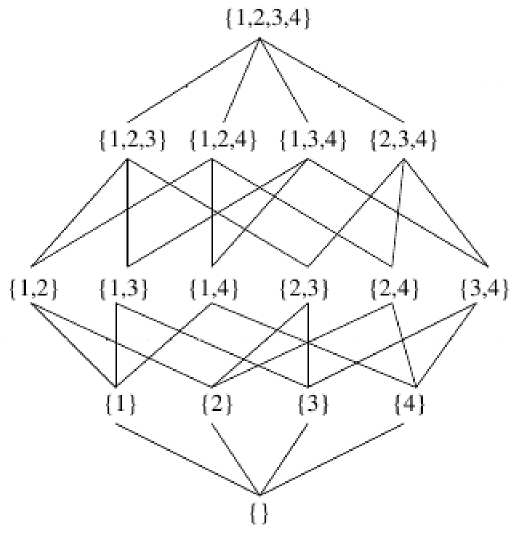

Figure 3.1: Subset bounds representation: the lower bound is $\{\}$ and the upper bound is $\{1, 2, 3, 4\}$.

Thanks to the set interval calculus proposed in [10], it is possible to generalize these constraints to n-ary constraints. Indeed a constraint of type $(S_1 \cup S_2) \subseteq (S_3 \cap S_4)$ can be decomposed in:

$$(S_1 \cup S_2) = S_{12}, (S_3 \cap S_4) = S_{34}, S_{12} \subseteq S_{34}$$

where S_{12} and S_{34} are new variables whose domains are obtained via the set interval calculus.

Bound consistency (BC) for the subset bounds representation is defined (in e.g. [25, 3]) as follows: given set variables $S_1, \ldots, S_n$, given a constraint $c(S_1, \ldots, S_n)$, S_i is BC on c if and only if $lb_{SB}(S_i)$ (resp. $ub_{SB}(S_i)$) is the intersection (resp. union) of all the sets for S_i that belong to an assignment of all S_j within their bounds satisfying the constraint c. The constraint c is BC if and only if all the variables S_i in the constraint are BC.

Cardinal Cardinal [1] is a finite sets constraint solver, publicly available in ECLiPSe Prolog [7], that exploits inferences over set cardinalities. Essentially such solver extends the subset bounds representation adding also cardinality bounds. *card* (also denoted by C) is a representation that contains only elements from $\mathcal{P}(U)$ that can be represented by a lower bound and an upper bound on the cardinality of the sets they contain. That is, in the representation *card*, $D_C(S) = \{s \mid lb_C(S) \leq |s| \leq ub_C(S)\}$, where $lb_C(S)$ and $ub_C(S)$ represent the smallest cardinality and the greatest cardinality of sets in $D_C(S)$. Bound consistency for *card* is defined as follows. Given set variables $S_1, \ldots, S_n$, given a constraint $c(S_1, \ldots, S_n)$, S_i is BC on c if and only if $lb_C(S_i)$ (resp. $ub_C(S_i)$) is the smallest (resp. greatest) cardinality of all the sets for S_i that belong to an assignment of all S_j within their bounds satisfying the constraint c. The constraint c is BC if and only if all the variables S_i in the constraint are BC. In Cardinal, BC is maintained independently for the two representations and inference rules are given in order to maintain consistency between the two domains.

3.2 Hybrid representation

In [19, 18] is proposed an hybrid domain to overcome some of the weaknesses of the traditional subset bound solvers. It enriches the *SB* and *card* representation with lexicographic bounds. The lexicographic representation has a strong analogy with the bounds representation for integer variables, that is space and time efficient for simple operations.

Definition 4 *The lexicographic ordering $\leq_{lex}$ on sets is defined as follows:*

$$s \leq_{lex} t \text{ iff } s = \{\} \vee x < y \vee x = y \wedge s \setminus \{x\} \leq_{lex} t \setminus \{y\}$$

where $x = max(s)$ and $y = max(t)$ and max denotes the largest element of s or t.

Thus, in this representation $D_{\leq_{lex}}(S) = \{s \mid lb_{\leq_{lex}}(S) \leq_{lex} s \leq_{lex} ub_{\leq_{lex}}(S)\}$.

A point of strength of this representation is its effectiveness in propagating lexicographic ordering constraints which are powerful tools to break symmetries in many CSPs. The main drawback of this representation is its inability to capture some fundamental constraints as the inclusion or the exclusion of a single element. For this reason in [18, 19] they propose to use simultaneously the three representations. Given a variable S, the hybrid domain specifies the set:

$$\{s \mid s \in D_{\leq_{lex}}(S) \wedge s \in D_{SB}(S) \wedge s \in D_C(S)\}$$

The authors then define a set of rules to maintain the three domains mutually consistent. In addition to that, they define also a set of inference rules for some common constraints. Their prototype is implemented in ECLiPSe [7]. Experimental results demonstrate that this approach improve both *SB* and its combination with *card* on common combinatorial design problems (CDP). Moreover it is also effective for symmetry breaking.

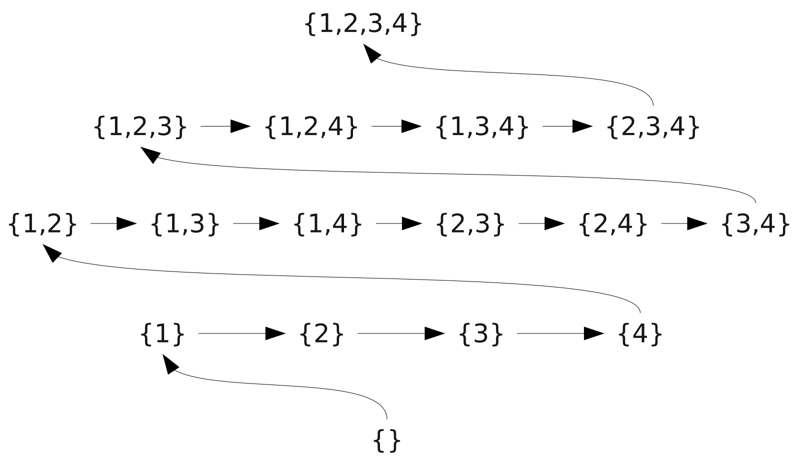

Figure 3.2: Lengthlex ordering.

3.3 *lengthlex* representation

lengthlex (also denoted by *LL*) was proposed in [11] to overcome the inherent difficulties of the subset bounds representation in handling cardinality and lexicographic constraints. *lengthlex* encodes directly cardinality and lexicographic information and totally orders the set domain. The key idea is to totally order the domain first using cardinality and then lexicographically.

Basically a *lengthlex* ordering is defined as follows:

Definition 5 *A lengthlex ordering $<_{LL}$ on sets is defined by:*

$s <_{LL} t$ *iff* $s = \{\} \vee |s| < |t| \vee |s| = |t| \wedge (s_1 < t_1 \vee s_1 = t_1 \wedge s \setminus \{s_1\} <_{LL} t \setminus \{t_1\})$

Example 7 *The subsets of* $\{1,2,3\}$ *are ordered as follows* $\{\} <_{LL} \{1\} <_{LL} \{2\} <_{LL} \{3\} <_{LL} \{1,2\} <_{LL} \{1,3\} <_{LL} \{2,3\} <_{LL} \{1,2,3\}$.

Given a set variable S, the *lengthlex* domain is a pair $\langle lb_{LL}(S), ub_{LL}(S)\rangle$ and it denotes the set $D_{LL}(S) = \{s \mid lb_{LL} \leq_{LL} s \leq_{LL} ub_{LL}\}$. Figure 3.2 represents a domain whose lower bound is empty and the upper bound is $\{1,2,3,4\}$. Each arrow represents a *lengthlex* relation between two sets.

Example 8 *Take variable S with domain* $\langle\{1,2\},\{1,2,3\}\rangle$. *The domain of S is the set* $\{\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$. *If the cardinality of S is constrained to be smaller than* 2*, then the domain of S can be restricted to* $\{\{1,2\},\{1,3\},\{2,3\}\}$ *that are all the sets of cardinality* 2.

Since *lengthlex* is a total order, BC is defined exactly as defined for integer domains (see Chapter 2). In the following we will cover in detail *lengthlex*, since Chapter 5 will reuse some of the results for this representation.

Unary and binary constraints The authors observe that pruning of a totally ordered domain, as *lengthlex*, consists of finding the first element after the lower bound and the last element before the upper bound satisfying some conditions, such as inclusion or exclusion of a set of elements and cardinality restrictions. Thus they define a number of polynomial $first$ and $last$ functions to perform such computations. These algorithms share the overall structure as the algorithm to compute the lengthlex successor of a $k - set$, due to Kreher and Stinson [13]. Thus the authors define a $first - r_{LL}(s, r)$, which computes the first successor of s that contains all elements in r.

Example 9 *Consider* $s = \{1, 3, 6, 7\}$ *and* $r = \{3, 4\}$*. Then* $first - r_{LL}(s, r)$ *returns the first set after* s*, with respect to lengthlex ordering, which contains the set* $\{3, 4\}$*. Such set is* $\{2, 3, 4, 5\}$*.*

With similar purposes, and similar construction, they build also a $first - e_{LL}(s, e)$ function which returns the first set after s which is disjoint from e.

Example 10 *Consider* $m = \{1, 7, 8\}$*,* $U = \{1, \ldots, 8\}$ *and* $e = \{3, 5, 7\}$*. Then* $first - e_{LL}(s, e)$ *returns the first set after* s*, with respect to lengthlex ordering, which is disjoint from* $\{3, 5, 7\}$*. Such set is* $\{2, 4, 6\}$*.*

The $first$ function which works for cardinality constraint is very simple as it does not need a real location phase. Indeed $first - c_{LL}(s, c)$ must return s if s has the desired cardinality, otherwise if $|s| < c$ then it can safely return $\{1, 2, \ldots, c\}$ since it is the smallest set of cardinality c according to lengthlex ordering.

Given this functions it is easy to build a set of inference rules that maintains bound consistency. They maintain BC for a set of basic unary constraint which is $\{s \subseteq S, s \oplus S, |S| \leq d, |S| \geq c\}$ where $\oplus$ denotes disjointness. The semantic is specified in terms of rewriting rules that manipulate configurations $\langle \gamma, \sigma \rangle$, where γ is a conjunction of constraints and σ is the domain store (i.e. a conjunction of domain constraints). For example, the rewriting rule for the inclusion constraint ($r \subseteq S$) is the following: $\langle r \subseteq S, S \in \langle lb, ub \rangle \rangle \mapsto \langle r \subseteq S, S \in \langle lb', ub' \rangle \rangle$ if $lb' = first - r_{LL}(lb, r)$ and $ub' = last - r_{LL}(ub, r)$. All the other rules are written in a similar form.

After these unary constraints, they show how to enforce BC also for the lexicographic binary constraint ($S_1 <_{LL} S_2$). A great strength of *lengthlex* is that lexicographic constraints are also arc consistent, because of the total ordering of the domain. However their lexicographic binary constraint is w.r.t. *lengthlex* ordering, so it considers also cardinality. Finally in the mentioned paper they give rules for binary disjointness and a constraint which combines disjointness and lexicographic constraints.

BC algorithm for binary constraints In [22] is presented a generic bound-consistency algorithm for any binary constraint which requires $O(n^2)$ calls, n size of the universe, to a feasibility subroutine of the binary constraint. Although NP hard in general, they present algorithms to enforce bound consistency on disjoint and cardinality constraints in time $O(n^3)$, n size of the universe. Their bound consistency algorithm only assumes the existence of an algorithm to check whether a constraint has a solution in two *lengthlex* intervals. The definition of the generic bound consistency algorithm relies on three functions: a function to check the existence of a solution (the feasibility subroutine) and two functions that are equal to the first and last functions used for unary constraints but that are extended to two variables. Leaving aside the technical details, the main result of this work is a generic bound consistency algorithm which is able to enforce BC in $O(n^2\alpha)$ where α is the computational cost of the feasibility routine and a specialized $O(n^3)$ algorithm of the disjoint constraint that can be generalized to $atmost - k$ and $atleast - k$. These results should extend to constraints of arity k, with an algorithm which runs in time $O(c^k\alpha)$, where c is the cardinality of the sets.

The knapsack constraint Given a set of items modeled by a set variable S which takes values from a universe of n items, a profit vector p which gives the non-negative profit of each item, a weight vector w which gives the weight of each item, a minimum profit B and a weight capacity C of the knapsack, the knapsack constraint $KP(S, p, w, B, C)$ requires that the solution $s \in D(S)$ satisfy the following two constraints: $\sum_{i\in s} p_i \geq B$ and $\sum_{i\in S} w_i \leq C$. For these constraints, BC can be enforced independently in polynomial time. In [27], the authors claim that BC on the knapsack constraint can be achieved by BC on each of these constraints. However [20] proves, theoretically and empirically, that:

- the fixpoint problem for this domain representation is NP-hard in general;
- for a tractable sub-family of Knapsack this encoding takes more time than exponential brute-force enumeration;
- experimental results on these constraints show that exponential-time fixpoint computation is the rule and not an exception.

Finally in [20] the author observes that the intractability proof can be applied to every representation in which sets are totally ordered.

Exponential propagation In [28] the authors put the focus in the exponential propagation for set variables. Indeed the constraint propagation algorithm for many elementary constraints over a *lengthlex* domain may take exponential time to converge to a fixpoint, as shown in the previous section. Moreover there are many set constraints that are generally intractable [3]. This abundance of negative results could discourage research in the direction of richer set representations. Surprisingly they show that richer representations, such as that which combine *lengthlex* and SB, may be highly beneficial in practice. It is motivated by the fact that reasonable exponential behavior in the filtering algorithm may produce a significant reduction of search space. This has also beneficial effects on constraint propagation allowing further reduction of the search space.

3.4 Sets as ROBDD

Hawkins, Lagoon and Stuckey [12] have proposed an interesting approach to set representations for set variables. In their work, they refuse to represent a domain with an approximation. In fact they fully represent a domain specifying its characteristic function using a reduced ordered binary decision diagram. The key idea is the following: each set can be represented as a boolean formula. For example, consider the set $\{1\}$ that takes values from the universe $U = \{1, 2, 3\}$. Then we can represent $\{1\}$ with the boolean formula $v_1 \wedge \neg v_2 \wedge \neg v_3$, where v_i is true if and only if i belongs to the set. Now it is easy to see that we can represent the domain of a set variable as a disjunction of boolean formulae.

Example 11 *Given a set variable S and the universe $U = \{1, 2, 3\}$, the domain $D(S) = \{\{1\}, \{1, 3\}, \{2, 3\}\}$ is represented by the formula $(v_1 \wedge \neg v_2 \wedge \neg v_3) \vee (v_1 \wedge \neg v_2 \wedge v_3) \vee (\neg v_1 \wedge v_2 \wedge v_3)$.*

However they can not represent the formulae in this extended form (they would be exponential in the size of the universe), thus they exploit binary decision diagram (BDD). A BDD is a direct acyclic graph which contains a set of decision nodes plus two terminal nodes called 0 and 1. Each decision node is labeled by a boolean variable and has two outgoing edges: the true edge and the false edge. Each path from the root to the terminal node 1 (resp. 0) represent an assignment for which the represented boolean function is true (resp. false). A Reduced Ordered Binary Decision Diagram is a specialization of a Binary Decision Diagram which permits efficient implementation of traditional boolean operations. In their case the decision nodes are the variables v_i.

Surprisingly many primitive set constraints and consistency levels can be easily modeled using boolean formulae and then ROBDD. Various conjunctions of all these formulae give propagators for many constraints.

Chapter 4

A framework for combining two representations

In this chapter we provide a formal framework for combining two representations and comparing them. Mutual combinations are a popular way of combining representations. We characterize some aspects that naturally arise when two representations are used at the same time for the same set variable. For example a desirable property is that two representations must agree on their bounds, that is they must be synchronized. Then we precisely define how bounds consistency can be understood in the case of combining two representations. Hence we provide the needed tools to compare combinations. Once the framework is properly defined, we instantiate it in order to launch a systematic study of representations for set variables. Our study provides insight into existing combined representations and reveal new and potentially useful representations. Finally, it discovers a number of surprises. For example, we prove that the *lengthlex* representation for set variables (which combines lexicographic ordering and cardinality information and is designed to exploit cardinality and ordering constraints) is no better on cardinality and ordering constraints than a representation for set variables that deals with the lexicographic ordering and cardinality information separately.

4.1 Revisiting the representations

In this section we summarize some of the representations that have been presented in Chapter 3 and we slightly modify the *card* representation in order to provide an adequate abstract definition of bounds consistency for a single representation.

One approach to represent set variables is based on the subset partial order [15, 10]. This is the subset bounds (SB) representation. More formally, given a variable S, $D_{SB}(S) = \{s \mid lb_{SB}(s) \subseteq s \subseteq ub_{SB}(s)\}$. Another approach is to define a total ordering $\leq_T$ on $\mathcal{P}(U)$ and to represent the domain through a lower bound lb_T and an upper bound ub_T. That is, in the representation T, $D_T(S) = \{s \mid lb_T(S) \leq_T s \leq_T ub_T(S)\}$. This is the case in [11, 19] where the sets are ordered lexicographically, though in two different ways. We here call *minlex* (also denoted by ML) the total ordering in which sets are ordered by $\leq_{ML}$ defined the following way: elements from sets are ordered from smallest to greatest and the lists obtained in this way are ordered lexicographically. Given a variable S, $lb_{ML}(S)$ and $ub_{ML}(S)$ represent the smallest and the greatest sets (w.r.t $\leq_{ML}$) in $D_{ML}(S)$. *lengthlex* (also denoted by LL) is a representation in which sets are ordered according to $\leq_{LL}$, that is, by increasing size, and ties are broken with $\leq_{ML}$. Given a variable S, $lb_{LL}(S)$ and $ub_{LL}(S)$ represent the smallest and the greatest sets (w.r.t. $\leq_{LL}$) in $D_{LL}(S)$. The hybrid domain in [19] orders elements in the sets from greatest to the smallest element as opposed to *minlex*, and then orders lexicographically the resulting lists. We refer to this ordering as *maxlex*.

We now redefine the *card* representation. Given a universe U take a set $s \in \mathcal{P}(U)$. We denote with:

$$[s] = \{s' \mid |s'| = |s|, s \in \mathcal{P}(U)\}$$

the equivalence class of s w.r.t. the equivalence relation based on cardinality.

We denote with $\mathcal{C}(U)$ the set of all equivalence classes given a universe U.

Example 12 *Take the universe* $U = \{1, 2, 3\}$*. Then* $[\{3\}] = \{\{1\}, \{2\}, \{3\}\}$ *and* $\mathcal{C}(U) = \{\{\{\}\}, \{\{1\}, \{2\}, \{3\}\}, \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}, \{\{1, 2, 3\}\}\}$.

Given two sets $s_1, s_2 \in \mathcal{P}(U)$, we define the following ordering $\preceq_C$:

$$[s_1] \preceq_C [s_2] \textit{ iff } |s_1| \leq |s_2|$$

$\preceq_C$ is a total ordering on $\mathcal{C}(U)$.

card is a representation that contains only elements from $\mathcal{P}(U)$ that belong to an equivalence class that is between a lower bound and an upper bound. That is:

$$D_C(S) = \{s \mid lb_C(S) \preceq_C [s] \preceq_C ub_C(S), lb_C(S) \in \mathcal{C}(U), ub_C(S) \in \mathcal{C}(U)\}$$

Example 13 *Take the universe $U = \{1, 2, 3\}$ and a set variable S which takes values from U. Assume that $lb_C(S) = \{\{1\}, \{2\}, \{3\}\}$ and $ub_C(S) = \{\{1, 2, 3\}\}$. Then $D_C(S) = \{\{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$.*

Given a set $A \subseteq \mathcal{P}(U)$, we define the greatest lower bound $glb_C(A)$ w.r.t. the *card* ordering as follows:

$$glb_C(A) = glb_{\preceq_C}(\{[a] \mid a \in A\})$$

Similarly $lub_C(A)$:

$$lub_C(A) = lub_{\preceq_C}(\{[a] \mid a \in A\})$$

Instead of writing every time the equivalence class, we indicate with k the equivalence class which contains all sets with cardinality k.

Example 14 *Take the universe $U = \{1, 2, 3\}$ and the set $A = \{\{1\}, \{1, 3\}\}$. Then $glb_C(A) = glb_{\preceq_C}(\{\{\{1\}, \{2\}, \{3\}\}, \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}\}) = \{\{1\}, \{2\}, \{3\}\}$ which we will refer to as $glb_C(A) = 3$ as a shorthand. Similarly $lub_C(A) = lub_{\preceq_C}(\{\{\{1\}, \{2\}, \{3\}\}, \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}\}) = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\} = 2$.*

We are now ready to define BC for a representation R. We denote by $c[S_i]_R$ the set $\{s_i \mid c(s_1, \dots, s_n) \wedge s_j \in D_R(S_j) \forall j \in 1..n\}$.

Definition 6 (Bound consistency on R) *Given a representation R, variables $S_1, S_2, \dots, S_n$, and a constraint $c(S_1 \dots, S_n)$, S_i is bound consistent on c for R iff $lb_R(S_i) = glb_R(c[S_i]_R)$ and $ub_R(S_i) = lub_R(c[S_i]_R)$, where glb_R and lub_R are the greatest lower bound and the least upper bound of the ordering induced by R.*

The constraint c is BC iff all the variables S_i in the constraint are BC.

Note that the above definition is consistent with respect to the usual BC definition provided in Chapter 2. Moreover it naturally extends to partially ordered domains.

4.2 Combining two representations

One way to have a tighter representation for a set variable is to combine two representations by maintaining upper and lower bounds simultaneously on two different orderings. By using two representations simultaneously, a desirable property is that both representations agree on their bounds. We call this property *synchronization* of the bounds. We consider two different ways to propagate a combined representation. In the *weak* sense, we apply bound consistency on each representation independently and we maintain synchronization between their bounds. In the *strong* sense, we ensure that the lower and upper bounds of a representation are bound consistent and at the same time consistent with each other. Given two representations R_1 and R_2, we will denote by $D_{R_1 R_2}(S)$ the set $D_{R_1}(S) \cap D_{R_2}(S)$.

4.2.1 Synchronization of bounds

We say that a representation R_1 is synchronized with a representation R_2 on a set variable S iff $lb_{R_1}(S)$ and $ub_{R_1}(S)$ are consistent with $D_{R_1 R_2}(S)$. We say that S is synchronized for R_1 and R_2 iff R_1 and R_2 are synchronized with each other.

Definition 7 (Synchronization) *Given a variable S, and two representations R_1 and R_2, the representation R_1 is synchronized with the representation R_2 on S iff:*

- $lb_{R_1}(S) = glb_{R_1}(D_{R_1 \cdot R_2}(S))$*;*
- $ub_{R_1}(S) = lub_{R_1}(D_{R_1 \cdot R_2}(S))$*;*

S is synchronized for R_1 and R_2 iff R_1 is synchronized with R_2 and R_2 is synchronized with R_1.

4.2.2 Bound consistency on combinations of two representations

Given two representations R_1 and R_2, the weak combination will be denoted by $R_1 + R_2$ and the strong combination by $R_1 \cdot R_2$.

Definition 8 (Bound consistency on $R_1 + R_2$) *Given variables $S_1, \dots, S_n$, and a constraint $c(S_1 \dots, S_n)$, S_i is bound consistent on c for $R_1 + R_2$ iff S_i is synchronized for R_1 and R_2, S_i is BC on c for R_1 and S_i is BC on c for R_2.*

Definition 9 (Bound consistency on $R_1 \cdot R_2$) *Given variables $S_1, \dots, S_n$ and a constraint $c(S_1 \dots, S_n)$, S_i is bound consistent on c from R_1 to R_2 iff:*

- $lb_{R_1}(S) = glb_{R_1}(c[S_i]_{R_1 \cdot R_2})$*;*

- $ub_{R_1}(S) = lub_{R_1}(c[S_i]_{R_1 \cdot R_2})$;

S_i is bound consistent on c for $R_1 \cdot R_2$ iff S_i is synchronized for R_1 and R_2 and S_i is bound consistent from R_1 to R_2 and vice versa.

We note that $SB{+}card$ representation is used in the Cardinal set solver [1]. The representation $SB \cdot card$ has been previously referred to as $subset-cardinality$ [14] or sbc−domain [28]. A similar BC definition for $SB \cdot card$ is given in [28].

4.3 Comparison of mutual combinations

To be able to compare two representations, we need to compare their ability to remove values from the original domains of some set variables when bound consistency is applied to their representation of these domains.

Definition 10 (Effective Propagation) *Given a constraint $c(S_1, \ldots, S_n)$, a representation R, the original domain $D(S_i)$ of S_i, $D_R(S_i)$ the tightest approximation of $D(S_i)$ using R, and $D'_R(S_i)$ the domain of S_i after the BC propagation of c, the* effective propagation *of c on S_i is the set $p_R\langle c\rangle(S_i) = D(S_i) \cap D'_R(S_i)$ of the values from $D(S_i)$ remaining in the approximated domain $D'_R(S_i)$ after BC has been enforced on c.*

The effective propagation *of c on S_i for a combination $H = R_1 + R_2$ or $H = R_1 \cdot R_2$ is the set $p_H\langle c\rangle(S_i) = D(S_i) \cap D'_{R_1}(S_i) \cap D'_{R_2}(S_i)$.*

Example 15 *Consider the constraint $S <_{lex} \{2, 3\}$, where S is a set variable which takes values from $U = \{1, 2, 3\}$. Assume that $D(S) = \{\{1\}, \{2\}, \{3\}\}$. Consider the tightest minlex approximation of the original domain, that is $D_{ML}(S) = \{\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 3\}, \{2\}, \{2, 3\}, \{3\}\}$. If we propagate the constraint to enforce BC, then the minlex domain reduces to $D'_{ML}(S) = \{\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 3\}, \{2\}\}$. Thus, $p_{ML}\langle c\rangle(S) = \{\{1\}, \{2\}\}$.*

To compare representations, we define the *stronger* relation.

Definition 11 (Stronger relation $\succeq$) *Given two representations R_1 and R_2, a constraint $c(S_1, \ldots, S_n)$ we say that R_1 is stronger than R_2 on c ($R_1 \succeq_c R_2$) iff $\forall D(S_i)$ with $D_{R_1}(S_i)$ and $D_{R_2}(S_i)$ the tightest approximation of $D(S_i)$ for R_1 and R_2, if there exists S_i such that $p_{R_2}\langle c\rangle(S_i) = \emptyset$ then $p_{R_1}\langle c\rangle(S_i) = \emptyset$.*

We say that $R_1 \succeq R_2$ iff $R_1 \succeq_c R_2$ for all constraints c. We say that R_1 is strictly stronger than R_2 on c ($R_1 \succ_c R_2$) iff $R_1 \succeq_c R_2$ and $R_2 \not\succeq_c R_1$.

We say that R_1 is strictly stronger than R_2 ($R_1 \succ R_2$) iff $R_1 \succeq R_2$ and $R_2 \not\succeq R_1$. We say that R_1 and R_2 are incomparable ($R_1 \sim R_2$) iff $R_1 \not\succeq R_2$ and $R_2 \not\succeq R_1$.

The stronger relation naturally extends to combinations $R_1 + R_2$ and $R_1 \cdot R_2$, because their p is well defined. Informally, this notion describes the ability of a BC propagator to remove values from the original set domains. This highly depends on how the domains are approximated and on the properties of the representations.

The following theorems are useful to compare combinations of representations.

Theorem 1 *Given two representations R_1 and R_2, $R_1 + R_2 \succeq R_1$ and $R_1 + R_2 \succeq R_2$.*

Proof: $\forall c(S_1, \dots, S_n)$ and $\forall S_i$, we have that $p_{R_1+R_2}\langle c\rangle(S_i) \subseteq p_{R_1}\langle c\rangle(S_i) \cap p_{R_2}\langle c\rangle(R_i)$ that is a subset of $p_{R_1}\langle c\rangle(S_i)$ and $p_{R_2}\langle c\rangle(S_i)$. As a result, if $p_{R_1}\langle c\rangle(S_i)$ or $p_{R_2}\langle c\rangle(S_i)$ is empty, $p_{R_1+R_2}\langle c\rangle(S_i)$ is empty as well. $\square$

Theorem 2 *Given two representations R_1 and R_2, $R_1 \cdot R_2 \succeq R_1 + R_2$.*

Proof: Let $c(S_1, \dots, S_n)$ be a constraint. Consider a set t from $D(S_i)$ that $p_{R_1+R_2}\langle c\rangle(S_i)$ has pruned. This means that t was removed either by BC on c for R_1, BC on c for R_2, or by synchronization between R_1 and R_2. Consider now BC on c for $R_1 \cdot R_2$. The two items of Definition 9 guarantee at least BC on c for R_1 and BC on c for R_2. In addition, synchronization between R_1 and R_2 is also ensured. Thus t is pruned by $p_{R_1 \cdot R_2}\langle c\rangle(S_i)$. As a result, if $p_{R_1+R_2}\langle c\rangle(S_i)$ is empty, $p_{R_1 \cdot R_2}\langle c\rangle(S_i)$ is empty as well. $\square$

4.4 Using the framework

We are now ready to launch a systematic study of combinations of certain set representations. In particular we will study various mutual combinations of *minlex*, *SB* and *card* representations. *minlex* is effective for lexicographic constraints, used to break symmetries for many CSPs, whereas *SB* and *card* are the two most popular representations. Before using the framework we will instantiate the synchronization property given in Definition 7 for synchronization between *minlex* and *card*, *minlex* and *SB* and *SB* and *card*. This instantiation is useful in order to actively use the framework and follow the theoretical study.

4.4.1 Instantiation of synchronization

In order to instantiate the synchronization property of Definition 7 for synchronization between *minlex* and *card*, *minlex* and *SB* and *SB* and *card*, we provide 5 theorems. In the following, we will reuse the concepts and notation introduced for the *card* ordering in Section 4.1.

Theorem 3 *Given a set variable S, minlex is synchronized with card on S if and only if $lb_C(S) \leq |lb_{ML}(S)| \leq ub_C(S)$ and $lb_C(S) \leq |ub_{ML}(S)| \leq ub_C(S)$.*

Proof: $\Longrightarrow$ If $lb_{ML}(S) = glb_{ML}(D_{ML \cdot C}(S))$ then $lb_{ML}(S) \in D_{ML \cdot C}(S)$ because *minlex* totally orders its sets. Thus $lb_C(S) \leq |lb_{ML}(S)| \leq ub_C(S)$, since $lb_{ML}(S)$ must also belong to the *card* domain. Similarly for $ub_{ML}(S)$.
$\Longleftarrow$ Suppose that $lb_C(S) \leq |lb_{ML}(S)| \leq ub_C(S)$. Then $lb_{ML}(S) \in D_{ML \cdot C}(S)$ and it is the greatest lower bound of the intersection because it is the smallest set of the *minlex* domain. Similarly for $ub_{ML}(S)$.
□

Theorem 4 *Given a set variable S, card is synchronized with minlex on S if and only if $\exists s_l, s_u \in D_{ML}(S)$ such that $|s_l| = lb_C(S)$, $|s_u| = ub_C(S)$.*

Proof: $\Longrightarrow$ By Definition 7, $lb_C(S) = glb_C(D_{ML \cdot C}(S)) = glb_{\preceq_C}(\{[s] \mid s \in D_{ML \cdot C}(S)\})$. Since $\preceq_C$ is a total ordering, it means that there exists at least a set $s_l \in D_{ML}(S)$ whose cardinality is $lb_C(S)$. Similarly for $ub_C(S)$.
$\Longleftarrow$ Suppose that $\exists s_l, s_u \in D_{ML}(S)$ such that $|s_l| = lb_C(S)$ and $|s_u| = ub_C(S)$. It means that s_l, $s_u \in D_{ML \cdot C}(S)$. Since $\preceq_C$ is a total order, $[s_l]$ and $[s_u]$ are the smallest (resp. greatest) equivalence class of the sets contained in $D_{ML \cdot C}(S)$ with respect to $\preceq_C$ ordering. Thus $[s_l] = glb_{\preceq_C}(\{[s] \mid s \in D_{ML \cdot C}(S)\})$ and $[s_u] = lub_{\preceq_C}(\{[s] \mid s \in D_{ML \cdot C}(S)\})$. □

Theorem 5 *Given a set variable S, minlex is synchronized with SB on S if and only if $lb_{ML}(S) \in D_{SB}(S)$ and $ub_{ML}(S) \in D_{SB}(S)$.*

Proof: $\Longrightarrow$ If $lb_{ML}(S) = glb_{ML}(D_{ML \cdot SB}(S))$ then $lb_{ML}(S) \in D_{ML \cdot SB}(S)$ as *minlex* totally orders its sets. Thus $lb_{ML}(S) \in D_{SB}(S)$. Similarly for $ub_{ML}(S)$.
$\Longleftarrow$ Suppose that $lb_{ML}(S) \in D_{SB}(S)$. Then $lb_{ML}(S) \in D_{ML \cdot SB}(S)$. Moreover it is the greatest lower bound of the intersection because it is the smallest set of the *minlex* domain. Similarly for $ub_{ML}(S)$. □

Theorem 6 *Given a set variable S, SB is synchronized with minlex on S if and only if $lb_{SB}(S) \supseteq \cap_{s \in D_{ML}(S)} s$ and $ub_{SB}(S) \subseteq \cup_{s \in D_{ML}(S)} s$.*

Proof: $\Longrightarrow$ By Definition 7, $lb_{SB}(S) = glb_{SB}(D_{ML \cdot SB}(S)) = \cap_{s \in D_{ML \cdot SB}(S)} s \supseteq \cap_{s \in D_{ML}(S)} s$ and $ub_{SB}(S) = lub_{SB}(D_{ML \cdot SB}(S)) = \cup_{s \in D_{ML \cdot SB}(S)} s \subseteq \cup_{s \in D_{ML}(S)} s$.
$\Longleftarrow$ Suppose that $lb_{SB}(S) \supseteq \cap_{s \in D_{ML}(S)} s$ and $ub_{SB}(S) \subseteq \cup_{s \in D_{ML}(S)} s$. We have that $\cap_{s \in D_{ML}(S)} s \subseteq lb_{SB}(S) \subseteq ub_{SB}(S) \subseteq \cup_{s \in D_{ML}(S)} s$. Thus $lb_{SB}(S)$ is the greatest lower bound of $D_{ML \cdot SB}(S)$. Indeed it contains at least all the values commons to each set of the *minlex* domain and it does not contain values that does not appear in the *minlex* domain. Moreover it is the greatest lower bound of the SB domain by construction. Similarly for $ub_{SB}(S)$. $\square$

Theorem 7 *Given a set variable S, card is synchronized with SB on S if and only if $\exists s_l, s_u \in D_{SB}(S)$ such that $|s_l| = lb_C(S)$, $|s_u| = ub_C(S)$.*

Proof: $\Longrightarrow$ By Definition 7, $lb_C(S) = glb_C(D_{SB \cdot C}(S)) = glb_{\preceq_C}(\{[s] \mid s \in D_{SB \cdot C}(S)\})$. Since $\preceq_C$ is a total ordering, it means that there exists at least a set $s_l \in D_{SB}(S)$ whose cardinality is $lb_C(S)$. Similarly for $ub_C(S)$.
$\Longleftarrow$ Suppose $\exists s_l, s_u \in D_{SB}(S)$ such that $|s_l| = lb_C(S)$ and $|s_u| = ub_C(S)$. It means that s_l, $s_u \in D_{SB \cdot C}(S)$. Since $\preceq_C$ is a total order, $[s_l]$ and $[s_u]$ are the smallest (resp. greatest) equivalence class of the sets contained in $D_{SB \cdot C}(S)$ with respect to $\preceq_C$ ordering. Thus $[s_l] = glb_{\preceq_C}(\{[s] \mid s \in D_{SB \cdot C}(S)\})$ and $[s_u] = lub_{\preceq_C}(\{[s] \mid s \in D_{SB \cdot C}(S)\})$. $\square$

Based on the previous theorems, we are able to provide the following instantiations.

Representations *minlex* **and** *card* The synchronization property of Definition 7 impose that given a variable S and the two representations *minlex* and *card*, S is synchronized for *minlex* and *card* iff *minlex* is synchronized with *card* on S and vice versa. Thus by Theorems 3 and 4, we have:

- $lb_C(S) \leq |lb_{ML}(S)| \leq ub_C(S)$, $lb_C(S) \leq |ub_{ML}(S)| \leq ub_C(S)$;
- $\exists s_l, s_u \in D_{ML}(S)$ such that $|s_l| = lb_C(S)$, $|s_u| = ub_C(S)$.

Representations *minlex* **and** *SB* The synchronization property of Definition 7 impose that given a variable S and the two representations *minlex* and SB, S is synchronized for *minlex* and SB iff *minlex* is synchronized with SB on S and vice versa. Thus by Theorems 5 and 6 we have:

- $lb_{ML}(S) \in D_{SB}(S)$, $ub_{ML}(S) \in D_{SB}(S)$;
- $lb_{SB}(S) \supseteq \cap_{s \in D_{ML}(S)} s$ and $ub_{SB}(S) \subseteq \cup_{s \in D_{ML}(S)} s$.

Representations SB and $card$ The synchronization property of Definition 7 impose that given a variable S and the two representations SB and $card$, S is synchronized for SB and $card$ iff SB is synchronized with $card$ on S and vice versa. Thus by Theorem 7 and by definition of glb and lub for the subset ordering, we have:

- $lb_{SB}(S) = glb_{SB}(D_{SB \cdot C}(S)) = \cap_{s \in D_{SB \cdot C}(S)} s$;
- $ub_{SB}(S) = lub_{SB}(D_{SB \cdot C}(S)) = \cup_{s \in D_{SB \cdot C}(S)} s$;
- $\exists s_l, s_u \in D_{SB}(S)$ such that $|s_l| = lb_C(S)$ and $|s_u| = ub_C(S)$.

4.4.2 Comparing pairwise the basic representations

In this section, we perform a pairwise comparison on the basic representations $minlex$, SB and $card$. Our results are summarized in Figure 4.1. A similar study can be done using $maxlex$ instead of $minlex$.

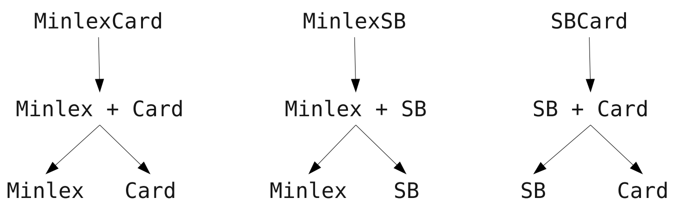

Figure 4.1: Pairwise comparison of representations $minlex, SB$ and $card$. $R_1 \rightarrow R_2$ means $R_1 \succ R_2$.

We start by comparing $minlex$ and $card$ combinations.

Theorem 8 (Comparing $minlex$ and $card$ combinations)

1. $minlex \cdot card \succ minlex + card$;
2. $minlex + card \succ minlex$;
3. $minlex + card \succ card$.

Proof: By Theorems 1 and 2, $minlex \cdot card \succeq minlex + card$, $minlex + card \succeq minlex$ and $minlex + card \succeq card$.

To show that $minlex \cdot card \succ minlex + card$, take the constraint $S_1 \cup S_2 \supseteq \{1, 3\}$, and the universe $U = \{1, 2, 3\}$. Assume that $D(S_1) = D(S_2) = \{\{1\}, \{2\}\}$. Then $minlex$ and $card$ domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1\}, \{1,2\}, \{1,2,3\}, \{1,3\}, \{2\}\}$;
- $lb_C(S_1) = ub_C(S_1) = lb_C(S_2) = ub_C(S_2) = 1$.

BC on $minlex \cdot card$ prunes all the sets from all the domains whilst BC on $minlex + card$ does not prune.

To show that $minlex + card \succ minlex$, take the constraint $S_1 \cup S_2 \supseteq \{1,2,3\}$, and the universe $U = \{1,2,3\}$. Assume that $D(S_1) = D(S_2) = \{\{1\}, \{2\}\}$. Then *minlex* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1\}, \{1,2\}, \{1,2,3\}, \{1,3\}, \{2\}\}$;
- $lb_C(S_1) = ub_C(S_1) = lb_C(S_2) = ub_C(S_2) = 1$.

BC on $minlex + card$ prunes all the sets from all the domains whilst BC on *minlex* does not prune.

Finally, to show that $minlex+card \succ card$, take the constraint $S \geq_{lex} \{2\}$ and the universe $U = \{1,2,3\}$. Assume that $D(S) = \{\{1\}, \{1,2\}\}$. Then *minlex* and *card* domains are the following:

- $D_{ML}(S) = \{\{1\}, \{1,2\}\}$;
- $lb_C(S) = 1$ and $ub_C(S) = 2$.

BC on $minlex + card$ prunes all the sets from all the domains since the constraint has no solution in the *minlex* domain. However BC on *card* alone does not prune. □

Similar results can be obtained also for *minlex* and *SB* combinations.

Theorem 9 (Comparing *minlex* and *SB* combinations)

1. *$minlex \cdot SB \succ minlex + SB$;*
2. *$minlex + SB \succ minlex$;*
3. *$minlex + SB \succ SB$.*

Proof: By Theorems 1 and 2, $minlex \cdot SB \succeq minlex + SB$, $minlex + SB \succeq minlex$ and $minlex + SB \succeq SB$.

To show that $minlex \cdot SB \succ minlex + SB$, take the constraint $|S1| + |S2| = 5$, and the universe $U = \{1,2,3,4,5\}$. Assume that $D(S_1) = D(S_2) = \{\{1,5\}, \{3\}\}$. The minlex and SB domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,5\}, \{2\}, \{2,3\}, \{2,3,4\},$
  $\{2,3,4,5\}, \{2,3,5\}, \{2,4\}, \{2,4,5\}, \{2,5\}, \{3\}\}$;

- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 3, 5\}$.

BC on $minlex \cdot SB$ prunes all the sets since the constraint has no solution in the joint domain whilst BC on $minlex + SB$ does not prune.

To show that $minlex + SB \succ minlex$, take the constraint $S_1 \oplus S_2$ and the universe $U = \{1, 2, 3\}$. Assume that $D(S_1) = D(S_2) = \{\{1, 3\}, \{3\}\}$. Then *minlex* and *SB* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1, 3\}, \{2\}, \{2, 3\}, \{3\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{3\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 3\}$.

BC on $minlex + SB$ clearly prunes all the sets from all the domains whilst BC on *minlex* alone does not prune.

Finally, to show that $minlex + SB \succ SB$, take the constraint $S \geq_{lex} \{1, 4\}$ and the universe $U = \{1, 2, 3, 4\}$. Assume that $D(S) = \{\{\}, \{1\}, \{1, 2\}, \{1, 2, 4\}\}$. Then *minlex* and *SB* domains are the following:

- $D_{ML}(S) = \{\{\}, \{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}, \{1, 2, 4\}\}$;
- $lb_{SB}(S) = \{\}$ and $ub_{SB}(S) = \{1, 2, 4\}$.

BC on $minlex + SB$ prunes all the sets from all the domains since the constraint has no solution in the *minlex* domain. However BC on *SB* alone does not prune. □

As expected, we can obtain similar results also for *SB* and *card*.

Theorem 10 (Comparing *SB* and *card* combinations)

1. $SB \cdot card \succ SB + card$;
2. $SB + card \succ card$;
3. $SB + card \succ SB$.

Proof: By Theorems 1 and 2, $SB \cdot card \succeq SB + card$, $SB + card \succeq SB$ and $SB + card \succeq card$.

To show that $SB \cdot card \succ SB + card$, take the constraint $S_1 \cup S_2 \supseteq \{2, 3, 4\}$ and the universe $U = \{1, 2, 3, 4\}$. Assume that $D(S_1) = D(S_2) = \{\{1, 2\}, \{1, 3\}, \{1, 4\}\}$. The *SB* and *card* domains are the following:

- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{1\}$ and $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 2, 3, 4\}$;
- $lb_C(S_1) = ub_C(S_1) = lb_C(S_2) = ub_C(S_2) = 2$.

BC on $SB \cdot card$ prunes all the sets from all the domain whereas BC on $SB + card$ does not prune.

To show that $SB + card \succ card$, take the constraint $S \supseteq \{2\}$ and the universe $U = \{1, 2, 3\}$. Assume that $D(S) = \{\{1\}, \{1, 3\}\}$. The SB and $card$ domains are:

- $lb_{SB}(S) = \{1\}$, $ub_{SB}(S) = \{1, 3\}$;
- $lb_C(S) = 1$, $ub_C(S) = 2$.

BC on $SB + card$ prunes all the domains of S whereas BC on $card$ alone does not prune.

To show that $SB + card \succ SB$, take the constraint $S_1 \cup S_2 \supseteq \{1, 2, 3\}$ and the universe $U = \{1, 2, 3\}$. Assume that $D(S_1) = D(S_2) = \{\{1\}, \{2\}, \{3\}\}$. Then SB and $card$ domains are the following:

- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$ and $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 2, 3\}$;
- $lb_C(S_1) = ub_C(S_1) = lb_C(S_2) = ub_C(S_2) = 1$.

BC on $SB + card$ prunes all the sets from all the domains whereas BC on SB alone does not prune. $\square$

4.4.3 *lengthlex* in our framework

Given that both *lengthlex* and combinations of *minlex* and *card* representations focus on the same information, we might ask how it positions in our framework on certain constraints like cardinality, lexicographic ordering, or combinations of these two constraints. Surprisingly, the *lengthlex* representation does not appear to be a better way to deal with such constraints. In fact although they focus on the same information, original domains are differently approximated and constraints are propagated in different ways.

Cardinality Constraints

A cardinality constraint is the unary constraint $b \leq |S| \leq d$.

Theorem 11 *Given a variable S and a cardinality constraint $b \leq |S| \leq d$, lengthlex $\succeq_{(b \leq |S| \leq d)}$ minlex.*

Proof: We observe that *lengthlex* totally orders sets giving priority to cardinalities. Hence sets are monotonically increasing ordered with respect to cardinality. Thus, for each set $s \in D_{LL}(S)$ we have $|lb_{LL}(S)| \leq |s| \leq$

$|ub_{LL}(S)|$. Then we are sure that if $b \leq |S| \leq d$ holds for the bounds then it holds also for all the sets in between.

Let t be a set from $D(S)$ pruned by BC on *minlex*, that is, $t \in D(S) \setminus p_{ML}\langle b \leq |S| \leq d\rangle(S)$. t necessarily violates the constraint $b \leq |S| \leq d$. Thus, t will be pruned when enforcing BC on $D_{LL}(S)$. Thus $p_{LL}\langle b \leq |S| \leq d\rangle(S) \subseteq p_{ML}\langle b \leq |S| \leq d\rangle(S)$. $\square$

Note that Theorem 11 also applies to any representation derived from *minlex*. Thus, given a variable S, we derive that BC for *lengthlex* on $b \leq |S| \leq d$ is stronger than BC for *minlex* + *card* and *minlex* · *card*.

Theorem 12 *Given a variable S and a cardinality constraint $b \leq |S| \leq d$, lengthlex $\succ_{(b\leq|S|\leq d)}$ minlex.*

Proof: Take the constraint $1 \leq |S| \leq 2$ and the universe $U = \{1, 2, 3, 4\}$. Assume that $D(S) = \{\{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}$. Then the *minlex* and *lengthlex* domains are the following:

- $D_{ML}(S) = \{\{1, 2, 3, 4\}, \{1, 2, 4\}, \{1, 3\}, \{1, 3, 4\}, \{1, 4\}, \{2\}, \{2, 3\}, \{2, 3, 4\}\}$;
- $D_{LL}(S) = \{\{1, 3, 4\}, \{2, 3, 4\}, \{1, 2, 3, 4\}\}$.

BC on *lengthlex* fails whilst with BC on *minlex* we have $D'_{ML}(S) = \{\{1, 3\}, \{1, 3, 4\}, \{1, 4\}, \{2\}, \{2, 3\}, \}$, so $p_{ML}\langle c\rangle(S) = \{\{1, 3, 4\}\} \neq \emptyset$. Thus *minlex* $\not\succeq_{(b\leq|S|\leq d)}$ *lengthlex*. $\square$

On the other hand, *lengthlex* is equally strong as *minlex* · *card* and *minlex* + *card* on a cardinality constraint.

Theorem 13 *Given a variable S and a cardinality constraint $b \leq |S| \leq d$, minlex · card $\succeq_{(b\leq|S|\leq d)}$ lengthlex and minlex + card $\succeq_{(b\leq|S|\leq d)}$ lengthlex.*

Proof: Enforcing BC on $b \leq |S| \leq d$ for *card* obviously prunes all violating tuples from $D_C(S)$. Thus $p_C\langle b \leq |S| \leq d\rangle(S) = p_{LL}\langle b \leq |S| \leq d\rangle(S)$. This is enough to conclude that $p_{ML+C}\langle b \leq |S| \leq d\rangle(S) \subseteq p_{LL}\langle b \leq |S| \leq d\rangle(S)$ and $p_{ML\cdot C}\langle b \leq |S| \leq d\rangle(S) \subseteq p_{LL}\langle b \leq |S| \leq d\rangle(S)$. $\square$

Since *lengthlex* $\succeq_{(b\leq|S|\leq d)}$ *minlex* · *card* and *minlex* · *card* $\succeq_{(b\leq|S|\leq d)}$ *lengthlex*, we conclude that *lengthlex* and *minlex* · *card* are equal strong on unary cardinality constraints. The same holds for *minlex* + *card*. Figure 4.2 summarizes the results.

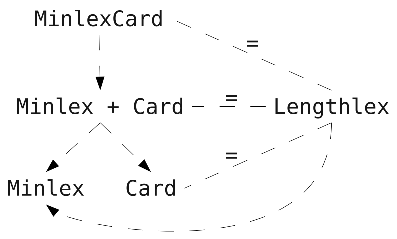

Figure 4.2: Comparison of *minlex* and *card* combinations with *lengthlex* on a cardinality constraint. $R_1 \dashrightarrow R_2$ means $R_1 \succ_{(b \leq |S| \leq d)} R_2$, an $=$ over a dashed line means that R_1 and R_2 are equal strong on a cardinality constraint.

Lexicographic constraints

A lexicographic ordering constraint is a binary constraint $S_1 \leq_{lex} S_2$ which ensures that the sets assigned to S_1 and S_2 are ordered according to $\leq_{ML}$. We will refer to this constraint in short as $\leq_{lex}$.

Theorem 14 *Given two set variables S_1 and S_2, minlex $\succ_{(\leq_{lex})}$ lengthlex.*

Proof: To show that *minlex* $\succeq_{(\leq_{lex})}$ *lengthlex*, we observe that *minlex* orders sets in increasing lexicographic ordering. Thus if S_1 and S_2 are BC for $S_1 \leq_{lex} S_2$, then each set $s \in D_{ML}(S_1)$ is compatible with $ub_{ML}(S_2)$ and each set $t \in D_{ML}(S_2)$ is compatible with $lb_{ML}(S_1)$. Let s be a set from $D(S_1)$ pruned by BC on *lengthlex*, that is, $s \in D(S_1) \setminus p_{LL}\langle S_1 \leq_{lex} S_2\rangle(S_1)$. s necessarily violates the constraint $S_1 \leq_{lex} S_2$. Hence, s will be pruned when enforcing BC on $D_{ML}(S_1)$. Thus, $p_{ML}\langle S_1 \leq_{lex} S_2\rangle(S_1) \subseteq p_{LL}\langle c\rangle(S_1)$. If BC on *lengthlex* prunes a set t from $D(S_2)$ we can apply similar arguments.

To show strictness (*minlex* $\succ_{(\leq_{lex})}$ *lengthlex*), take the constraint $S_1 \leq_{lex} S_2$ and the universe $U = \{1, 2, 3\}$. Assume that $D(S_1) = \{\{2\}, \{2, 3\}\}$ and $D(S_2) = \{\{1\}, \{1, 2\}\}$. Then the *minlex* and *lengthlex* domains are the following:

- $D_{ML}(S_1) = \{\{2\}, \{2, 3\}\}$;
- $D_{ML}(S_2) = \{\{1\}, \{1, 2\}\}$;
- $D_{LL}(S_1) = \{\{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$;
- $D_{LL}(S_2) = \{\{1\}, \{2\}, \{3\}, \{1, 2\}\}$.

BC on *minlex* prunes all the sets from both the domains, whilst with BC on *lengthlex* we have $p_{LL}\langle S_1 \leq_{lex} S_2\rangle(S_1) = D(S_1)$ and $p_{LL}\langle S_1 \leq_{lex} S_2\rangle(S_2) = D(S_2) \setminus \{\{1\}\}$. Thus *lengthlex* $\not\succeq$ *minlex*. $\square$

Since *minlex* + *card* and *minlex* · *card* are stronger than *minlex*, we conclude $minlex{+}card \succ_{(\leq_{lex})} lengthlex$ and $minlex{\cdot}card \succ_{(\leq_{lex})} lengthlex$. Figure 4.3 summarizes the results.

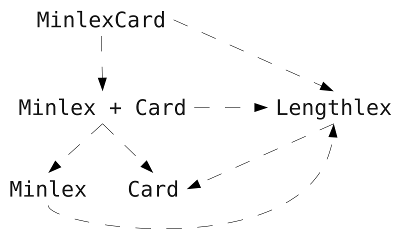

Figure 4.3: Comparison of *minlex* and *card* combinations with *lengthlex* on a lexicographic ordering constraint. $R_1 \dashrightarrow R_2$ means $R_1 \succ_{(\leq_{lex})} R_2$.

Cardinality and lexicographic constraints

We consider a conjunction of a lexicographic constraint $S_1 <_{lex} S_2$ together with cardinality constraints $b_1 \leq |S_1| \leq d_1$ and $b_2 \leq |S_2| \leq d_2$. On such constraints, the *lengthlex* representation has shown promise [26, 28]. This type of constraint is often used to break symmetries in constraint problems with matrix models. We denote such a conjunction by *cclex*.

Theorem 15 *lengthlex is incomparable to minlex, minlex+card and minlex·card on cclex constraints.*

Proof: We first show $lengthlex \not\succeq_{cclex} minlex$. Take the constraint $S_1 <_{lex} S_2 \wedge |S_1| = 2 \wedge |S_2| = 2$ and $U = \{1, 2, 3, 4\}$. Assume that $D(S_1) = \{\{2\}, \{2, 3\}, \{2, 3, 4\}\}$ and $D(S_2) = \{\{1\}, \{1, 2\}, \{1, 3\}, \{1, 2, 3\}\}$. The *minlex* and *lengthlex* domains are the following:

- $D_{ML}(S_1) = \{\{2\}, \{2, 3\}, \{2, 3, 4\}\}$;
- $D_{ML}(S_2) = \{\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}, \{1, 2, 4\}, \{1, 3\}\}$;
- $D_{LL}(S_1) = \{\{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}, \{1, 2, 4\}, \{1, 3, 4\}, \{2, 3, 4\}\}$;
- $D_{LL}(S_2) = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}\}$.

BC on *minlex* fails for both domains whilst BC on *lengthlex* does not fail: $p_{LL}\langle c\rangle(S_1) = D(S_1) \setminus \{\{2\},\{2,3,4\}\}$ and $p_{LL}\langle c\rangle(S_2) = D(S_2) \setminus \{\{1\},\{1,2\},\{1,2,3\}\}$. Thus, *lengthlex* $\not\succeq_{cclex}$ *minlex*, and obviously we deduce *lengthlex* $\not\succeq_{cclex}$ *minlex* + *card* and *lengthlex* $\not\succeq_{cclex}$ *minlex* · *card*.

We now show *minlex* · *card* $\not\succeq_{cclex}$ *lengthlex*. Take the constraint $S_1 <_{lex} S_2 \wedge |S_1| = 2 \wedge |S_2| = 2$ and $U = \{1,2,3,4\}$. Assume that $D(S_1) = \{\{2,3\},\{2,4\},\{3,4\},\{1,2,3\}\}$ and $D(S_2) = \{\{4\},\{1,2\},\{1,3\},\{1,4\}\}$. The *minlex*, *lengthlex* and *card* domains are the following:

- $D_{ML}(S_1) = \{\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},\{1,3,4\},\{1,4\},\{2\},\{2,3\},\{2,3,4\},\{2,4\},\{3\},\{3,4\}\}$;
- $D_{ML}(S_2) = \{\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},\{1,3,4\},\{1,4\},\{2\},\{2,3\},\{2,3,4\},\{2,4\},\{3\},\{3,4\},\{4\}\}$;
- $D_{LL}(S_1) = \{\{2,3\},\{2,4\},\{3,4\},\{1,2,3\}\}$;
- $D_{LL}(S_2) = \{\{4\},\{1,2\},\{1,3\},\{1,4\}\}$;
- $lb_C(S_1) = 2$, $ub_C(S_1) = 3$;
- $lb_C(S_2) = 1$, $ub_C(S_2) = 2$.

With BC on *minlex*·*card* we obtain $p_{ML\cdot C}\langle c\rangle(S_1) = D(S_1)\setminus\{\{1,2,3\},\{3,4\}\}$ and $p_{ML\cdot C}\langle c\rangle(S_2) = D(S_2) \setminus \{\{4\},\{1,2\},\{1,3\}\}$. BC on *lengthlex* fails on both domains. Thus, *minlex*·*card* $\not\succeq_{cclex}$ *lengthlex*, and obviously we deduce *minlex* + *card* $\not\succeq_{cclex}$ *lengthlex* and *minlex* $\not\succeq_{cclex}$ *lengthlex*. $\square$

Figure 4.4 summarizes the results in general (any type of constraint). In fact, there are many other basic constraints, like subset, disjoint, atleast-k, atmost k, where *lengthlex* and the *minlex* and *card* combinations are incomparable.

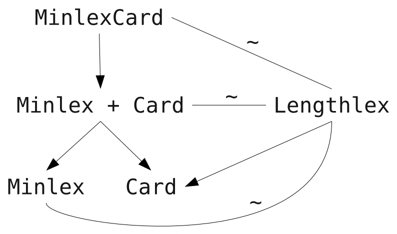

Figure 4.4: Comparison of *minlex* and *card* combinations with *lengthlex*. $R_1 \rightarrow R_2$ means $R_1 \succ R_2$ whilst $R_1 \tilde{-} R_2$ means that $R_1 \sim R_2$.

Other constraints

In the following we further compare *minlex* and *card* with *lengthlex* for other constraints. We have selected some of the constraints for which there exist polynomial propagators for *lengthlex* [22, 11]. We will show that *lengthlex* and *minlex* are still incomparable: the main reason is that the pruning power highly depends on how the representations approximate real domains. This makes *minlex* based combinations appealing for practical purposes.

For each constraints c, we use the following schema: we first show that *lengthlex* is not stronger than *minlex* for BC on c; then we show that *minlex* $\cdot$ *card* is not stronger than *lengthlex* for BC on the same constraint. Since *minlex* $\cdot$ *card* is stronger than *minlex* + *card* that is stronger than *minlex* for BC, using this schema we prove that *lengthlex* is incomparable with all these combinations.

Subset constraint A subset constraint is a binary constraint $S_1 \subseteq S_2$ which ensures that the sets assigned to S_1 are subset of sets assigned to S_2. We denote the subset constraint with $\subseteq$.

Theorem 16 *lengthlex is incomparable to minlex, minlex+card and minlex$\cdot$ card on subset constraints.*

Proof:

We first show that *lengthlex* $\not\succeq_{\subseteq}$ *minlex*. Take the following constraint $S_1 \subseteq S_2$ on two set variables S_1 and S_2 which take values from $U = \{1, 2, 3\}$. Assume that $D(S_1) = \{\{2, 3\}, \{3\}\}$ and $D(S_2) = \{\{1\}, \{1, 2\}\}$. The *minlex* and *lengthlex* domains are the following:

- $D_{ML}(S_1) = \{\{2, 3\}, \{3\}\}$;
- $D_{ML}(S_2) = \{\{1\}, \{1, 2\}\}$;
- $D_{LL}(S_1) = \{\{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}\}$;
- $D_{LL}(S_2) = \{\{1\}, \{2\}, \{3\}, \{1, 2\}\}$.

When the constraint is propagated for *minlex*, it prunes all the sets from all the domains whereas for *lengthlex* we have $p_{LL}\langle c\rangle(S_1) = \{\{3\}\}$ and $p_{LL}\langle c\rangle(S_2) = \{\{1, 2\}\}$. Thus, *lengthlex* $\not\succeq_{\subseteq}$ *minlex* and obviously we deduce that *lengthlex* $\not\succeq_{\subseteq}$ *minlex* + *card* and *lengthlex* $\not\succeq_{\subseteq}$ *minlex* $\cdot$ *card*.

We now show *minlex* $\cdot$ *card* $\not\succeq_{\subseteq}$ *lengthlex*. Take the following constraint $S_1 \subseteq S_2$ on two set variables S_1 and S_2 which take values from $U = \{1, 2, 3, 4\}$. Assume that $D(S_1) = \{\{2, 3\}, \{2, 4\}, \{3, 4\}, \{1, 2, 3\}\}$ and $D(S_2) = \{\{3\}, \{4\}, \{1, 2\}, \{1, 3\}\}$. The *minlex*, *lengthlex* and *card* domains are the following:

- $D_{LL}(S_1) = \{\{2,3\},\{2,4\},\{3,4\},\{1,2,3\}\}$;
- $D_{LL}(S_2) = \{\{3\},\{4\},\{1,2\},\{1,3\}\}$;
- $D_{ML}(S_1) = \{\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},\{1,3,4\},\{1,4\},\{2\},$ $\{2,3\},\{2,3,4\},\{2,4\},\{3\},\{3,4\}\}$;
- $D_{ML}(S_2) = \{\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},\{1,3,4\},\{1,4\},$ $\{2\},\{2,3\},\{2,3,4\},\{2,4\},\{3\},\{3,4\},\{4\}\}$;
- $lb_C(S_1) = 2$, $ub_C(S_1) = 3$;
- $lb_C(S_2) = 1$, $ub_C(S_2) = 2$.

When the constraint is propagated for lengthlex, it fails on both original domains. Instead for *minlex · card*, we have that $p_{ML \cdot C}\langle c\rangle(S_1) = D(S_1) \setminus \{\{1,2,3\}\}$ and $p_{ML \cdot C}\langle c\rangle(S_2) = \{\{1,3\}\}$. Thus, *minlex · card* $\not\succeq_{\subseteq}$ *lengthlex*, and obviously we deduce *minlex + card* $\not\succeq_{\subseteq}$ *lengthlex* and *minlex* $\not\succeq_{\subseteq}$ *lengthlex*. □

Disjoint constraint A disjoint constraint is a binary constraint $S_1 \oplus S_2$ which ensures that $S_1 \cap S_2 = \emptyset$. We denote the disjoint constraint with $\oplus$.

Theorem 17 *lengthlex is incomparable to minlex, minlex+card and minlex· card on disjoint constraints.*

Proof: We first show that *lengthlex* $\not\succeq_{\oplus}$ *minlex*. Take the following constraint $S_1 \oplus S_2$ on two set variables S_1 and S_2 which take values from $U = \{1,2,3\}$. Assume that $D(S_1) = D(S_2) = \{\{1\},\{1,2\},\{1,2,3\}\}$. The *minlex* and *lengthlex* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1\},\{1,2\},\{1,2,3\}\}$;
- $D_{LL}(S_1) = D_{LL}(S_2) = \{\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.

When the constraint is propagated for *minlex*, it prunes all the sets from all the domains whereas for *lengthlex* we have $p_{LL}\langle c\rangle(S_1) = p_{LL}\langle c\rangle(S_2) = \{\{1\},\{1,2\}\}$. Thus, *lengthlex* $\not\succeq_{\oplus}$ *minlex* and obviously we deduce that *lengthlex* $\not\succeq_{\oplus}$ *minlex + card* and *lengthlex* $\not\succeq_{\oplus}$ *minlex · card*.

We now show *minlex · card* $\not\succeq_{\oplus}$ *lengthlex*. Take the following constraint $S_1 \oplus S_2$ on two set variables S_1 and S_2 which take values from $U = \{1,2,3,4\}$. Assume that $D(S_1) = \{\{3\},\{4\},\{1,2\}\}$ and $D(S_2) = \{\{1,3,4\},\{2,3,4\},\{1,2,3,4\}\}$. The *minlex*, *lengthlex* and *card* domains are the following:

- $D_{LL}(S_1) = \{\{3\}, \{4\}\{1,2\}\}$;
- $D_{LL}(S_2) = \{\{1,3,4\}, \{2,3,4\}, \{1,2,3,4\}\}$;
- $D_{ML}(S_1) = \{\{1,2\}, \{1,2,3\}, \{1,2,3,4\}, \{1,2,4\}, \{1,3\}, \{1,3,4\}, \{1,4\}, \{2\}, \{2,3\}, \{2,3,4\}, \{2,4\}, \{3\}, \{3,4\}, \{4\}\}$;
- $D_{ML}(S_2) = \{\{1,2,3,4\}, \{1,2,4\}, \{1,3\}, \{1,3,4\}, \{1,4\}, \{2\}, \{2,3\}, \{2,3,4\}\}$;
- $lb_C(S_1) = 1$, $ub_C(S_1) = 2$;
- $lb_C(S_2) = 3$, $ub_C(S_2) = 4$.

When the constraint is propagated for *lengthlex*, it prunes all the sets from all the domains. Instead for *minlex* $\cdot$ *card* we have that $p_{ML \cdot C}\langle c\rangle(S_1) = \{\{3\}\}$ and $p_{ML \cdot C}\langle c\rangle(S_2) = \{\{1,3,4\}\}$. Thus, *minlex* $\cdot$ *card* $\not\succeq_{\oplus}$ *lengthlex*, and obviously we deduce *minlex* + *card* $\not\succeq_{\oplus}$ *lengthlex* and *minlex* $\not\succeq_{\oplus}$ *lengthlex*. $\square$

atleast-k constraint An atleast-k constraint is a binary constraint which ensures that $S_1 \cap S_2 \geq k$. We denote the atleast-k constraint with *atleast*.

Theorem 18 *lengthlex is incomparable to minlex, minlex+card and minlex$\cdot$card on atleast constraints.*

Proof: We first show that *lengthlex* $\not\succeq_{atleast}$ *minlex*.

Consider the constraint $|S_1 \cap S_2| \geq 2$ on the two set variables S_1 and S_2 which take values from $U = \{1,2,3,4\}$. Assume that $D(S_1) = \{\{1\}, \{1,2\}\}$ and $D(S_2) = \{\{2,3\}, \{2,3,4\}, \{2,4\}, \{3\}\}$. The *minlex* and *lengthlex* domains are the following:

- $D_{ML}(S_1) = \{\{1\}, \{1,2\}\}$;
- $D_{ML}(S_2) = \{\{2,3\}, \{2,3,4\}, \{2,4\}, \{3\}\}$;
- $D_{LL}(S_1) = \{\{1\}, \{2\}, \{3\}, \{4\}, \{1,2\}\}$;
- $D_{LL}(S_2) = \{\{3\}, \{4\}, \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}, \{1,2,3\}, \{1,2,4\}, \{1,3,4\}, \{2,3,4\}\}$.

Enforcing BC for *minlex* prunes all the sets from all the domains. Instead for *lengthlex* we have $p_{LL}\langle c\rangle(S_1) = \{\{1,2\}\}$ and $p_{LL}\langle c\rangle(S_2) = \{\{2,3\}, \{2,4\}\}$. Thus, *lengthlex* $\not\succeq_{atleast}$ *minlex* and obviously we deduce that *lengthlex* $\not\succeq_{atleast}$ *minlex* + *card* and *lengthlex* $\not\succeq_{atleast}$ *minlex* $\cdot$ *card*.

We now show $minlex \cdot card \not\succeq_{atleast} lengthlex$. Take the constraint $|S_1 \cap S_2| \geq 2$ on the two set variables S_1 and S_2 which take values from $U = \{1,2,3,4\}$. Assume that $D(S_1) = \{\{2,4\},\{3,4\}\}$ and $D(S_2) = \{\{4\},\{1,2\},\{1,3\},\{1,4\},\{1,2,3\}\}$. The *minlex*, *lengthlex* and *card* domains are the following:

- $D_{ML}(S_1) = \{\{2,4\},\{3\},\{3,4\}\}$;
- $D_{ML}(S_2) = \{\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},\{1,3,4\},\{1,4\},\{2\},\{2,3\},\{2,3,4\},\{2,4\},\{3\},\{3,4\},\{4\}\}$;
- $D_{LL}(S_1) = \{\{2,4\},\{3,4\}\}$;
- $D_{LL}(S_2) = \{\{3\},\{4\},\{1,2\},\{1,3\},\{1,4\},\{2,3\},\{2,4\},\{3,4\},\{1,2,3\}\}$;
- $lb_C(S_1) = 2$, $ub_C(S_1) = 2$;
- $lb_C(S_2) = 1$, $ub_C(S_2) = 3$;

Propagating the constraint for *lengthlex*, we obtain that $p_{LL}\langle c\rangle(S_2) = \{\}$. However for $minlex \cdot card$ we have $p_{ML \cdot C}\langle c\rangle(S_2) = \{\{1,3\},\{1,4\}\}$. Both representations do not prune with respect to the original domain of S_1. Thus, $minlex \cdot card \not\succeq_{atleast} lengthlex$, and obviously we deduce $minlex + card \not\succeq_{atleast} lengthlex$ and $minlex \not\succeq_{atleast} lengthlex$. $\square$

atmost-k constraint An atmost-k constraint is a binary constraint which ensures that $S_1 \cap S_2 \leq k$. We denote the atmost-k constraint with *atmost*.

Theorem 19 *lengthlex is incomparable to minlex, minlex+card and minlex·card on atmost constraints.*

Proof: We first show that $lengthlex \not\succeq_{atmost} minlex$.

Consider the constraint $|S_1 \cap S_2| \leq 1$ on the two set variables S_1 and S_2 which take values from $U = \{1,2,3\}$. Assume that $D(S_1) = D(S_2) = \{\{1,2\},\{1,2,3\}\}$.The *minlex* and *lengthlex* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,2\},\{1,2,3\}\}$;
- $D_{LL}(S_1) = D_{LL}(S_2) = \{\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$.

Propagating the constraint for the *minlex* domain obviously prune all the sets. Instead for *lengthlex* we have $p_{LL}\langle c\rangle(S_1) = p_{LL}\langle c\rangle(S_2) = \{\{1,2\}\}$. Thus, $lengthlex \not\succeq_{atmost} minlex$ and obviously we deduce that $lengthlex \not\succeq_{atmost} minlex + card$ and $lengthlex \not\succeq_{atmost} minlex \cdot card$.

We now show $minlex \cdot card \not\preceq_{atmost} lengthlex$. Take the constraint $|S_1 \cap S_2| \leq 1$ on the two set variables S_1 and S_2 which take values from $U = \{1, 2, 3\}$. Assume that $D(S_1) = D(S_2) = \{\{2, 3\}, \{1, 2, 3\}\}$. The *minlex*, *lengthlex* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1, 2, 3\}, \{1, 3\}, \{2\}, \{2, 3\}\}$;
- $D_{LL}(S_1) = D_{LL}(S_2) = \{\{2, 3\}, \{1, 2, 3\}\}$;
- $lb_C(S_1) = lb_C(S_2) = 2$, $ub_C(S_1) = ub_C(S_2) = 3$;

Propagating the constraint for *lengthlex* fails for both domains whereas for $minlex \cdot card$ we have $p_{ML \cdot C}\langle c\rangle(S_1) = p_{ML \cdot C}\langle c\rangle(S_2) = \{\{2, 3\}\}$. Thus, $minlex \cdot card \not\preceq_{atmost} lengthlex$, and we deduce $minlex + card \not\preceq_{atmost} lengthlex$ and $minlex \not\preceq_{atmost} lengthlex$. □

Chapter 5

Tractability

In Chapter 4 we deeply studied *minlex* and its combinations with *SB* and *card*. Then we compared *minlex* and *card* combinations with *lengthlex*. This threw out an unexpected result: *lengthlex* is not better than a combination of *minlex* and *card* on cardinality and lexicographic constraints for which *lengthlex* has shown promise [26, 28]. Therefore in this chapter we further investigate *minlex* and we provide some complexity results for constraint propagation. We first show some good properties of *minlex*, focusing on similarities with *lengthlex*. Then we show how to exploit some *lengthlex* algorithms to prove the tractability of i) BC on some common unary and binary constraints for the *minlex* representation ii) synchronization of *minlex* with *SB* and *card*.

5.1 General results

There are a number of negative results about set variables. In this section, we to summarize some of them.

Enforcing BC is in general NP-hard for any representation. We use a reduction from SAT, the boolean satisfiability problem, to BC. Consider the SAT formula ϕ on x_1,..., x_n and its models. Each model I is associated with the set s_I of the indices of the positive literals in I. Consider the unary constraint c that accepts all sets s of integers that correspond to a model. BC on it will fail iff ϕ is unsatisfiable. Thus BC is NP-hard.

Another result is about the fix point behavior. Although filtering algorithms for many elementary constraints are polynomial, the constraint propagation algorithm may take exponential time to converge to a fix point [14].

Bessiere et al. [3] have proved that BC on some commons global con-

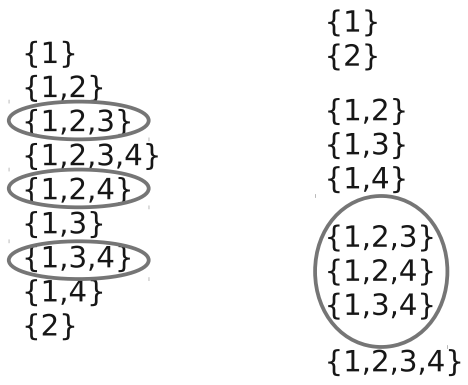

Figure 5.1: Minlex (left) embeds lengthlex (right).

straints such as $atmost1 - incommon$ and *distinct* on sets of fixed cardinalities is NP-hard to propagate. Yip et al. [28] have proved that also BC on a simplification of the $atmost1 - incommon$ constraint is intractable.

There is thus an abundance of negative results on propagating set variables. However, some constraints are tractable in the *minlex* representation, as we will show. Moreover we will also show that synchronization with *minlex* is polynomial.

5.2 *minlex* properties

We recall briefly the definition of the *lengthlex* ordering:

Definition 12 *A lengthlex ordering $<_{LL}$ on sets is defined by:*

$$s <_{LL} t \textit{ iff } s = \{\} \vee |s| < |t| \vee |s| = |t| \wedge (s_1 < t_1 \vee s_1 = t_1 \wedge s \setminus \{s_1\} <_{LL} t \setminus \{t_1\})$$

minlex generalizes this ordering by removing priority on cardinality:

Definition 13 *A minlex ordering $<_{ML}$ on sets is defined by:*

$$s <_{ML} t \textit{ iff } s = \{\} \vee (s_1 < t_1 \vee s_1 = t_1 \wedge s \setminus \{s_1\} <_{ML} t \setminus \{t_1\})$$

Figure 5.1 shows that a *minlex* domain can be seen as the composition of different *lengthlex* domains of fixed cardinality. Moreover we note that the order between sets of the same cardinality is equal in the two representations.

Now we show some properties of the *minlex* ordering which are exploited by our algorithms.

Property 1 *Given a set s which takes values from a universe U, its successor, if it exists, has cardinality $|s| + 1$ or $|s| - 1$.*

Proof: If s is the empty set (cardinality 0), then its successor is $\{1\}$ which has cardinality 1.

Then we distinguish two complementary cases:

- $s_k = n$ (i.e. the last element is the greatest from the universe): according to definition 13, we build the successor removing the last element and then increasing by 1 the new last element. We cannot add to s any other element as otherwise it would become smaller than s;

- $s_k < n$: according to definition 13, we build the successor of s adding $s_k + 1$ to s which is the smallest element that can be added to s.

Therefore we have that the successor of s has cardinality $|s| - 1$ in the first case and $|s| + 1$ in the second. $\square$

For the sake of simplicity, in what follows we consider non-empty sets.

Property 2 (Maximal extension) *Given a set s which takes values from a universe U, the largest set t that contains all the elements of s and $t \geq_{ML} s$ has cardinality $|t| = |s| + n - s_k$ and we call such set t as the "maximal extension" of s.*

Proof: We distinguish two complementary cases:

- $s_k = n$ (i.e. the last element is the greatest from the universe): in this case $t = s$: we can not add to t any other element otherwise it would become smaller than s;

- $s_k < n$: in this case set $t = \{s_1, \ldots, s_k, s_k{+}1, s_k{+}2, \ldots, n\}$. We can not add to t any other element from U otherwise it would become smaller than s.

In both cases the cardinality is $|t| = |s| + n - s_k$. $\square$

Property 3 *Given a set s, for each set t which is equal to s except for the last element s_k where $t_k > s_k$ then the maximal extension of s has cardinality greater than the maximal extension of t.*

Proof: It follows directly from property 2. $\square$

Definition 14 (Extendible) *We say that a set s is extendible iff $s_k < n$, that is if its maximal extension is different from itself. Moreover we say that a set s is extendible to a set of cardinality c, with $c > k$, iff the maximal extension t of s has cardinality $|t| \geq c$. The extension of cardinality c of s is the set $\{s_1, \dots, s_k, s_k + 1, \dots, s_k + c - k\}$.*

Example 16 *Take $U = \{1, 2, 3, 4\}$ and a set $s = \{2, 3\}$. The max extension of s is $\{2, 3, 4\}$ which has cardinality 3. We also say that s is extendible. Take now $t = \{2, 4\}$: its max extension is itself and we say that t is not extendible. Moreover we verify that the cardinality of maximal extension of s is greater than the cardinality of maximal extension of t.*

Take the set $\{1\}$. It is extendible to a set of cardinality 3 which is $\{1, 2, 3\}$. The set $\{2, 3\}$ is not extendible to a set of cardinality 4 because its maximal extension has cardinality 3.

Property 4 *If a set s is extendible to a set t with cardinality c, then t is the first set after s of cardinality c.*

Proof: It follows from definition 14 and Property 2. $\square$

Property 5 *Given a set $s = \{s_1, \dots, s_c, \dots, s_k\}$ and an integer $0 < c < k$, the smallest set t greater than s with cardinality $|t| = c$ is $t = \{s_1, \dots, s_c + 1\}$, if it exists.*

Proof: We exploit the fact that *minlex* embeds *lengthlex*. Consider the set $u = \{s_1, \dots, s_c\}$. We have that u is less than or equal to s, according to the *minlex* ordering, and the successor of u in the *lengthlex* domain of cardinality c is t which comes after s according to definition 13.

Example 17 *Take $s = \{1, 2, 3\}$ from the universe $U = \{1, 2, 3, 4\}$. Then the smallest set of cardinality 2 greater than s is $\{1, 3\}$. Indeed $\{1, 3\}$ is the successor of $\{1, 2\}$ in the embedded lengthlex domain of cardinality 2 and $\{1, 3\}$ comes after $\{1, 2, 3\}$.*

Property 6 *Given a set $s = \{s_1, \dots, s_k\}$, if s is not extendible to a set of cardinality c where $c > k$ then for every set t such that $s \leq_{ML} t <_{ML} \{s_1, \dots, s_{k-1} + 1\}$, t is not extendible to a set of cardinality c.*

Proof: Every set t that is $s \leq_{ML} t <_{ML} \{s_1, \dots, s_{k-1} + 1\}$ must have the common beginning $\{s_1, \dots, s_{k-1}\}$. Moreover we have that t_k must be greater or equal than s_k, otherwise it would be outside the interval. Thus applying Property 3 we have that each set t cannot be extended to a set of cardinality c.

Example 18 *Take for example the set* $s = \{1, 3, 5\}$ *which takes value from the universe* $U = \{1, 2, 3, 4, 5\}$*. Suppose we want extend* s *to a set with cardinality* 4*. The maximal extension of* s *is* s *itself. Thus we apply Property 6 and we test the set* $\{1, 4\}$*. Again, it is not extendible to a set of cardinality* 4*. Thus using Property 6 we safely skip all the sets until* $\{2\}$*, which can be extended to* $\{2, 3, 4, 5\}$*, of cardinality* 4*.*

5.3 Updating *minlex* bounds based on cardinality

In this section we give two functions that update *minlex* bounds given a cardinality constraint of type $|S| = c$. Following the schema proposed in [11], we define the following two functions:

- $first - c_{ML}(s, c, U)$: given a set s which takes value from a universe U, returns the first set t such that $t \geq_{ML} s$ and $|t| = c$;

- $last - c_{ML}(s, c, U)$: given a set s which takes value from a universe U, returns the last set t such that $t \leq_{ML} s$ and $|t| = c$.

For the sake of simplicity we assume that c is positive and s is not the empty set.

5.3.1 $first - c_{ML}(s, c, U)$

We distinguish three cases:

- $|s| = c$: the function returns immediately s;

- $|s| < c$: here we distinguish two more subcases. The first is that s can be extended to a set with cardinality greater than or equal to c. To do this we exploit Property 2 and Definition 14. If this is possible, then we extend s until the desired case, computing the successor of s until cardinality c is reached. Property 4 ensures that this set is the smallest set with cardinality c after s. This step takes at most $O(n)$. If s is not extendible to a set of cardinality c, then we search for the first t that can be extended to a set of cardinality c. To do this in polynomial time, we take advantage of Property 6 and we try only meaningful values. Basically at each step we remove the last element from s, we increase by one the new last element and then we re-execute the extendibility test. This step takes at most $O(n^2)$;

- $|s| > c$: we simply remove from s all the elements which have index greater than c; then we add 1 to the new last element of s (by Property 5).

Algorithm 2 is an auxiliary algorithm which extends a set s to a desired cardinality in $O(n)$. Instead the $first - c_{ML}(s, c, U)$ function is implemented by Algorithm 1. Lines from 1 to 3 implement the first case, lines from 4 to 16 the second and finally lines from 17 to 19 the third. The computational cost is $O(n^2)$, since $extend_{ML}$ is invoked inside the for loop of line 9. Note that the algorithm returns $\{-1\}$ if does not exist a set of cardinality c after s.

Algorithm 1: $first - c_{ML}(s, c, U)$

```
   Data: s: set, c: integer, U: set
   Result: Returns the first set of cardinality c after s, otherwise it
           returns {−1}.
 1 if $|s| = c$ then
 2     return $s$;
 3 end
 4 if $|s| < c$ then
 5     if $|s| + n - s_k \geq c$ then
 6         return $extend_{ML}(s, c, U)$;
 7     end
 8     else
 9         for $i = |s| - 1$ down to 1 do
10             $s = \{s_1, \ldots, s_i + 1\}$;
11             if $|s| + n - s_k \geq c$ then
12                 return $extend_{ML}(s, c, U)$;
13             end
14         end
15     end
16 end
17 if $|s| > c$ then
18     return $\{s_1, \ldots, s_{c-1}, s_c + 1\}$;
19 end
20 return $\{-1\}$;
```

Example 19 *Take* $s = \{1, 2, 4\}$ *and* $U = \{1, 2, 3, 4\}$*. We want to find the first set after* $\{1, 2, 4\}$ *which has cardinality* 4*. Such a set does not exist. We*

Algorithm 2: *extend*$_{ML}(s, c, U)$

```
Data: $s$: set, $c$: integer, $U$: set. Requires that $|s| \leq c$.
Result: Extends $s$ to a set of cardinality $c$.
1 $result = s$;
2 $m = s_k + 1$;
3 for $i = c - |s|$ down to 1 do
4     $result = result \cup \{m\}$;
5     $m = m + 1$;
6 end
7 return $result$;
```

start the algorithm and only the condition of line 4 is satisfied. As s can not be extended to a set of cardinality 4 *(line 5), we enter in the for loop of line 9. Then we use Property 6 to try only meaningful values and we repeat the extendibility test for* $\{1, 3\}$ *and* $\{2\}$*. As they fail, we skip to line 20 and we return* $\{-1\}$*.*

5.3.2 $last - c_{ML}(s, c, U)$

For the sake of simplicity, we can build $last - c_{ML}(s, c, U)$ function on top of the $first - c_{ML}(s, c, U)$ function which has been just defined and the $prec_{LL}(s)$ function, from the *lengthlex* domain. $prec_{LL}(s)$ computes the predecessor of s with the same cardinality as s. If such a predecessor does not exist, we assume that it returns $\{-1\}$. The computational cost of $prec_{LL}$ is $O(n)$.

We have two cases:

- $|s| = c$: we can safely return s;
- $|s| \neq c$: in this case $last - c_{ML}(s, c, U) = prec_{LL}(first - c_{ML}(s, c, U))$. If $first - c_{ML}$ returns $\{-1\}$, then $last - c_{ML}$ can safely return the set $\{n - c + 1, \ldots, n\}$, since it is the biggest set, wrt *minlex* ordering, of cardinality c. If $prec_{LL}$ returns $\{-1\}$ then $last - c_{ML}$ does the same.

Algorithm 3 performs exactly these simple steps. Its computational cost is dominated by the $first - c_{ML}$ function, thus it is $O(n^2)$.

Example 20 *Take the following minlex domain:* $\{\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 3\}, \{2\}, \{2, 3\}, \{3\}\}$*. Suppose we want compute* $last - c_{ML}(\{2\}, 2, \{1, 2, 3\})$*: first we jump to* $first - c_{ML}(\{2\}, 2, \{1, 2, 3\}) = \{2, 3\}$ *and then we go back to* $prec_{LL}(\{2, 3\}) = \{1, 3\}$ *that is the last set before* $\{2\}$ *that contains two elements.*

Algorithm 3: $last - c_{ML}(s, c, U)$

```
Data: $s$: set, $c$: integer, $U$: set
Result: Returns the last set of cardinality $c$ before $s$.
if $|s| = c$ then
    return $s$;
end
if $first - c_{ML}(s, c, U) = \{-1\}$ then
    return $\{n - c + 1, \ldots, n\}$;
end
else
    return $prec_{LL}(first - c_{ML}(s, c, U))$;
end
```

5.4 BC for unary constraints

For the sake of simplicity we assume that in what follows we deal only with positive cardinalities and non-empty sets.

Despite the general intractability, we are able to show that BC on some unary constraints can be enforced in polynomial time for the *minlex* representation. Applying Definition 6, we have that given a variable S and a *minlex* domain, enforcing BC for unary constraints means finding the first value after $lb_{ML}(S)$ for which the constraint holds and the last value before $ub_{ML}(S)$ for which the constraint holds (or the bounds themselves if the constraint is true when applied to them). For example we can use the functions defined in the previous section to enforce BC for unary constraint of type $|S| = c$. In the following sections, we will analyze other common unary constraint.

5.4.1 Inclusion constraint

To enforce BC for the inclusion constraint ($S \supseteq r$) the authors of [11] define two functions:

- $first - r_{LL}(s, r, c)$: computes the first set after s with cardinality c that contains all elements in r (or s itself if it contains r);

- $last - r_{LL}(s, r, c)$: computes the last set before s with cardinality c that contains all elements in r (or s itself if it contains r).

They are both polynomial and $first - r_{LL}$ takes $O(n)$. Moreover we assume that these functions return $\{-1\}$ if does not exist a set which satisfies the conditions.

Thus we want to build similar functions:

- $first - r_{ML}(s, r, U)$: given a set s which takes values from the universe U, returns the first set after s which contains the required elements r, or s itself if it contains r;
- $last - r_{ML}(s, r, U)$: given a set s which takes values from the universe U, returns the last set before s which contains the required elements r, or s itself if it contains r;

We will show how to build $first - r_{ML}(s, r, U)$: $last - r_{ML}(s, r, U)$ can be built in the same fashion. We build $first - r_{ML}(s, r, U)$ on top of the $first - r_{LL}(s, r, c)$. We give the basic intuition with an example.

Given a variable S, we want enforce BC for the constraint $S \supseteq \{3\}$ over the following *minlex* domain: $\{\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 3\}, \{2\}\}$. The embedded *lengthlex* subdomains are:

- the *lengthlex* subdomain of cardinality 1 is $\{\{1\}, \{2\}\}$;
- the *lengthlex* subdomain of cardinality 2 is $\{\{1, 2\}, \{1, 3\}\}$;
- the *lengthlex* subdomain of cardinality 3 is $\{\{1, 2, 3\}\}$.

We can search the first set which satisfies the constraint looking in polynomial time at each *lengthlex* subdomain, taking the minimum w.r.t. *minlex* ordering. We observe that the maximum number of *lengthlex* subdomain is $O(n)$, therefore we remain polynomial. We execute the following steps:

- $first - r_{LL}(first - c_{ML}(\{1\}, 1, \{1, 2, 3\}), \{3\}, 1)$ returns the first set of cardinality 1, after $\{1\}$, which contains the set $\{3\}$. $first - r_{LL}$ would return $\{3\}$, that is discarded because it is greater than the upper bound.
- $first - r_{LL}(first - c_{ML}(\{1\}, 2, \{1, 2, 3\}), \{3\}, 2)$ returns the first set of cardinality 2, after $\{1, 2\}$, which contains the set $\{3\}$. Thus $first - r_{LL}$ returns $\{1, 3\}$, which is a candidate for the solution;
- $first - r_{LL}(first - c_{ML}(\{1\}, 3, \{1, 2, 3\}), \{3\}, 3)$ returns the first set of cardinality 3, after $\{1, 2, 3\}$, which contains the set $\{3\}$. Thus $first - r_{LL}$ returns $\{1, 2, 3\}$, which is another candidate for the solution.

Then we keep the minimum, w.r.t. *minlex* ordering, between $\{1, 3\}$ and $\{1, 2, 3\}$ that is $\{1, 2, 3\}$.

We build the $first - r_{ML}(s, r, U)$ following the given intuition. Note that we do not check if the found element is greater than the upper bound: it will be the BC algorithm which will take care about this.

Algorithm 4: $first - r_{ML}(s, r, U)$

```
   Data: $s$: set, $r$: set, $U$: set
   Result: Returns the first set after $s$ which contains $r$.
1  $result = \{-1\}$;
2  for $i = |r|$ to $n$ do
3      $firstc = first - c_{ML}(s, i, U)$;
4      if $firstc \neq \{-1\}$ then
5          $firstr = first - r_{LL}(firstc, r, i)$;
6          if $firstr \neq \{-1\}$ then
7              if $result = \{-1\}$ then
8                  $result = firstr$;
9              end
10             else
11                 $result = min_{ML}(firstr, result)$;
12             end
13         end
14     end
15 end
16 return $result$;
```

$first - r_{LL}$ is implemented by Algorithm 4. It uses the *lengthlex* function $first - r_{LL}(s, r, c)$, which is polynomial, for all the possible cardinalities (for loop at line 2) and it keeps the minimum value, w.r.t minlex ordering. $first - c_{ML}(s, c, U)$ (line 3) is used to select the lower bound of the embedded *lengthlex* domain of cardinality i. The min_{ML} function used at line 11 simply returns the minimum between two values w.r.t. *minlex* ordering. The algorithm returns $\{-1\}$ if does not exist a set which contains the required elements. Its computational cost is $O(n^3)$ which can be surely improved but it is enough to see that it is tractable.

In Figure 5.2 we can see how $first - r_{ML}$ can be used to update the lower bound of a given *minlex* domain. Figure 5.2 also shows at each step what the algorithm does.

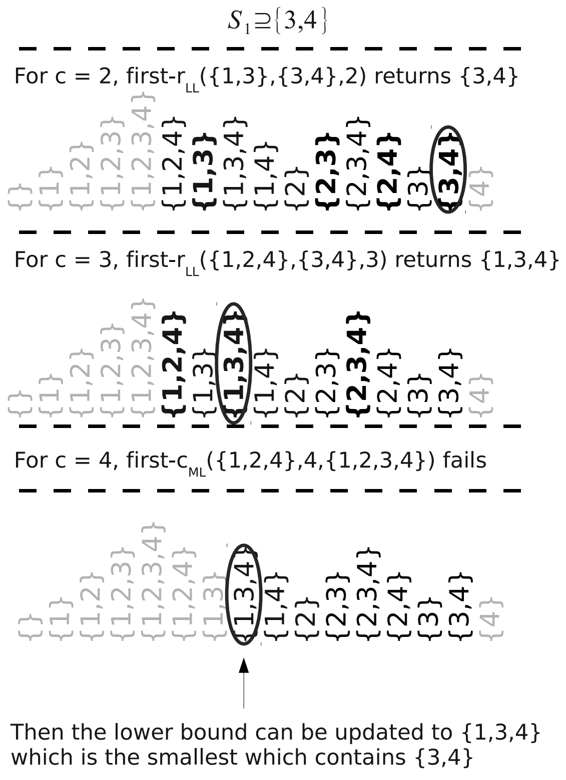

Figure 5.2: Enforcing BC for the unary constraint $S \supseteq \{3,4\}$: updating the lower bound. Sets take value from the universe $U = \{1,2,3,4\}$. The *minlex* domain goes from $\{1,2,4\}$ up to $\{3,4\}$.

5.4.2 Disjoint constraint

To enforce BC for the disjoint constraint ($S \oplus e$) the authors of [11] define two functions:

- $first - e_{LL}(s, e, c)$: computes the first set after s with cardinality c that does not include any element in e (or s itself if it is disjoint from e);
- $last - e_{LL}(s, e, c)$: computes the last set before s with cardinality c that does not include any element in e (or s itself if it is disjoint from e).

Using the same technique of the inclusion constraint, we can construct similar functions:

- $first - e_{ML}(s, e, U)$: given a set s which takes values from a universe U, it returns the first element after s which is disjoint from e, or s itself if it disjoint from e, otherwise it returns $\{-1\}$;
- $last - e_{ML}(s, e, U)$: given a set s which takes values from a universe U, it returns the last element before s which is disjoint from e, or s itself if it disjoint from e, otherwise it returns $\{-1\}$.

5.4.3 Conjunction of unary constraints

We are also able to enforce BC in polynomial time for conjunction of unary constraints. In the following two sections we will show how to build functions for the subset constraint $s_1 \subseteq S \subseteq s_2$ and the cardinality constraint $c \leq |S| \leq d$.

Subset constraint ($s_1 \subseteq S \subseteq s_2$)

To propagate the subset constraint, we need the following two functions:

- $first - s_1s_{2ML}(s, s_1, s_2, U)$: given a set s which takes values from a universe U, it returns the first set after s for which $s_1 \subseteq s \subseteq s_2$ holds, or s itself if it is in the interval, otherwise it returns $\{-1\}$;
- $last - s_1s_{2ML}(s, s_1, s_2, U)$: given a set s which takes values from a universe U, it returns the last set before s for which $s_1 \subseteq s \subseteq s_2$ holds, or s itself if it is in the interval, otherwise it returns $\{-1\}$.

We show only how to build the $first - s_1s_{2ML}(s, s_1, s_2, U)$ function. The $last - s_1s_{2ML}(s, s_1, s_2, U)$ can be built in the same fashion.

Without loss of generality we assume that $s_2 = \{1, 2, \ldots, |s_2|\}$ and $s \subseteq s_2$. Then $first - s_1s_{2ML}(s, s_1, s_2, U)$ is equal to $first - r_{ML}(s, s_1, s_2)$ because

$first - r_{ML}(s, s_1, s_2)$ returns only those sets which contains s_1, according to the definition of $first - r_{ML}$, and which are contained to s_2 because $first - r_{ML}$ runs on the restricted universe s_2. The assumptions are not strong because if s_2 is not in the form $\{1, 2, \dots, |s_2|\}$ we can always use a map function. We can overcome the last assumption using $s' = first - e_{ML}(s, U \setminus s_2, U)$ instead of s. Indeed s' is the first set after s which is contained in s_2.

Example 21 *Take* $s = \{1, 3\}$ *which takes values from* $U = \{1, 2, 3, 4\}$. *We want to compute* $first - {s_1s_2}_{ML}(s, \{2, 3\}, \{1, 2, 3\}, U)$. *Then we execute* $first - r_{ML}(s, \{2, 3\}, \{1, 2, 3\})$ *which returns the set* $\{2, 3\}$.

Cardinality constraint $(c \leq |S| \leq d)$

Enforcing BC for this constraint is also tractable. To do that, we need the following functions:

- $first - cd_{ML}(s, c, d, U)$: given a set s which takes values from the universe U, it returns the first set after s which has cardinality between c and d, or s itself if its cardinality is between c and d otherwise it returns $\{-1\}$;

- $last - cd_{ML}(s, c, d, U)$: given a set s which takes values from the universe U, it returns the last set before s which has cardinality between c and d, or s itself if its cardinality is between c and d otherwise it returns $\{-1\}$.

We show only the first function, the last can be derived in the same fashion. We distinguish three cases:

- $|s| < c$: the smallest set $t >_{ML} s$ (w.r.t. *minlex* ordering) with cardinality $|t| \geq c$ must have cardinality $|t| = c$. Indeed Property 1 ensures that cardinality can be incremented at most by 1 at each step when computing the successor. Thus the first set after s which has cardinality between c and d must have cardinality c. Therefore we can use $first - c_{ML}(s, c, U)$ function;

- $|s| > d$: with analogous arguments of the previous point, we can use $first - c_{ML}(s, d, U)$ to obtain the first set with cardinality d;

- $c \leq |s| \leq d$: the constraint is satisfied by s.

Thus we can safely assume to have these two polynomial functions.

5.5 Synchronization with *minlex*

In this section we analyze the complexity of enforcing synchronization for *minlex* and *card* and *minlex* and SB. We use some of the functions defined in the previous sections. We note that synchronization between two representations can be understood as a domain constraint (e.g. $S \in D_{R_1R_2}(S)$). Therefore the same considerations made in Section 5.4 also apply to synchronization. For the sake of simplicity, we assume that lower and upper bound of the variables are not empty sets.

5.5.1 Synchronization for *minlex* and *card*

We recall the instantiation of synchronization for *minlex* and *card* provided in Section 4.4.1.

Given the two representations *minlex* and *card*, given a variable S, we say that S is synchronized for *minlex* and *card* iff:

- $lb_C(S) \leq |lb_{ML}(S)| \leq ub_C(S)$, $lb_C(S) \leq |ub_{ML}(S)| \leq ub_C(S)$;
- there exist a set s_l and a set s_u between $lb_{ML}(S)$ and $ub_{ML}(S)$ such that $|s_l| = lb_C(S)$ and $|s_u| = ub_C(S)$.

The first part can be enforced using $first{-}cd_{ML}(lb_{ML}(S), lb_C(S), ub_C(S), U)$ and $last - cd_{ML}(lb_{ML}(S), lb_C(S), ub_C(S), U)$, both polynomial.

Given a variable S and the *minlex* representation, let *maxcard* be the cardinality of the set with maximum cardinality which belongs to $D_{ML}(S)$, and *mincard* the cardinality of the set with minimum cardinality which belongs to $D_{ML}(S)$. A consequence of Property 1 is that for each $e \in [mincard, maxcard]$ there exists a set u such that $e = |u|$ and $lb_{ML}(S) \leq_{ML} u \leq_{ML} ub_{ML}(S)$ (otherwise we could not move from a set of cardinality *maxcard* to a set with cardinality *mincard*).

Example 22 *Given a variable S and a universe $U = \{1, 2, 3, 4, 5\}$, let $lb_{ML}(S) = \{1, 4\}$ and $ub_{ML}(S) = \{2, 4\}$. Then $maxcard = 4$ and $mincard = 1$. For $e = 1$ we can take the set $\{2\}$, for $e = 2$ the set $\{1, 4\}$, for $e = 3$ the set $\{1, 4, 5\}$ and for $e = 4$ the set $\{2, 3, 4, 5\}$.*

Thus if $lb_C(S)$ belongs to $[mincard, maxcard]$ then we are sure that there exists a set of card $lb_C(S)$ which belongs to the *minlex* domain. Similarly for $ub_C(S)$.

First we give two algorithms $min{-}card(lb, ub, U)$ and $max{-}card(lb, ub, U)$ which compute the cardinality of the smallest (resp. largest) set in a given *minlex* interval.

Algorithm 5: $min - card_{ML}(lb, ub, U)$

```
  Data: $lb$: set, $ub$: set, $U$: set
  Result: the cardinality of the smallest set in the interval
1 for $i = 1$ to $n$ do
2     $s = first - c_{ML}(lb, i, U)$;
3     if $s \neq \{-1\}$ and $s \leq_{ML} ub$ then
4         return $i$
5     end
6 end
```

Algorithm 5 computes the smallest set in a naive way, but polynomial. Basically, in ascending order of cardinality (for loop of line 1) it checks if there exists a set with that cardinality which belongs to the interval (lines 2 and 3). The computational cost is $O(n^3)$. We can build the $max - card_{ML}$ in the same fashion.

Algorithm 6: $max - card_{ML}(lb, ub, U)$

```
  Data: $lb$: set, $ub$: set, $U$: set
  Result: the cardinality of the largest set in the interval
1 for $i = n$ down to 1 do
2     $s = first - c_{ML}(lb, i, U)$;
3     if $s \neq \{-1\}$ and $s \leq_{ML} ub$ then
4         return $i$
5     end
6 end
```

Algorithm 7 enforces synchronization for *minlex* and *card*. Line 1 and line 2 ensure synchronization for *minlex* with *card*. Lines 6 and 7 ensure synchronization for *card* with *minlex*. If it is impossible to enforce synchronization for the given variable, it fails.

Algorithm 7: $sync_{ML,C}(S, U)$

```
  Data: $S$: set variable, $U$: set
  Result: minlex synchronized with card on $S$ and vice versa
1 $lb_{ML}(S) = first - cd_{ML}(lb_{ML}(S), lb_C(S), ub_C(S), U)$;
2 $ub_{ML}(S) = last - cd_{ML}(ub_{ML}(S), lb_C(S), ub_C(S), U)$;
3 if $lb_{ML}(S) >_{ML} ub_{ML}(S)$ or $lb_{ML}(S) = \{-1\}$ or $ub_{ML}(S) = \{-1\}$
  then
4     fail;
5 end
6 $lb_C(S) = max(lb_C(S), min - card_{ML}(lb_{ML}(S), ub_{ML}(S), U))$;
7 $ub_C(S) = min(ub_C(S), max - card_{ML}(lb_{ML}(S), ub_{ML}(S), U))$;
```

5.5.2 Synchronization for *minlex* and *SB*

We recall the instantiation of synchronization for *minlex* and *SB* given in section 4.4.1. Given the *minlex* representation and the subset bounds representation *SB*, given a variable S we say that S is *synchronized* for *minlex* and *SB* iff:

- $lb_{ML}(S) \in D_{SB}(S)$, $ub_{ML}(S) \in D_{SB}(S)$;
- $ub_{SB}(S) \subseteq \bigcup_{s \in D_{ML}(S)} s$ and $lb_{SB}(S) \supseteq \bigcap_{s \in D_{ML}(S)} s$.

For the first part we can exploit the two functions $first - s_1 s_{2ML}(lb_{ML}(S), lb_{SB}(S), ub_{SB}(S), U)$ and $last - s_1 s_{2ML}(ub_{ML}(S), lb_{SB}(S), ub_{SB}(S), U)$. For the second we define the following two auxiliary functions:

- $union_{ML}(lb, ub, U)$: returns the union of all the sets which belong to the *minlex* interval. It is implemented by Algorithm 8. It uses $first - r_{ML}(s, r, U)$ (line 3) applied to each element $u \in U$ (line 2) in order to compute the union of all the elements which belong to at least one set in the interval (line 4);
- $intersection_{ML}(lb, ub, U)$: returns the intersection of all the sets which belong to the *minlex* interval. $intersection_{ML}$ is implemented by Algorithm 9. It exploits the $first - e_{ML}(s, e, U)$ function. For each value $u \in U$ (line 2), if $first - e_{ML}(lb, u, U)$ returns a set that does not belong to the interval then u can not belong to the intersection (lines 3-5).

In the following there are the algorithms for union and intersection:

Algorithm 8: $union_{ML}(lb, ub, U)$

```
Data: $lb$: set, $ub$: set, $U$: set
Result: returns the union of all the sets which belong to the interval
1 $result = \{\}$;
2 for $\forall u \in U$ do
3     if $first - r_{ML}(lb, u, U) \neq \{-1\}$ and $first - r_{ML}(lb, u, U) \leq ub$
      then
4         $result = result \cup u$;
5     end
6 end
7 return $result$;
```

Algorithm 9: $intersection_{ML}(lb, ub, U)$

```
Data: $lb$: set, $ub$: set, $U$: set
Result: returns the intersection of all the sets which belong to the
        interval
1 $result = U$;
2 for $\forall u \in U$ do
3     if $first - e_{ML}(lb, u, U) \neq \{-1\}$and $first - e_{ML}(lb, u, U) \leq ub$
      then
4         $result = result \setminus \{u\}$;
5     end
6 end
7 return $result$;
```

Both $union_{ML}$ and $intersection_{ML}$ have computational cost that is equal to $O(n^4)$. Synchronization for *minlex* and SB is implemented by Algorithm 10: line 1 and line 2 ensure synchronization for *minlex* with SB. Lines 6 and 7 ensure synchronization for SB with *minlex*. If it is impossible to enforce synchronization for the given variable, it fails. The computational cost is $O(n^4)$.

Algorithm 10: $sync_{ML,SB}(S, U)$

```
Data: $S$: set variable, $U$: set
Result: $minlex$ synchronized with $SB$ on $S$ and vice versa
1 $lb_{ML}(S) = first - s_1s_{2ML}(lb_{ML}(S), lb_{SB}(S), ub_{SB}(S), U)$;
2 $ub_{ML}(S) = last - s_1s_{2ML}(ub_{ML}(S), lb_{SB}(S), ub_{SB}(S), U)$;
3 if $lb_{ML}(S) > ub_{ML}(S)$ or $lb_{ML}(S) = \{-1\}$ or $ub_{ML}(S) = \{-1\}$ then
4     fail;
5 end
6 $lb_{SB}(S) = lb_{SB}(S) \cup intersection_{ML}(lb_{ML}(S), ub_{ML}(S), U)$;
7 $ub_{SB}(S) = ub_{SB}(S) \cap union_{ML}(lb_{ML}(S), ub_{ML}(S), U)$;
```

5.6 Binary constraints

In [22] authors present a generic bounds consistency algorithm for a *lengthlex* domain. Their BC algorithm is expressed in terms of three functions hs_{LL}, $succ_{LL,S_1}$ and $pred_{LL,S_1}$. Given a constraint c, two set variables S_1 and S_2 and the *lengthlex* representation LL, $hs_{LL}\langle c\rangle(S_1, S_2) \equiv \exists x \in S_1, y \in S_2 : c(x, y)$.

hs_{LL} is also called the feasibility routine, since it returns true if and only if the constraint can be satisfied. For a constraint c and two set variables S_1 and S_2 such that $hs_{LL}\langle c\rangle(S_1, S_2)$ is true, function $succ_{LL,S_1}\langle c\rangle(S_1, S_2)$ returns the smallest set $x \in D_{LL}(S_1)$, with respect to LL ordering, which belongs to a solution of the constraint:

$$succ_{LL,S_1}\langle c\rangle(S_1, S_2) \equiv min_{LL}\{x \in D_{LL}(S_1) \mid \exists y \in D_{LL}(S_2) : c(x, y)\}$$

For a constraint c and two set variables S_1 and S_2 such that $hs_{LL}\langle c\rangle(S_1, S_2)$ is true, function $pred_{LL,S_1}\langle c\rangle(S_1, S_2)$ returns the largest set $x \in D_{LL}(S_1)$, with respect to *lengthlex* ordering, which belongs to a solution of the constraint, i.e.,

$$pred_{LL,S_1}\langle c\rangle(S_1, S_2) \equiv max_{LL}\{x \in D_{LL}(S_1) \mid \exists y \in D_{LL}(S_2) : c(x, y)\}$$

The definitions are similar for the S_2 variable. Algorithm 11 enforces BC on a binary constraint for the *lengthlex* representation. Line 2 ensures that $lb_{LL}(S_1)$ is updated to the smallest set of $D_{LL}(S_1)$ (w.r.t. *lengthlex* ordering) which satisfies the constraint. On the other hand, line 3 ensures that $ub_{LL}(S_1)$ is updated to the greatest set of $D_{LL}(S_1)$ (w.r.t. *lengthlex* ordering) which satisfies the constraint. Lines 4 and 5 perform the same operations for the S_2 variable. Algorithm 11 is clearly consistent with our BC definition given in Chapter 4.

Algorithm 11: $bc_{LL}\langle c\rangle(S_1, S_2)$

```
   Data: $S_1$: set variable, $S_2$: set variable
   Result: Returns the bounds updated to be BC.
 1 if $hs_{LL}\langle c\rangle(S_1, S_2)$ then
 2     $lb_{LL}(S_1) = succ_{LL,S_1}\langle c\rangle(S_1, S_2)$;
 3     $ub_{LL}(S_1) = pred_{LL,S_1}\langle c\rangle(S_1, S_2)$;
 4     $lb_{LL}(S_2) = succ_{LL,S_2}\langle c\rangle(S_2, S_1)$;
 5     $ub_{LL}(S_2) = succ_{LL,S_2}\langle c\rangle(S_2, S_1)$;
 6     return true
 7 end
 8 else
 9     return false
10 end
```

The two functions $succ_{LL,S_1}$ and $pred_{LL,S_1}$ are built on top of the feasibility routine hs_{LL}. The authors give the generic $succ_{LL,S_1}$ algorithm with complexity $O(\alpha c^2 \log n)$ where α is the computational cost of hs_{LL}, c is the

Algorithm 12: $bc_{ML}\langle c\rangle(S_1, S_2)$

```
Data: $S_1$: set variable, $S_2$: set variable
Result: Returns the bounds updated to be BC.
if $hs_{ML}\langle c\rangle(S_1, S_2)$ then
    $lb_{ML}(S_1) = succ_{ML,S_1}\langle c\rangle(S_1, S_2)$;
    $ub_{ML}(S_1) = pred_{ML,S_1}\langle c\rangle(S_1, S_2)$;
    $lb_{ML}(S_2) = succ_{ML,S_2}\langle c\rangle(S_2, S_1)$;
    $ub_{ML}(S_2) = pred_{ML,S_2}\langle c\rangle(S_2, S_1)$;
    return true
end
else
    return false
end
```

cardinality of the sets (it could be smaller than the size of the universe) and n is the size of the universe.

Since *minlex* totally orders its sets, we use their same BC algorithm but in *minlex* version (as shown by Algorithm 12).

The definitions of hs_{ML}, $succ_{ML,S_1}$ and $pred_{ML,S_1}$ are analogous to the corresponding *lengthlex* counterparts. We build these functions on top of hs_{LL}, $succ_{LL,S_1}$ and $pred_{LL,S_1}$ functions, recalling them a polynomial number of times. Thus whenever their algorithms are polynomial so are ours. Hence our generic BC algorithm is also polynomial.

For the sake of simplicity, we focus only on non-empty sets. We start with hs_{ML}. We note the following: if a constraint $c(S_1, S_2)$ is satisfied by $x \in D_{ML}(S_1)$, $y \in D_{ML}(S_2)$ then x and y must belong to their embedded *lengthlex* subdomains respectively of cardinality $|x|$ and $|y|$. It is enough to search the solution over all possible combinations of *lengthlex* subdomains: they are at most $O(n^2)$. Therefore given a constraint c, for each lengthlex subdomain of S_1, for each lengthlex subdomain of S_2, we call the hs_{LL} function on these *lengthlex* intervals.

Example 23 *Take two variables S_1 and S_2 with the following domains: $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 3\}, \{2\}, \{2, 3\}, \{3\}\}$. Consider the constraint $|S_1 \cap S_2| \geq 2$: it can be satisfied by $x = \{1, 2\}$ and $y = \{1, 2, 3\}$. Thus hs_{ML} returns true because hs_{LL} returns true when called on the two lengthlex embedded domains of cardinality 2 (for S_1) and cardinality 3 (for S_2).*

We use the auxiliary function $llsub_{ML}(S_1, c)$ implemented by Algorithm 13. It returns the embedded *lengthlex* domain of cardinality c (it is a pair

$\langle lb, ub\rangle$) or $\langle\{-1\},\{-1\}\rangle$ if it does not include a *lengthlex* domain of such cardinality. It exploits the $first - c_{ML}$ and the $last - c_{ML}$ algorithms previously defined. Therefore its computational cost is $O(n^2)$. The soundness of Algorithm 13 is given by the following property.

Property 7 *Given a minlex domain $\langle lb, ub\rangle$ and a cardinality c, let s be the first set after lb of cardinality c and t the last set before ub of cardinality c which belongs to the domain, then for every set u such that $s \leq_{LL} u \leq_{LL} t$, u belongs also to the given minlex domain.*

Proof: It is enough to observe that in the *lengthlex* ordering values of same cardinality are ordered lexicographically. Thus for each set u the following holds: $lb \leq_{ML} s \leq_{ML} u \leq_{ML} t \leq_{ML} ub$. □

```
Algorithm 13: $llsub_{ML}(S_1, c)$
  Data: $S_1$: set variable, $c$: integer, $U$: set
  Result: Returns the embedded lengthlex domain of cardinality c.
1 $lb = first - c_{ML}(S_1)(lb_{ML}(S), c, U)$;
2 $ub = last - c_{ML}(S_1)(ub_{ML}(S), c, U)$;
3 if $ub >_{ML} lb$ or $lb = \{-1\}$ or $ub = \{-1\}$ then
4     return $\langle\{-1\},\{-1\}\rangle$;
5 end
6 else
7     return $\langle lb, ub\rangle$;
8 end
```

Algorithm 14 realizes the feasibility routine hs_{ML}. The two nested loops ensure that all possible *lengthlex* subdomains will be traversed. The algorithm returns true when a pair that satisfies the constraint is found (lines 5-6), false otherwise. Line 2 costs $O(n^2)$ while the for loop of lines 3-9 costs $O(n^3\alpha)$, where α is the computational cost of hs_{LL}. Therefore hs_{ML} takes $O(n^4\alpha)$.

Algorithm 14: $hs_{ML}\langle c\rangle(S_1, S_2)$

```
Data: $S_1$: set variable, $S_2$: set variable
Result: Returns true if the constraint is satisfied over the two minlex
        intervals, otherwise false
for $i = 1$ to $n$ do
    if $llsub_{ML}(S_1, i) \neq \langle\{-1\}, \{-1\}\rangle$ then
        for $j = 1$ to $n$ do
            if $llsub_{ML}(S_2, j) \neq \langle\{-1\}, \{-1\}\rangle$ then
                if $hs_{LL}\langle c\rangle(llsub_{ML}(S_1, i), llsub_{ML}(S_2, j))$ then
                    return $true$
                end
            end
        end
    end
end
return $false$
```

Algorithm 15 realizes the $succ_{ML,S_1}\langle c\rangle(S_1, S_2)$ function. It exploits the same technique used before. The min_{ML} function (line 8) ensures that the variable $result$ will contain the smallest set of $D_{ML}(S_1)$ which satisfy the constraint, if it exists. Otherwise it returns $\{-1\}$. Considering that $succ_{LL,S_1}$ takes $O(\alpha n^2 \log n)$, then $succ_{ML,S_1}$ is $O(\alpha n^4 \log n)$. $pred_{ML,S_1}$ can be built in the same way, computing max_{ML} instead of min_{ML}.

We will now show that BC on some commons binary constraints is tractable for *minlex* representation. Moreover we will show that the binary lexicographic constraint can be propagated very efficiently.

5.6.1 Disjoint, atleast-k and atmost-k

In [22] the authors give specialized polynomial algorithms ($hs_{LL}\langle c\rangle$ and $succ_{LL,S_1}$ $\langle c\rangle$) for the binary disjoint constraint ($|S_1 \cap S_2| = 0$). Thus we conclude that enforcing BC for a binary disjoint constraint is polynomial also for the *minlex* representation. Moreover, in the same paper, they claim to have polynomial algorithms for the $atleast{-}k$ ($|S_1 \cap S_2| \geq k$) and the $atmost{-}k$ ($|S_1 \cap S_2| \leq k$) constraints. Thus we can state that $atleast - k$ and $atmost - k$ are tractable also for the *minlex* representation.

Algorithm 15: $succ_{ML,S_1}\langle c\rangle(S_1, S_2)$

```
Data: $S_1$: set variable, $S_2$: set variable
Result: Returns the first value of $S_1$ which satisfy the constraint
result = {-1};
for $i = 1$ to $n$ do
    if $llsub_{ML}(S_1, i) \neq \langle\{-1\}, \{-1\}\rangle$ then
        for $j = 1$ to $n$ do
            if $llsub_{ML}(S_2, j) \neq \langle\{-1\}, \{-1\}\rangle$ then
                if $succ_{LL,S_1}\langle c\rangle(llsub_{ML}(S_1, i), llsub_{ML}(S_2, j)) \neq \{-1\}$ then
                    if $result \neq \{-1\}$ then
                        $result = min_{ML}(result, succ_{LL,S_1}\langle c\rangle(llsub_{ML}(S_1, i), llsub_{ML}(S_2, j)))$;
                    end
                    else
                        $result = succ_{LL,S_1}\langle c\rangle(llsub_{ML}(S_1, i), llsub_{ML}(S_2, j))$;
                    end
                end
            end
        end
    end
end
return result
```

Algorithm 16: $bc_{ML}\langle lex_{\leq}\rangle(S_1, S_2)$

```
  Data: $S_1$: set variable, $S_2$: set variable
  Result: Enforce BC for the constraint $S_1 \leq_{lex} S_2$
1 $ub_{ML}(S_1) = min_{ML}(ub_{ML}(S_1), ub_{ML}(S_2))$;
2 $lb_{ML}(S_2) = max_{ML}(lb_{ML}(S_1), lb_{ML}(S_2))$;
```

5.6.2 Binary lexicographic constraint

Due to the nature of the *minlex* ordering, it is very easy to enforce BC for a binary lexicographic constraint $S_1 \leq_{lex} S_2$. In fact *minlex* totally orders the domain w.r.t. the lexicographic order. Thus we have only to adjust the upper bound of S_1, taking the minimum between itself and the upper bound of S_2, and the lower bound of S_2, taking the maximum between itself and the lower bound of S_1 as shown in Algorithm 16 (lines 1 and 2). Note that the lexicographic binary constraint of [11] is not purely lexicographic: it considers also cardinality. Lexicographic constraints are a point of strength of *minlex* representation.

Chapter 6

Generalizing the framework

In this chapter we generalize the framework developed in Chapter 4 to deal with an arbitrary number of representations. In fact the recent trend is to consider more than two representations at a time. Given that there exist combinations of three representations in the literature, we study various triple combinations of *minlex*, *SB* and *card*. We then position $lengthlex \cdot SB$ [28] and hybrid domain [18, 19] within our framework. We show that $lengthlex \cdot SB$ is incomparable to all *minlex* based representations and hybrid domain does not maintain consistency between all the bounds at the same time.

6.1 Combining multiple representations

In this section, we provide the notions of synchronization and bound consistency between combinations of an arbitrary number of representations. The combinations that are supported by the extended framework are strong combinations $I = R_1 \cdots R_n$ and weak combinations $H = I_1 + \ldots + I_n$.

6.1.1 Synchronization of bounds

When we combine more than two representations, synchronization can be understood in two different ways. Synchronization for a strong combination requires that each bounds of each representation is consistent with respect to the joint domain. On the other hand, if we have a weak combination H we require that each representation must be consistent with each strong combination. This allows us to have different types of synchronization depending on how the representations are combined.

Definition 15 (Synchronization for $R_1 \cdots R_n$) *Given a variable S, the representation R_i and the combination $R_1 \cdots R_{i-1} \cdot R_{i+1} \cdots R_n$, the representation R_i is synchronized with $R_1 \cdots R_{i-1} \cdot R_{i+1} \cdots R_n$ on S iff:*

- $lb_{R_i}(S) = glb_{R_i}(D_{R_1 \cdots R_n}(S))$;
- $ub_{R_i}(S) = lub_{R_i}(D_{R_1 \cdots R_n}(S))$.

S is synchronized for $R_1 \cdots R_n$ iff $\forall i \in \{1, \ldots, n\}$, R_i is synchronized with $R_1 \cdots R_{i-1} \cdot R_{i+1} \cdots R_n$ on S.

Definition 16 (Synchronization for $R_1 \cdots R_n + \ldots + R_{n+k} \cdots R_m$) *Given a variable S, and the combination $R_{i_1} \cdots R_{j_1} + R_{i_2} \cdots R_{j_2}$, we say that $R_{i_1} \cdots R_{j_1}$ is synchronized with $R_{i_2} \cdots R_{j_2}$ on S iff S is synchronized for $R_{i_1} \cdots R_{j_1}$ and $\forall i \in \{i_1, \ldots, j_1\}$, R_i is synchronized with $R_{i_2} \cdots R_{j_2}$ on S.*

S is synchronized for $R_1 \cdots R_n + \ldots + R_{n+k} \cdots R_m$ iff every strong combination is synchronized with the others on S.

6.1.2 Bound consistency on combinations of multiple representations

Bound consistency on combination of representations is the natural extension of the one proposed in Chapter 4. In the following we first define BC for the strong combination and then for the weak.

Definition 17 (Bound consistency on $R_1 \cdots R_n$) *Given variables $S_1, \ldots, S_m$, and a constraint $c(S_1, \ldots, S_m)$, S_i is bound consistent on c from R_j to $R_1 \cdots R_{j-1} \cdot R_{j+1} \cdots R_n$ iff:*

- $lb_{R_j}(S) = glb_{R_j}(c[S_i]_{R_1 \cdots R_n})$;
- $ub_{R_j}(S) = lub_{R_j}(c[S_i]_{R_1 \cdots R_n})$;

S_i is bound consistent on c for $R_1 \cdots R_n$ iff S_i is synchronized for $R_1 \cdots R_n$ and $\forall j \in \{1, \ldots, n\}$, S_i is bound consistent from R_j to $R_1 \cdots R_{j-1} \cdot R_{j+1} \cdots R_n$.

Definition 18 (Bound consistency on $R_1 \cdot \cdot R_n + .. + R_{n+k} \cdot \cdot R_m$) *Given variables $S_1, \ldots, S_p$ and a constraint $c(S_1 \ldots, S_p)$, S_i is bound consistent on c for $R_1 \cdots R_n + \ldots + R_{n+k} \cdots R_m$ iff S_i is BC on c for every strong combination and S_i is synchronized for $R_1 \cdots R_n + \ldots + R_{n+k} \cdots R_m$.*

6.2 Comparison of multiple combinations

We define effective propagation to be able to compare different combinations of representations. As in Chapter 4, we measure the ability to remove values from the original domains when BC is applied.

Definition 19 (Effective Propagation) *Given a constraint $c(S_1, \dots, S_n)$, a combination $R_1 \cdots R_n + \dots + R_{n+k} \cdots R_m$, the original domain $D(S_i)$ of S_i, $D_{R_j}(S_i)$ the tightest approximation of $D(S_i)$ using R_j, and $D'_{R_j}(S_i)$ the domain of S_i after the BC propagation of c, the* effective propagation *of c on S_i is the set $p_{R_1 \cdots R_n + \dots + R_{n+k} \cdots R_m}\langle c\rangle(S_i) = D(S_i) \cap D'_{R_1}(S_i) \cap \cdots \cap D'_{R_m}(S_i)$ of the values from $D(S_i)$ remaining in the intersection of approximated domain $D'_{R_1} \cap \cdots \cap D'_{R_m}(S_i)$ after BC has been enforced on c.*

Then the stronger relation $\succeq$ is naturally extended as follows.

Definition 20 (Stronger relation $\succeq$) *Let H refer to a combined representation $R_1 \cdots R_n + \dots + R_{n+k} \cdots R_m$.*
Given a constraint $c(S_1, \dots, S_p)$ we say that H_1 is stronger than H_2 on c ($H_1 \succeq_c H_2$) iff $\forall D(S_i)$ with $D_{R_j}(S_i)$ the tightest approximation of $D(S_i)$ using R_j, if there exists S_i such that $p_{H_2}\langle c\rangle(S_i) = \emptyset$ then $p_{H_1}\langle c\rangle(S_i) = \emptyset$.

We say that $H_1 \succeq H_2$ iff $H_1 \succeq_c H_2$ for all constraints c. We say that H_1 is strictly stronger than H_2 on c ($H_1 \succ_c H_2$) iff $H_1 \succeq_c H_2$ and $H_2 \not\succeq_c H_1$. We say that H_1 is strictly stronger than H_2 ($H_1 \succ H_2$) iff $H_1 \succeq H_2$ and $H_2 \not\succeq H_1$. We say that H_1 and H_2 are incomparable ($H_1 \sim H_2$) iff $H_1 \not\succeq H_2$ and $H_2 \not\succeq H_1$.

As done for the framework of Chapter 4, we now give the needed theorems to compare combinations of representations.

Theorem 20 *Let I refer to a combined representation $R_i \cdots R_j$. Given two combinations $H_1 = I_1 + \dots + R_i \cdots R_j + \dots + I_n$ and $H_2 = I_1 + \dots + R_i \cdots R_k + R_{k+1} \cdots R_j + \dots + I_n$, we have that $H_1 \succeq H_2$.*

Proof: We note that the H_1 is equal to H_2 except that $R_i \cdots R_j$ of H_1 is split in $R_i \cdots R_k + R_{k+1} \cdots R_j$ in H_2.

Let $c(S_1, \dots, S_p)$ be a constraint. Consider a set t from $D(S_i)$ such that $t \notin p_{H_2}\langle c\rangle(S_i)$. This means that t was removed either by BC on c for each strong combination of H_2 or by synchronization between those.

Consider now BC on c for H_1. The BC definition guarantees at least BC on c for every strong combination of H_2 and also synchronization between them.

Thus $t \notin p_{H_1}\langle c\rangle(S_i)$. As a result, if $p_{H_2}\langle c\rangle(S_i)$ is empty then $p_{H_1}\langle c\rangle(S_i)$ is empty as well. $\square$

Theorem 21 *Let I refer to a combined representation $R_i \cdots R_j$. We have that $I_1 + I_2 + \ldots + I_n$ is stronger than I_k for all $k \in \{1, \ldots, n\}$.*

Proof: Let $c(S_1, \ldots, S_p)$ be a constraint. Consider a set t from $D(S_i)$ such that $t \notin p_{I_k}\langle c\rangle(S_i)$. This means that t was removed by BC on c for I_k. Consider now BC on c for $I_1 + \ldots + I_n$. The BC definition guarantees at least that S_i must be BC on c for I_j, $j \in \{1, \ldots, n\}$.

Thus $t \notin p_{I_1+\ldots+I_n}\langle c\rangle(S_i)$. As a result, if $p_{I_k}\langle c\rangle(S_i)$ is empty, $p_{I_1+\ldots+I_n}\langle c\rangle(S_i)$ is empty as well. $\square$

6.3 Using the framework

We now study various triple combinations of *minlex*, *SB* and *card* representations. Then we position *lengthlex* $\cdot$ *SB* [28] and the hybrid domain [18, 19] within our framework. Before using the framework we will instantiate the synchronization property given in Definition 15 for synchronization between triple *minlex* based combinations. This instantiation is useful in order to actively use the framework and follow the theoretical study.

6.3.1 Instantiation of synchronization

In addition to the theorems used in Section 4.4.1, we give the following theorems in order to obtain the desired instantiation.

Theorem 22 *Given a set variable S, the representation card and the combination minlex$\cdot$SB, card is synchronized with minlex$\cdot$SB on S iff $\exists s_l, s_u \in D_{ML\cdot SB}(S)$ such that $|s_l| = lb_C(S)$, $|s_u| = ub_C(S)$.*

Proof: $\Longrightarrow$ By Definition 15, $lb_C(S) = glb_C(D_{ML\cdot SB\cdot C}(S)) = glb_{\preceq_C}(\{[s] \mid s \in D_{ML\cdot SB\cdot C}(S)\})$. Since $\preceq_C$ is a total ordering, it means that there exists at least a set $s_l \in D_{ML\cdot SB}(S)$ whose cardinality is $lb_C(S)$. Similarly for $ub_C(S)$. $\Longleftarrow$ Suppose that $\exists s_l, s_u \in D_{ML\cdot SB}(S)$ such that $|s_l| = lb_C(S)$ and $|s_u| = ub_C(S)$. It means that s_l, $s_u \in D_{ML\cdot SB\cdot C}(S)$. Since $\preceq_C$ is a total order, $[s_l]$ and $[s_u]$ are the smallest (resp. greatest) equivalence class of the sets contained in $D_{ML\cdot SB\cdot C}(S)$ with respect to $\preceq_C$ ordering. Thus $[s_l] = glb_{\preceq_C}(\{[s] \mid s \in D_{ML\cdot SB\cdot C}(S)\})$ and $[s_u] = lub_{\preceq_C}(\{[s] \mid s \in D_{ML\cdot SB\cdot C}(S)\})$. $\square$

Theorem 23 *Given a set variable S, the representation SB and the combination $minlex \cdot card$, SB is synchronized with $minlex \cdot card$ on S iff $lb_{SB}(S) \supseteq \cap_{s \in D_{ML \cdot C}} s$ and $ub_{SB}(S) \subseteq \cup_{s \in D_{ML \cdot C}} s$.*

Proof: $\Longrightarrow$ By Definition 15 $lb_{SB}(S) = glb_{SB}(D_{ML \cdot SB \cdot C}(S)) = \cap_{s \in D_{ML \cdot SB \cdot C}(S)} s \supseteq \cap_{s \in D_{ML \cdot C}(S)} s$ and $ub_{SB}(S) = lub_{SB}(D_{ML \cdot SB \cdot C}(S)) = \cup_{s \in D_{ML \cdot SB \cdot C}(S)} s \subseteq \cup_{s \in D_{ML \cdot C}(S)} s$.
$\Longleftarrow$ Suppose that $lb_{SB}(S) \supseteq \cap_{s \in D_{ML \cdot C}(S)} s$ and $ub_{SB}(S) \subseteq \cup_{s \in D_{ML \cdot C}(S)} s$. We have that $\cap_{s \in D_{ML \cdot C}(S)} s \subseteq lb_{SB}(S) \subseteq ub_{SB}(S) \subseteq \cup_{s \in D_{ML \cdot C}(S)} s$. Thus $lb_{SB}(S)$ is the greatest lower bound of $D_{ML \cdot SB \cdot C}(S)$. Indeed it contains at least all the values commons to each set of $D_{ML \cdot C}(S)$ and it does not contain values that does not appear in $D_{ML \cdot C}(S)$. Moreover it is the greatest lower bound of the SB domain by construction. Similarly for $ub_{SB}(S)$. $\Box$

Theorem 24 *Given a set variable S, the representation minlex and the combination $SB \cdot card$, minlex is synchronized with $SB \cdot card$ on S iff $lb_{ML}(S) \in D_{SB \cdot C}(S)$ and $ub_{ML}(S) \in D_{SB \cdot C}(S)$.*

Proof: $\Longrightarrow$ If $lb_{ML}(S) = glb_{ML}(D_{ML \cdot SB \cdot C}(S))$ then $lb_{ML}(S) \in D_{ML \cdot SB \cdot C}(S)$ as *minlex* totally orders its sets. Thus $lb_{ML}(S) \in D_{SB \cdot C}(S)$. Similarly for $ub_{ML}(S)$.
$\Longleftarrow$ Suppose that $lb_{ML}(S) \in D_{SB \cdot C}(S)$. Then $lb_{ML}(S) \in D_{ML \cdot SB \cdot C}(S)$. Moreover it is the greatest lower bound of the intersection because it is the smallest set of the *minlex* domain. Similarly for $ub_{ML}(S)$. $\Box$

Given a variable S and a combination $R_1 + R_2 \cdot R_3$, S is synchronized for $R_1 + R_2 \cdot R_3$ iff S is synchronized for $R_2 \cdot R_3$, R_1 is synchronized with $R_2 \cdot R_3$ on S and R_2 and R_3 are synchronized independently with R_1 on S. Thus using the above theorems together with the analogous theorems of Chapter 4 we can easily obtain the following instantiations for combinations with *minlex*, SB and *card*.

Combination $minlex \cdot SB + card$ Given the three representations *minlex*, SB and *card* a variable S is synchronized for $minlex \cdot SB + card$ iff:

- S is synchronized for *minlex* and SB;
- $\exists s_l, s_u \in D_{ML \cdot SB}(S)$ such that $|s_l| = lb_C(S)$, $|s_u| = ub_C(S)$;
- $lb_C(S) \leq |lb_{ML}(S)| \leq ub_C(S)$ and $lb_C(S) \leq |ub_{ML}(S)| \leq ub_C(S)$;
- $lb_{SB}(S) = \cap_{s \in D_{SB \cdot C}(S)} s$ and $ub_{SB}(S) = \cup_{s \in D_{SB \cdot C}(S)} s$.

Combination $minlex \cdot card + SB$ Given the three representations $minlex$, $card$ and SB a variable S is synchronized for $minlex \cdot card + SB$ iff:

- S is synchronized for $minlex$ and $card$;
- $lb_{SB}(S) \supseteq \cap_{s \in D_{ML \cdot C}} s$ and $ub_{SB}(S) \subseteq \cup_{s \in D_{ML \cdot C}} s$;
- $lb_{ML}(S) \in D_{SB}(S)$ and $ub_{ML}(S) \in D_{SB}(S)$;
- $\exists s_l, s_u \in D_{SB \cdot C}(S)$ such that $|s_l| = lb_C(S)$, $|s_u| = ub_C(S)$.

Combination $SB \cdot card + minlex$ Given the three representations SB, $card$ and $minlex$ a variable S is synchronized for $SB \cdot card + minlex$ iff:

- S is synchronized for SB and $card$;
- $lb_{ML}(S) \in D_{SB \cdot C}(S)$ and $ub_{ML}(S) \in D_{SB \cdot C}(S)$;
- $lb_{SB}(S) \supseteq \cap_{s \in D_{ML}} s$ and $ub_{SB}(S) \subseteq \cup_{s \in D_{ML}} s$;
- $\exists s_l, s_u \in D_{ML}(S)$ such that $|s_l| = lb_C(S)$, $|s_u| = ub_C(S)$.

We now instantiate the synchronization property for $minlex \cdot card \cdot SB$ and $minlex + card + SB$.

Combination $minlex \cdot SB \cdot card$ Given the three representations $minlex$, SB and $card$ a variable S is synchronized for $minlex \cdot SB \cdot card$ iff:

- $lb_{ML}(S) = glb_{ML}(D_{ML \cdot SB \cdot C}(S))$ and $ub_{ML}(S) = lub_{ML}(D_{ML \cdot SB \cdot C}(S))$;
- $lb_{SB}(S) = glb_{SB}(D_{ML \cdot SB \cdot C}(S))$ and $ub_{SB}(S) = lub_{SB}(D_{ML \cdot SB \cdot C}(S))$;
- $lb_C(S) = glb_C(D_{ML \cdot SB \cdot C}(S))$ and $ub_C(S) = lub_C(D_{ML \cdot SB \cdot C}(S))$.

Combination $minlex + SB + card$ Given the three representations $minlex$, SB and $card$ a variable S is synchronized for $minlex + SB + card$ iff:

- S is synchronized for $minlex$ and SB;
- S is synchronized for $minlex$ and $card$;
- S is synchronized for SB and $card$.

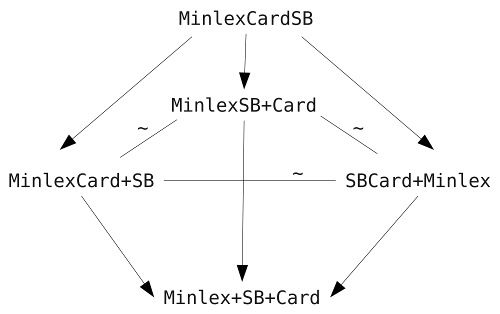

Figure 6.1: Comparisons of triple combinations with *minlex*, *SB* and *card*. $H_1 \to H_2$ means $H_1 \succ H_2$ whilst $H_1 \tilde{-} H_2$ means that $H_1 \sim H_2$.

6.3.2 Comparing *minlex*, *card* and *SB* combinations

In this section we study triple combinations with *minlex*, *card* and *SB*. The results are summarized in Figure 6.1. Comparisons are divided into three sections, covering the three levels of the graph of Figure 6.1.

First level The following 3 theorems shows that $minlex{\cdot}card{\cdot}SB$ is strictly stronger than $minlex \cdot card + SB$, $minlex \cdot SB + card$ and $SB \cdot card + minlex$, as expected.

Theorem 25 $minlex \cdot card \cdot SB \succ minlex \cdot card + SB$.

Proof: By Theorem 20, $minlex \cdot card \cdot SB \succeq minlex \cdot card + SB$.

To show that $minlex \cdot card \cdot SB \succ minlex \cdot card + SB$, take the constraint $|S_1 \cap S_2| = 1$ on the two set variables S_1 and S_2 which take value from the universe $U = \{1, 2, 3, 4, 5\}$. Assume that $D(S_1) = D(S_2) = \{\{1, 5\}, \{3, 4\}\}$. The *minlex*, *SB* and *card* domain are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1, 5\}, \{2\}, \{2, 3\}, \{2, 3, 4\}, \{2, 3, 4, 5\}, \{2, 3, 5\}, \{2, 4\}, \{2, 4, 5\}, \{2, 5\}, \{3\}, \{3, 4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 3, 4, 5\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex \cdot SB \cdot card$ clearly fails since the constraint cannot be satisfied in the joint domain. Instead $minlex \cdot card + SB$ does not prune. $\square$

Theorem 26 $minlex \cdot card \cdot SB \succ minlex \cdot SB + card$.

Proof: By Theorem 20, $minlex \cdot card \cdot SB \succeq minlex \cdot SB + card$.

To show that $minlex \cdot card \cdot SB \succ minlex \cdot SB + card$, take the constraint $S_1 \cup S_2 \supseteq \{2,3,4\}$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1,2,3,4\}$. Assume that $D(S_1) = D(S_2) = \{\{1,2\},\{1,3\},\{1,4\}\}$. The *minlex*, *SB* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},$ $\{1,3,4\},\{1,4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{1\}, ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,2,3,4\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex \cdot SB \cdot card$ clearly fails and $minlex \cdot SB + card$ does not prune. □

Theorem 27 $minlex \cdot card \cdot SB \succ SB \cdot card + minlex$.

Proof: By Theorem 20, $minlex \cdot card \cdot SB \succeq SB \cdot card + minlex$.

To show that $minlex \cdot card \cdot SB \succ SB \cdot card + minlex$, take the constraint $|S_1 \cap S_2| = 1$ on the two set variables S_1 and S_2 which take value from the universe $U = \{1,2,3,4,5\}$. Assume that $D(S_1) = D(S_2) = \{\{1,5\},\{3,4\}\}$. The *minlex*, *SB* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,5\},\{2\},\{2,3\},\{2,3,4\},\{2,3,4,5\},\{2,3,5\},$ $\{2,4\},\{2,4,5\},\{2,5\},\{3\},\{3,4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,3,4,5\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex \cdot SB \cdot card$ clearly fails since the constraint cannot be satisfied in the joint domain. Instead $SB \cdot card + minlex$ does not prune. □

Second level In the following, we compare $minlex \cdot card + SB$, $minlex \cdot SB + card$ and $SB \cdot card + minlex$. It is interesting to see that all these combinations are pairwise incomparable. The reason for this is due to the different ways to propagate the constraints for the different combinations.

Theorem 28 $minlex \cdot card + SB \sim minlex \cdot SB + card$.

Proof: We first show $minlex \cdot card + SB \not\succeq minlex \cdot SB + card$. Take the constraint $|S_1 \cap S_2| = 1$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1,2,3,4,5\}$. Assume that $D(S_1) = D(S_2) = \{\{1,5\},\{3,4\}\}$. The *minlex*, *SB* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,5\},\{2\},\{2,3\},\{2,3,4\},\{2,3,4,5\},\{2,3,5\},$ $\{2,4\},\{2,4,5\},\{2,5\},\{3\},\{3,4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,3,4,5\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex \cdot SB + card$ prune all the sets from all the domains ($\{3\}$ is pruned thanks to synchronization with $card$) and $minlex \cdot card + SB$ does not prune.

We now show $minlex \cdot SB + card \not\succeq minlex \cdot card + SB$. Take the constraint $S_1 \cup S_2 \supseteq \{2,3,4\}$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1,2,3,4\}$. Assume that $D(S_1) = D(S_2) = \{\{1,2\},\{1,3\},\{1,4\}\}$. The *minlex*, *SB* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},$ $\{1,3,4\},\{1,4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{1\}, ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,2,3,4\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex{\cdot}card{+}SB$ prunes all the sets from all the domains, $minlex{\cdot}SB{+}card$ does not prune. $\square$

Theorem 29 $minlex \cdot SB + card \sim SB \cdot card + minlex$.

Proof: We first show $minlex \cdot SB + card \not\succeq SB \cdot card + minlex$. Take the constraint $S_1 \cup S_2 \supseteq \{2,3,4\}$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1,2,3,4\}$. Assume that $D(S_1) = D(S_2) = \{\{1,2\},\{1,3\},\{1,4\}\}$. The *minlex*, *SB* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},\{1,$ $3,4\},\{1,4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{1\}, ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,2,3,4\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$SB{\cdot}card{+}minlex$ prunes all the sets from all the domains. $minlex{\cdot}SB{+}card$ does not prune.

We now show $SB{\cdot}card{+}minlex \not\succeq minlex{\cdot}SB{+}card$. Take the constraint $|S_1 \cap S_2| = 1$ on the two set variables S_1 and S_2 which take value from the universe $U = \{1,2,3,4,5\}$. Assume that $D(S_1) = D(S_2) = \{\{1,5\},\{3,4\}\}$. The *minlex*, *SB* and *card* domain are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,5\},\{2\},\{2,3\},\{2,3,4\},\{2,3,4,5\},\{2,3,5\},$ $\{2,4\},\{2,4,5\},\{2,5\},\{3\},\{3,4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,3,4,5\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex \cdot SB + card$ prunes all the sets from all the domains ($\{3\}$ is pruned thanks to synchronization with $card$) whereas $SB \cdot card + minlex$ does not prune. □

Theorem 30 $SB \cdot card + minlex \sim minlex \cdot card + SB$.

Proof: We first show $minlex \cdot card + SB \not\succeq SB \cdot card + minlex$. Take the constraint $S_1 \cup S_2 \not\subseteq \{1,2,4\}$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1,2,3,4\}$. Assume that $D(S_1) = D(S_2) = \{\{1\},\{2\},\{4\}\}$. The $minlex$, SB and $card$ domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1\},\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,3\},$ $\{1,3,4\},\{1,4\},\{2\},$ $\{2,3\},\{2,3,4\},\{2,4\},\{3\},\{3,4\},\{4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,2,4\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 1$.

$SB{\cdot}card{+}minlex$ prunes all the sets from all the domains. $minlex{\cdot}card{+}SB$ does not prune: bounds of $minlex$ are supported by $\{3\}$ and for SB is enough to observe that empty is supported by $\{1,2,4\}$ and vice versa (so the intersection is empty and union is $\{1,2,4\}$).

We now show $SB \cdot card + minlex \not\succeq minlex \cdot card + SB$. Take the constraint $S_1 \cup S_2 \supseteq \{2,3,4\}$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1,2,3,4\}$. Assume that $D(S_1) = D(S_2) = \{\{\},\{1\},\{1,2\},\{1,3\},\{1,4\}\}$. The $minlex$, SB and $card$ domains are:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{\},\{1\},\{1,2\},\{1,2,3\},\{1,2,3,4\},\{1,2,4\},\{1,$ $3\},\{1,3,4\},\{1,4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,2,3,4\}$;
- $lb_C(S_1) = lb_C(S_2) = 0, ub_C(S_1) = ub_C(S_2) = 2$.

$minlex \cdot card + SB$ prunes all the sets from all the domains. For $SB \cdot card + minlex$, we have that $p_{SB \cdot C+ML}\langle c\rangle(S_1) = p_{SB \cdot C+ML}\langle c\rangle(S_1) = \{\{1\},\{1,2\},\{1,3\},$ $\{1,4\}\}$ because of card lower bound that is updated to 1. □

Third level Finally we show that $minlex \cdot card + SB$, $minlex \cdot SB + card$ and $SB \cdot card + minlex$ are strictly stronger than $minlex + SB + card$, as expected.

Theorem 31 $minlex \cdot card + SB \succ minlex + SB + card$

Proof: By Theorem 20, $minlex \cdot card + SB \succeq minlex + SB + card$.

To show that $minlex \cdot card + SB \succ minlex + SB + card$, take the constraint $S_1 \cup S_2 \supseteq \{2, 3, 4\}$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1, 2, 3, 4\}$. Assume that $D(S_1) = D(S_2) = \{\{1, 2\}, \{1, 3\}, \{1, 4\}\}$. The *minlex*, *SB* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}, \{1, 2, 4\}, \{1, 3\}, \{1, 3, 4\}, \{1, 4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{1\}, ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 2, 3, 4\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex{\cdot}card{+}SB$ prunes all the sets from all the domains, $minlex{+}SB{+}card$ does not prune. □

Theorem 32 $minlex \cdot SB + card \succ minlex + SB + card$

Proof: By Theorem 20, $minlex \cdot SB + card \succeq minlex + SB + card$.

To show that $minlex{\cdot}SB{+}card \succ minlex{+}SB{+}card$, take the constraint $|S_1 \cap S_2| = 1$ on the two set variables S_1 and S_2 which take value from the universe $U = \{1, 2, 3, 4, 5\}$. Assume that $D(S_1) = D(S_2) = \{\{1, 5\}, \{3, 4\}\}$. The *minlex*, *SB* and *card* domain are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1, 5\}, \{2\}, \{2, 3\}, \{2, 3, 4\}, \{2, 3, 4, 5\}, \{2, 3, 5\}, \{2, 4\}, \{2, 4, 5\}, \{2, 5\}, \{3\}, \{3, 4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 3, 4, 5\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$minlex \cdot SB + card$ prune all the set from all the domains ($\{3\}$ is pruned thanks to synchronization with *card*) and $minlex{+}card{+}SB$ does not prune. □

Theorem 33 $SB \cdot card + minlex \succ minlex + SB + card$

Proof: By Theorem 20, $SB \cdot card + minlex \succeq minlex + SB + card$.

To show that $SB \cdot card + minlex \succ minlex + SB + card$, take the constraint $S_1 \cup S_2 \supseteq \{2, 3, 4\}$ on the two set variables S_1 and S_2 which take values from the universe $U = \{1, 2, 3, 4\}$. Assume that $D(S_1) = D(S_2) = \{\{1, 2\}, \{1, 3\}, \{1, 4\}\}$. The *minlex*, *SB* and *card* domains are the following:

- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}, \{1, 2, 4\}, \{1, 3\}, \{1, 3, 4\}, \{1, 4\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{1\}, ub_{SB}(S_1) = ub_{SB}(S_2) = \{1, 2, 3, 4\}$;
- $lb_C(S_1) = lb_C(S_2) = ub_C(S_1) = ub_C(S_2) = 2$.

$SB \cdot card + minlex$ prunes all the sets from all the domains. $minlex + SB + card$ does not prune. □

6.3.3 *lengthlex* · *SB* in our framework

In Figure 6.2, we compare the various combinations of *minlex*, *SB* and *card* against *lengthlex* · *SB* defined in [14]. In [28], the authors refer to this representation as *ls*-domain; however they experiment with only *lengthlex* + *SB*. We prove that *lengthlex* · *SB* is incomparable to any *minlex* based combination. Again, the reason of this result is mainly due to how original domains are approximated.

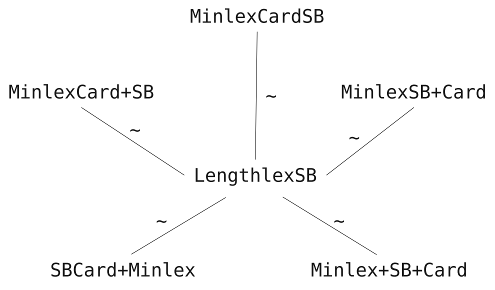

Figure 6.2: Comparisons with *lengthlex* · *SB*. $H_1 \to H_2$ means $H_1 \succ H_2$ whilst $H_1 \tilde{-} H_2$ means that $H_1 \sim H_2$.

Theorem 34 *lengthlex·SB is incomparable to minlex, minlex·SB, minlex·card, minlex + SB + card, minlex · card + SB, minlex · SB + card, SB · card + minlex and minlex · card · SB.*

Proof: We first show that $lengthlex \cdot SB \not\succeq minlex$. Take the constraint $S \geq_{lex} \{1,3\}$ on the set variable S which takes values from the universe $U = \{1,2,3\}$. Assume that $D(S) = \{\{\}, \{1\}, \{1,2\}, \{1,2,3\}\}$.

The *lengthlex*, *SB* and *minlex* domains are the following:

- $D_{ML}(S) = \{\{\}, \{1\}, \{1,2\}\{1,2,3\}\}$;
- $D_{LL}(S) = \{\{\}, \{1\}, \{2\}, \{3\}, \{1,2\}, \{1,3\}, \{2,3\}, \{1,2,3\}\}$;
- $lb_{SB}(S) = \{\}$, $ub_{SB}(S) = \{1,2,3\}$.

Enforcing BC on c for *minlex* prunes all the domain whereas $p_{LL \cdot SB}\langle c \rangle(S) = \{\{1,2\}\}$. Thus *lengthlex* is not stronger than all *minlex* based combinations with SB and *card*.

We now show that $minlex{\cdot}card{\cdot}SB \not\succeq lengthlex{\cdot}SB$. Take the constraint $S_1 \cup S_2 = \{1,2,3,4\}$ on two set variables S_1 and S_2 which take values from the universe $U = \{1,2,3,4\}$. Assume that $D(S_1) = D(S_2) = \{\{3\}, \{4\}, \{1,2\}\}$. The *minlex*, *lengthex*, *SB* and *card* domains are the following:

- $D_{LL}(S_1) = D_{LL}(S_2) = \{\{3\}, \{4\}, \{1,2\}\}$;
- $lb_{SB}(S_1) = lb_{SB}(S_2) = \{\}$, $ub_{SB}(S_1) = ub_{SB}(S_2) = \{1,2,3,4\}$;
- $D_{ML}(S_1) = D_{ML}(S_2) = \{\{1,2\}, \{1,2,3\}, \{1,2,3,4\}, \{1,2,4\}, \{1,3\}, \{1,3,4\}, \{1,4\}, \{2\}, \{2,3\}, \{2,3,4\}, \{2,4\}, \{3\}, \{3,4\}, \{4\}\}$;
- $lb_C(S_1) = lb_C(S_1) = 1$, $ub_C(S_2) = ub_C(S_2) = 2$.

$lengthlex{\cdot}SB$ prunes all the sets from all the domains whereas $p_{ML \cdot SB \cdot C}\rangle c \langle(S) = \{\{1,2\}\}$ (note that the lower bound of *card* is updated to 2). Thus all *minlex* based combinations with SB and *card* are not stronger than $lengthlex \cdot SB$.
□

6.3.4 Hybrid domain in our framework

In [19], a form of combination between *maxlex*, *SB* and *card*, called *hybrid domain*, is introduced. Related inference rules are given to maintain synchronization between the bounds as well as to enforce some form of local consistency on common set constraints. By just looking at the inference rules given to maintain synchronization between the bounds, we can see that the hybrid domain is a combination that falls between $maxlex + SB + card$ and $maxlex \cdot SB \cdot card$. In other words, the reason is that 2 of the bounds are updated by looking at the other two at the same time, which is a stronger reasoning than considering them mutually as we do in $maxlex{+}SB{+}card$. The reasoning however do not consider globally all the bounds at the same time. Consider for instance a variable S with the domain $lb_{maxlex}(S) = \{4, 2, 1\}$, $ub_{maxlex}(S) = \{4, 3, 1\}$, $lb_{SB}(S) = \{4\}$, $ub_{SB}(S) = \{1, 2, 3, 4\}$, $lb_C(S) = ub_C(S) = 3$. The inference rules will not change the subset bounds, but 1 must be in lb_{SB} for the three bounds to be synchronized together. Similarly, consider a variable S with the domain $lb_{maxlex}(S) = \{4, 3\}$, $ub_{maxlex}(S) = \{5, 1\}$, $lb_{SB}(S) = \{\}$, $ub_{SB}(S) = \{1, 2, 3, 4, 5\}$, $lb_C(S) = ub_C(S) = 2$. The inference rules will not change the subset bounds, but 2 must be removed from $ub_{SB}(S)$ for the three bounds to be synchronized together.

Chapter 7

Conclusions and future work

Set variables are natural objects for modelling a wide range of constraint satisfaction problems. In general, the real domain of a set variable is exponential in size. Therefore it is often approximated with a representation. A promising research area combines multiple representations [28, 18, 19]. These previous works have differed on how representations are combined, how consistency between representations is maintained, and how the constraints are propagated in order to reduce the search space. So far there was no framework to compare these combinations. This thesis bridged this gap.

In Chapter 4, we have devised a general framework for combining two different representations of set variables. We have defined the notion of synchronization between them. We have considered two ways to propagate a combined representation: a weak method which considers each representation independently, and a strong method which considers them together. We have proposed a way to compare representations that measures their ability to prune the search space. Then we started with an exhaustive pairwise study of different representations. We studied various mutual combinations of *minlex*, an effective representation for lexicographic constraints, with the two popular representations *SB* and *card*. We then positioned the existing mutual combinations of representations within our framework. This threw out an expected result: the *lengthlex* representation [11], which is designed to deal well with cardinality and lexicographic ordering constraints, is not better on such constraints than a combination which deals with lexicographic ordering (*minlex*) and cardinality separately.

In Chapter 5, we further studied *minlex* and we have shown that synchronization between *minlex* and *SB* and *minlex* and *card* can be achieved in polynomial time.

In Chapter 6, we extended the framework to deal with multiple representations. We studied various triple combinations of *minlex*, *SB* and *card* representations. We then positioned the existing triple combinations of representations within our framework ($lengthlex \cdot SB$ and hybrid domain). We have shown that $lengthlex \cdot SB$ is incomparable to all *minlex* based representations and hybrid domain does not maintain consistency between all the bounds at the same time.

Our framework gives a structured approach for combining different representations and provides theoretical tools to compare them. However, especially when representations are incomparable, a practical evaluation with standard constraint satisfaction problems is still needed. Therefore in order to evaluate the *minlex* combinations, we have to devise some efficient algorithms to propagate constraints over *minlex* based combinations. From the theoretical point of view, we also need to extend our framework to deal with expressiveness. So far, we have compared representations considering only their ability to prune the search space. This ability is affected by how representations approximate the real domains. Thus it could be very interesting to compare the expressiveness of the various representations captured in our framework.

Part of the hereby presented work is submitted to the Twenty-Fifth Conference on Artificial Intelligence (AAAI 2011).[1]

[1]http://www.aaai.org/Conferences/AAAI/aaai11.php

Bibliography

[1] F. Azevedo. Cardinal: A finite sets constraint solver. *Constraints*, 12(1):93–129, 2007.

[2] Nicolas Barnier and Pascal Brisset. Solving the kirkman's schoolgirl problem in a few seconds. In *CP*, pages 477–491, 2002.

[3] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. Disjoint, partition and intersection constraints for set and multiset variables. In *In CP-04*, pages 138–152. Springer-Verlag, 2004.

[4] Christian Bessiere, Emmanuel Hebrard, Brahim Hnich, and Toby Walsh. The complexity of reasoning with global constraints. *Constraints*, 12(2):239–259, 2007.

[5] David Cohen, editor. *Principles and Practice of Constraint Programming - CP 2010 - 16th International Conference, CP 2010, St. Andrews, Scotland, UK, September 6-10, 2010. Proceedings*, volume 6308 of *Lecture Notes in Computer Science*. Springer, 2010.

[6] C. J. Colbourn, J. H. Dinitz, and D. R. Stinson. Applications of combinatorial designs to communications, cryptography, and networking, 1999.

[7] ECLiPSe. http://eclipseclp.org.

[8] Dieter Fox and Carla P. Gomes, editors. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*. AAAI Press, 2008.

[9] Ian P. Gent, editor. *Principles and Practice of Constraint Programming - CP 2009, 15th International Conference, CP 2009, Lisbon, Portugal, September 20-24, 2009, Proceedings*, volume 5732 of *Lecture Notes in Computer Science*. Springer, 2009.

[10] Carmen Gervet. Conjunto: Constraint propagation over set constraints with finite set domain variables. In *ICLP*, page 733, 1994.

[11] Carmen Gervet and Pascal Van Hentenryck. Length-lex ordering for set CSPs. 2006.

[12] Peter Hawkins, Vitaly Lagoon, and Peter J. Stuckey. Solving set constraint satisfaction problems using robdds. *J. Artif. Intell. Res. (JAIR)*, 24:109–156, 2005.

[13] Donald L. Kreher and Douglas R. Stinson. Combinatorial algorithms: generation, enumeration, and search. *SIGACT News*, 30(1):33–35, 1999.

[14] Yuri Malitsky, Meinolf Sellmann, and Willem Jan van Hoeve. Length-lex bounds consistency for knapsack constraints. In Stuckey [21], pages 266–281.

[15] J.F. Puget. PECOS: a high level constraint programming language. In *Singapore international conference on Intelligent Systems (SPICIS)*, pages 137–142, 1992.

[16] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[17] Andrew Sadler and Carmen Gervet. Global reasoning on sets. In *Proc. of CP Workshop on Modelling and Problem Formulation*, 2001.

[18] Andrew Sadler and Carmen Gervet. Hybrid set domains to strengthen constraint propagation and reduce symmetries. In Wallace [24], pages 604–618.

[19] Andrew Sadler and Carmen Gervet. Enhancing set constraint solvers with lexicographic bounds. *J. Heuristics*, 14(1):23–67, 2008.

[20] Meinolf Sellmann. On decomposing knapsack constraints for length-lex bounds consistency. In *CP*, pages 762–770, 2009.

[21] Peter J. Stuckey, editor. *Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*, volume 5202 of *Lecture Notes in Computer Science*. Springer, 2008.

[22] Pascal Van Hentenryck, Justin Yip, Carmen Gervet, and Grégoire Dooms. Bound consistency for binary length-lex set constraints. In Fox and Gomes [8], pages 375–380.

[23] Mark Wallace. Practical applications of constraint programming. *Constraints*, 1(1/2):139–168, 1996.

[24] Mark Wallace, editor. *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, 2004.

[25] Toby Walsh. Consistency and propagation with multiset constraints: A formal viewpoint. In *CP*, pages 724–738, 2003.

[26] Justin Yip and Pascal Van Hentenryck. Evaluation of length-lex set variables. In Gent [9], pages 817–832.

[27] Justin Yip and Pascal Van Hentenryck. Length-lex bound consistency for knapsack constraints. In *SAC*, pages 1397–1401, 2009.

[28] Justin Yip and Pascal Van Hentenryck. Exponential propagation for set variables. In Cohen [5], pages 499–513.

Acknowledgements

It is a pleasure to thank the person who made this thesis possible, prof. Zeynep Kiziltan. Her perpetual energy and enthusiasm in research had motivated me. In addition, she was always willing to help and correct me. I would like to thank prof. Christian Bessiere: this work is the result of a close collaboration with him. His insights were always brilliant. I also thank Toby Walsh, who contributed to this work with his vast experience in constraint programming.

Ringrazio la mia famiglia per avermi supportato e sopportato nei momenti di difficoltà e non. Ringrazio di cuore gli amici, i vecchi, i nuovi e il Matto. Un grazie a Squera, per avermi dedicato parte del suo tempo. Infine grazie a te, Chiara: ogni altra parola banalizzerebbe ciò che provo per te.