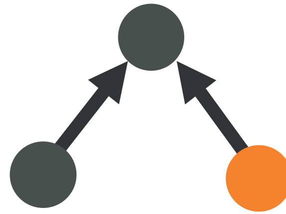
# SOFT CONSTRAINTS IN MINIBRASS:
## FOUNDATIONS AND APPLICATIONS



Alexander Schiendorfer

Dissertation
zur Erlangung des Doktorgrades
*doctor rerum naturalium* (Dr. rer. nat.)
der Fakultät für Angewandte Informatik
der Universität Augsburg, 2018

| | |
|---|---|
| Erstgutachter: | Prof. Dr. Wolfgang Reif, Universität Augsburg |
| Zweitgutachter: | Prof. Dr. Alexander Knapp, Universität Augsburg |
| Drittgutachter: | Dr. Guido Tack, Monash University, Melbourne, Australien |
| | |
| Tag der mündlichen Prüfung: | 12. November 2018 |

# Abstract

Over-constrained problems are ubiquitous in real-world decision and optimization problems, in particular, those emerging from self-organizing, autonomous systems where the full problem specification is only available at runtime. To address over-constrainedness, several theoretical formalisms to describe soft constraints have been proposed, including weighted, fuzzy, or probabilistic constraints. All of them were shown to be instances of algebraic structures such as valuation structures or c-semirings.

In terms of implemented modeling languages and solvers, however, the field of soft constraints lags far behind the state of the art in classical constraint optimization. Therefore, this dissertation describes *MiniBrass*, a versatile soft constraint modeling language building on the unifying algebraic framework of partially-ordered valuation structures (PVS). It is implemented as an extension of MiniZinc and MiniSearch. The most important characteristics of MiniBrass, and the ones that distinguish it from previous work, are that it is extensible and modular, supports a variety of concrete soft constraint formalisms, works with many solvers (inherited from MiniZinc), admits a graphical modeling language, and has been applied to several real-life case studies.

Contributions of this dissertation include the following: (1) The design, implementation, and performance evaluation of MiniBrass using 28 benchmark problems and six solvers, (2) A formal foundation that includes the systematic derivation of partially-ordered valuation structures and c-semirings from partial orders using basic category theory, (3) The qualitative soft constraint formalism *constraint preferences*, and (4) concepts for multi-agent optimization with (possibly) antagonistic preferences, including lexicographic and Cartesian products as well as voting operators.

// Kurzfassung

Zahlreiche Entscheidungs- und Optimierungsprobleme in praktischen Anwendungen sind überbestimmt, also unlösbar mit der gegebenen Menge an Nebenbedingungen (Constraints). Im Besonderen betrifft dies jene Probleme, die dem Umfeld selbstorganisierender oder autonomer Systeme entstammen und deren Parameter erst zur Laufzeit vollständig bekannt sind. Zur Spezifikation und Lösung solcher Probleme wurden mehrere theoretische Formalismen vorgestellt, wie zum Beispiel gewichtete, unscharfe oder probabilistische Constraints, welche die Beschreibung weicher Bedingungen ermöglichen, um Probleme lösbar zu machen. All diese Formalismen lassen sich als Instanzen algebraischer Strukturen wie Bewertungsstrukturen oder C-Halbringen ausdrücken.

Was tatsächlich implementierte Modellierungssprachen betrifft, hinkt das Gebiet der weichen Bedingungen allerdings weit dem Stand der Technik in klassischer Constraint-Optimierung hinterher. Zu diesem Zweck stellt diese Dissertation *MiniBrass* vor, eine vielseitige Modellierungssprache für weiche Bedingungen, die auf dem vereinheitlichenden algebraischen Rahmen der partiell geordneten Bewertungsstrukturen (eng. *PVS*) aufsetzt. Sie ist als Erweiterung zu MiniZinc und MiniSearch implementiert. Die wichtigsten Eigenschaften des MiniBrass-Systems (und zugleiche jene, die es von vorherigen Ansätzen abgrenzen) sind, dass es erweiterbar und modular ist, eine Vielzahl von konkreten Formalismen unterstützt, für zahlreiche Solver übersetzen kann (durch die Übersetzung nach MiniZinc), eine grafische Modellierungsform erlaubt und auf mehrere Fallstudien angewandt wurde.

Zu den Beiträgen dieser Dissertation zählen: (1) der Entwurf, die Implementierung und die experimentelle Evaluation der Leistungsfähigkeit von MiniBrass auf 28 Benchmark-Problemen mit sechs Solvern; (2) eine formale Fundierung, welche die systematische Konstruktion partiell geordneter Bewertungsstrukturen und von C-Halbringen basierend auf partiellen Ordnungen mittels elementarer Kategorientheorie umfasst; (3) der qualitative Spezifikationsformalismus *Constraint-Präferenzen*; sowie (4) Konzepte und Algorithmen zur Optimierung in Multiagenten-Systemen mit (möglicherweise antagonistischen) Präferenzen unter Verwendung von lexikographischen und kartesischen Produkten sowie sozialen Auswahlfunktionen.

# Acknowledgments

First of all, many thanks to my advisor Wolfgang Reif for supporting me and my work, for giving me enough freedom in research but not too much, and for pushing me in the right direction when I was about to get lost in theory.

I would like to thank Alexander Knapp for introducing me to the field of formal methods that proved to be a solid foundation for constraint programming as an interesting area of research. He supported all my proof endeavors, provided very helpful feedback on a draft of this dissertation, and acted as a mentor to me.

My external reviewer, Guido Tack from Monash University, provided very helpful and detailed feedback at the conceptual and even code level of some of the more intricate parts of the dissertation. Thank you for this invaluable help!

Thanks to the team of researchers and students at the Institute of Software & Systems Engineering in Augsburg for creating an encouraging and motivating working atmosphere – and a lot of cold-brewed coffee during the final months of writing. My office mate Oliver Kosak helped me a lot by patiently enduring fundamental discussions as well as by thoroughly proofreading chapters of this dissertation. Benedikt Eberhardinger provoked an essential part of MiniBrass by applying the tool to lunch selection problems at our annual retreat. My colleagues in the OC-Trust project, in particular, Gerrit Anders and Hella Ponsar often pointed out interesting viewpoints. Adrian, thank you for many coffees and bouldering sessions to lighten up the mood.

Thanks to my colleagues in the field soft constraints and constraint programming, especially to Ugo Montanari for pointing out interesting algebraic constructions and relevant papers, to Stefano Bistarelli for discussions about c-semirings at ICTAI'14, to Peter Stuckey for introducing me to MiniSearch at the poster session at CP'15, to Simon de Givry for providing help with Toulbar2, and to Guido Tack and his MiniZinc development team at Monash University. I hope to continue and enjoy these forms of cooperation.

I would like to thank my parents for always being supportive, and for advising without pushing. My mother, Gabriele, taught me the value of enjoying one's work, curiosity, and perseverance – indispensable ingredients of this dissertation which would not have been possible without her support.

Finally, many thanks go to my partner (in crime) Alexandra for her love, patience, and understanding during both the smooth and challenging times of writing this dissertation. You are still my favorite person to solve all kinds of everyday decision problems with me.

Alexander Schiendorfer

# Contents


1 Introduction 1
1.1 State of the Art and Statement of Purpose . . . 2
1.2 Scientific Contribution . . . 5

2 Application Scenarios 7
2.1 Distributed Energy Systems . . . 7
2.1.1 The Unit Commitment Problem . . . 8
2.1.2 Complexity of the Problem . . . 9
2.2 Self-organizing Robotic Systems . . . 10
2.2.1 Task and Resource Allocation in Reconfigurable Swarms . . . 10
2.2.2 Self-organizing Resource-Flow Systems . . . 11
2.3 Preference-oriented Systems . . . 12
2.3.1 Exam Appointment Scheduling . . . 12
2.3.2 Mentor Matching . . . 12
2.3.3 Multi-User Multi-Display Exhibitions . . . 13
2.4 Related Work . . . 13
2.5 Challenges for MiniBrass . . . 14

3 Preliminaries and Related Work 15
3.1 Classical Constraint Satisfaction and Optimization . . . 15
3.2 Over-Constrainedness . . . 16
3.2.1 Specific Soft Constraint Formalisms . . . 17
3.2.2 Algebraic Structures for Soft Constraints . . . 18
3.2.3 Soft Constraint Satisfaction Problems . . . 20
3.3 Algorithms to Solve (Soft) Constraint Problems . . . 21
3.3.1 Systematic Search . . . 22
3.3.2 Constraint Propagation and Global Constraints . . . 23
3.3.3 Local and Large-Neighborhood Search . . . 24
3.3.4 Existing Implementations . . . 25
3.4 Modeling Languages . . . 25
3.4.1 MiniZinc and MiniSearch . . . 26
3.4.2 Essence and Numberjack . . . 27

3.5 Related Work . . . . . 27

4 MiniBrass – A Soft Constraint Modeling Language **31**
4.1 A Hello-World Example . . . . . 33
4.2 PVS Types and Instantiations . . . . . 35
4.3 Examples of Soft Constraint Formalisms as PVS Types . . . . . 37
4.3.1 Integer-Valued: Weighted CSP or Cost Function Networks . . . . . 37
4.3.2 Comparative: The Free PVS and Constraint Preferences . . . . . 40
4.3.3 Real-Valued: Fuzzy CSP and Probabilistic CSP . . . . . 46
4.4 Morphisms to Switch PVS . . . . . 47
4.4.1 Products of PVS . . . . . 48
4.4.2 PVS-based Search . . . . . 49
4.5 Modeling Case Studies in MiniBrass . . . . . 51
4.5.1 Unit Commitment . . . . . 51
4.5.2 Mentor Matching . . . . . 54

5 Constraint Preferences for Soft Constraints **57**
5.1 Qualitative Specification . . . . . 59
5.1.1 Semantics of Dominance Properties . . . . . 59
5.1.2 Transforming Constraint Preferences to Weighted Constraints . . . . . 62
5.1.3 Concrete Weight Functions . . . . . 62
5.2 Illustrating Constraint Preferences: A Ski-Day Planner . . . . . 64
5.2.1 Personas & Preferences in the Ski-Day Example . . . . . 64
5.2.2 Changing Preferences . . . . . 66
5.2.3 Changing Constraints . . . . . 66
5.3 Constraint Preferences and Related Formalisms . . . . . 67
5.3.1 Reducing Constraint Hierarchies to Constraint Preferences . . . . . 67
5.3.2 Relationship with CP-Nets . . . . . 69
5.4 Solving "Constraint Preferences" Problems . . . . . 69
5.5 Evaluation . . . . . 71
5.5.1 Modeling Influence . . . . . 71
5.5.2 Algorithmic Efficiency . . . . . 72

6 Algebraic Structures for Soft Constraints **73**
6.1 Looking for *Free* Partial Valuation Structures . . . . . 74
6.2 The Free Monoid over a Set . . . . . 77
6.3 The Free PVS over a Partial Order . . . . . 79
6.3.1 The Free PVS as Single-Predecessors-Lifting . . . . . 79
6.3.2 The TPD-Lifting for Constraint Preferences . . . . . 82
6.4 The Free C-Semiring over a PVS . . . . . 85
6.5 Adequacy of Algebraic Structures . . . . . 90

7 Hierarchically Layered Soft Constraints **95**
7.1 Towards Lexicographic Products of PVS . . . . . 99
7.1.1 Collapsing Elements as an Obstacle . . . . . 99
7.1.2 The Lexicographic Product excludes Collapsing Elements . . . . . 100
7.2 Constraint Hierarchies as Products of PVS . . . . . 102

7.2.1 Locally Predicate Better . . . 103
7.2.2 Globally Better – Real-valued PVS . . . 104
7.3 A Mapping from the Maximum PVS to a $p$-Norm PVS . . . 105
7.4 Optima-Simulation . . . 107
7.4.1 Admissible Soft Constraint Problems . . . 108
7.4.2 Substituting PVS for Optimization: Optima-Simulation and Optima-Equivalence . . . 109
7.4.3 Optima-Simulating Max by Large $p$-Norm PVS . . . 111
7.4.4 Optima-Simulating Max with Finite Multisets . . . 112
7.5 Discussion: Applicability and Consequences . . . 115

8 Aggregating Soft Constraints by Voting **117**
8.1 Computational Social Choice . . . 118
8.1.1 Aggregating Preferences . . . 118
8.1.2 Restrictions imposed by Arrow's Theorem . . . 120
8.1.3 Soft Constraints and Voting – Related Work . . . 122
8.1.4 A Counterexample for Sequential Voting . . . 123
8.2 Voting in MiniBrass . . . 127
8.2.1 Approval Voting . . . 127
8.2.2 Condorcet Voting . . . 130

9 Evaluation **135**
9.1 Encoded Weighted CSP versus Native Toulbar2 . . . 138
9.2 Smyth-Optimization versus Weighted-Optimization . . . 138
9.3 Most Important First versus Default . . . 142

10 Conclusion and Outlook **145**
10.1 Achieved Contributions . . . 145
10.2 Outlook and Future Work . . . 146

Bibliography **147**

Chapter 1

# Introduction

**Summary.** This chapter provides the rationale and application scenarios that motivate and explain why a new modeling language for *soft* and hard constraints is required. In particular, this affects the domain of self-organizing or autonomous systems where most problems faced in practice are over-constrained. Also, this chapter highlights the contributions that the dissertation offers while providing an outline of its contents.

Autonomous, "intelligent", and self-organizing systems are beginning to permeate industry and everyday life alike. Examples range from adaptive production systems (e.g., [Audi, 2018], where autonomous carts replace conveyor belts and jobs are assigned by the system itself) to "smart" electricity producers and consumers that schedule operations according to a (renewable) energy availability (e.g., [LEW, 2018]) in so-called virtual power plants. Common motivations are increased efficiency and flexibility, more efficient use of shared resources in personal settings (e.g., scheduling of car availabilities for a shared car), and reduction of manual labor by designing higher levels of autonomy in those systems – also in terms of resilience against faults. Such endeavors require expertise from several domains.

To make informed decisions, besides extracting information from data (as witnessed by the recent uprise of machine learning), these systems have to solve computationally demanding combinatorial problems – depending on the application at hand: For instance, an adaptive production cell needs to find a new task allocation and resource flow in case some tool breaks (see Section 2.2.2) or a virtual power plant needs to schedule energy productions and consumptions of its member devices according to demand and weather forecasts (see Section 2.1), which involves selecting which plants to activate at all. Since there are many *discrete* choices to be made under (at least) hard constraints on valid combinations, the problems suffer from a combinatorial explosion and are (frequently) NP-complete, i.e., inherently hard to solve [Lenstra and Kan, 1979]. Still, we expect systems to come up with good allocations, schedules, or resource flows, in a reasonable amount of time and without considerable human intervention.

Fortunately, nowadays, there is a rich and general algorithmic toolkit implemented in various discrete optimization solvers – ranging from constraint solvers based on search and propagation to mathematical programming optimizers based on (non)-linear (integer) programming. These tools encapsulate decades worth of research and make them available for new problems. However, expressing a given problem as an instance that a particular solver understands requires considerable expertise and experience with several APIs and languages – a burden

especially for domain experts that are not well versed in modeling mathematical optimization problems. In recent years MiniZinc [Nethercote et al., 2007] emerged as a solver-independent, high-level constraint modeling language. MiniZinc enables its users to state decision variables, constraints among them, and a numeric objective function. Due to the higher level of abstraction, modelers are (ideally) not tied to a specific solving technology when formulating optimization problems. This becomes even more relevant if during the lifetime of a project, new constraints that change the applicable technology (e.g., nonlinear constraints are introduced to a problem that used to be solvable by mixed integer linear programming) are introduced.

However, many (perhaps most) industrial combinatorial optimization problems in practice tend to be **over-constrained** (see, e.g., [van Hoeve, 2011]). A most common remedy is to iteratively refine the initial constraint model by manually weakening or dropping constraints until a solution can be found. However, this approach does not work if the actual problem instance, i.e., all input parameters, is only available at runtime. But this is precisely the case when a system is intended to act autonomously as the aforementioned smart factories or smart grids. For instance, a constraint such as "do not exceed $x$ kWh in consumption" only makes sense after the available energy production is known. Simply failing with `unsatisfiable` is not an option here – instead, a compromise solution is necessary. To accomplish this, a constraint model should be written with the intention of *graceful degradation* in the first place. Some constraints have to be *softened* if necessary or even ignored such that we can, e.g., at least maximize the number of satisfied (soft) constraints.

This dissertation presents **MiniBrass**, the extension of MiniZinc to soft constraints that precisely addresses modeling such situations. Due to MiniBrass and MiniZinc being high-level modeling languages instead of low-level constraint solver implementations, they are more suitable for discussions with the previously mentioned domain experts that may state individual preferences. Generally speaking, user preferences are a second important driving force for a soft constraint modeling language. If simply stated as additional hard constraints, they inevitably lead to over-constrained situations. Due to the rise of technical systems interfering with personal spaces such as the aforementioned smart energy clients, user preferences have to be kept in the loop. Otherwise, e.g., switching on a washing machine during a toddler's nap time for economic or ecological reasons leads to crises that parents can hardly quantify.

## 1.1 State of the Art and Statement of Purpose

The motivating problems have attracted the attention of diverse research communities for decades (see Section 3.5), leading, inter alia, to a unified theory of *soft constraints* that subsumes over-constrained problems and preferences [Meseguer et al., 2006]. It offers a more general treatment of satisfaction (or violation) degrees of soft constraints as an ordered set accompanied by a combination operation (to combine the valuations of multiple soft constraints for an assignment) and dedicated top and bottom elements, i.e., an *algebraic structure*. Instead of working with well-known *specific* orderings, such as $(\mathbb{N}, \leq)$ or $(\mathbb{Q}, \leq)$, calculations and orderings are studied from an *abstract* algebra perspective to obtain more generality. The leading frameworks for so-called *preference structures* are *c(onstraint)-semirings* [Bistarelli et al., 1997] and (totally ordered) *valuation structures* [Schiex et al., 1995], i.e., ordered monoids. Moreover, by means of product operators such as a direct product (for Pareto-orderings) and a lexicographic product, complex valuation structures can be formed from elementary ones, allowing for modular specification and runtime combinations [Schiendorfer et al., 2015c; Gad-

ducci et al., 2013] (and subject of Chapter 7). Still, there were substantial theoretical and practical problems left open that MiniBrass addresses. In particular, ready-to-use implementations for recent constraint platforms are virtually absent – with Toulbar2 [Allouche et al., 2010], a solver designed for (integer) cost function networks as a specific valuation structure, being the exception to the rule. Therefore, the most important goal of this dissertation is to develop

> *A modeling language for specifying and solving soft constraint problems in a user-friendly, yet generic, extensible, and mathematically sound manner.*

Up to now, no such system or modeling language has been proposed, despite the theoretical efforts for unifying soft constraint formalisms. Additionally, some formalisms presented in this dissertation offer an intuitive visualization, e.g., a graph indicating the priority of soft constraints. In combination with Pareto and lexicographic combinations that can be arranged horizontally and vertically, respectively, we can cater to modelers' and end-users' needs by means of an understandable graphical notation – contrary to common objective expressions in mathematical optimization. To demonstrate the idea, we present a simplified example inspired by the case study involving virtual power plants (VPP) that we discuss in more detail in Section 2.1.[1] Each preference structure is displayed as a rectangle and represents a distinct goal in one "currency" where several soft constraints map to the preference structure's underlying set such as, e.g., costs in EUR. A preference structure's type defines how the individual soft constraints' valuations are combined (e.g., taking the sum or max, etc.).

**Example 1.1 – Unit commitment in a VPP as a soft constraint problem**
Consider a virtual power plant that consists of two members, an electric vehicle that primarily serves as a battery and a small biogas plant supplying power. For the electric vehicle, there are three boolean desirable properties that can be be partially ordered: there is a preferred minimal battery level threshold never to fall below ($\texttt{prefBL}_{\texttt{EV}}$); in the morning, the battery level should be higher than 70% ($\texttt{BLmorning}_{\texttt{EV}}$); within one day, the car should not be charged and discharged more than three times ($\texttt{limitBU}_{\texttt{EV}}$). *Qualitative* constraint preferences (see Chapter 5) are the right type for this preference structure (`EV`). The two more important ones can only dominate a single constraint, so "single-predecessor-dominance" (SPD) is the right choice (see Chapter 5 for a more in-depth explanation). The biogas' preferences are twofold: the overall cost, i.e., a cost function mapping to the *sum* of `costsMaintenance` and `costsFuel` are the first priority, second-tier preferences are boolean properties such that the gas tank should never exceed 90% in order to maintain strategic market flexibility ($\texttt{gasNotFull}_{\texttt{bio}}$), the plant should be set to an economical output value ($\texttt{ecoSweet}_{\texttt{bio}}$) and it should be switched on/off at most twice per day ($\texttt{onOff}_{\texttt{bio}}$). Having space in the gas tank is more important than the two other wishes combined, so "transitive-predecessors-dominance" (TPD) is used. A preference structure combining the biogas plant's economic and technical preferences is formed by taking a lexicographic product: $\texttt{biogas1} \ltimes \texttt{biogas2}$. The two members' structures are Pareto-combined to treat them equivalently in scheduling. However, both members' wishes are less important than the overall goals (`vppGoals`), reaching minimal deviation between supply and demand. Here, `deviation`[1] represents the deviation at time step 1 – it makes thus no sense to sum

[1] The case study will serve as a source of illustrative examples throughout the dissertation although MiniBrass is a general purpose soft constraint modeling language.

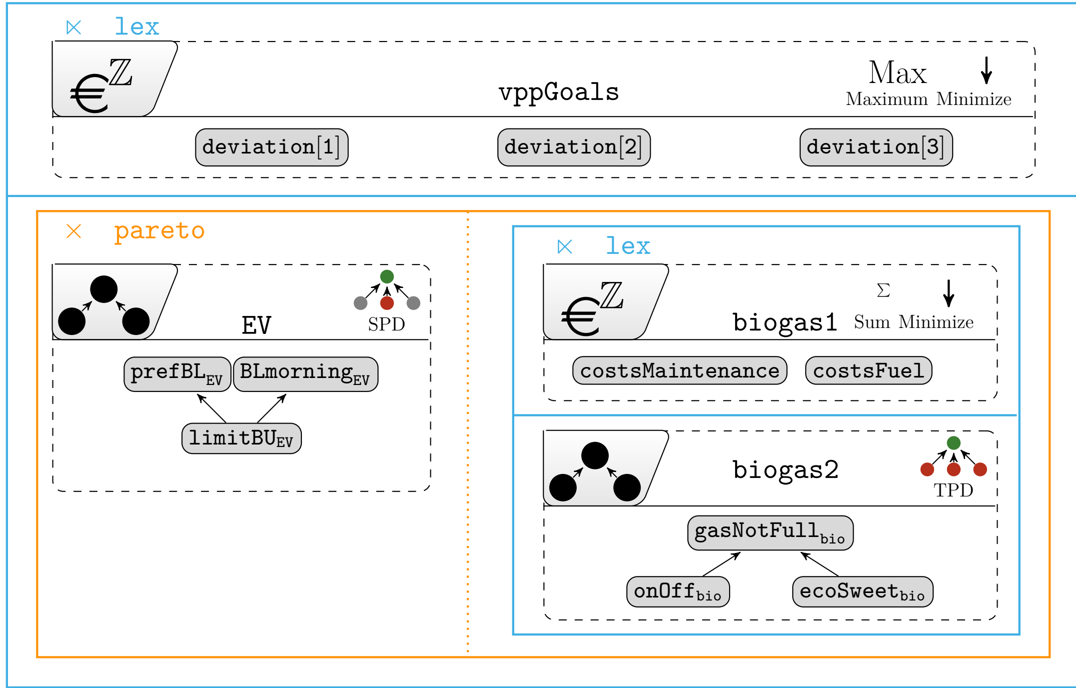



Figure 1.1: A graphical depiction of the overall complex preference structure for a simplified instance in the unit commit problem, as described in Example 1.1. Algebraically, the overall goal is `vppGoals` ⋉ (`EV` × (`biogas1` ⋉ `biogas2`)).

these values but rather have a look at the *worst* value which is why we use the maximum. Finally, the overall preference structure is given by `vppGoals` ⋉ (`EV` × (`biogas1` ⋉ `biogas2`))

In classical optimization and decision theory, specifying objectives amounts to defining a numeric function that maps variable assignments to some numeric codomain such as the rational numbers [Hansson, 1994]. The goal is to minimize or maximize the function's value. Multi-objective optimization extends this concept to multiple numeric objectives and most often optimizes according to a Pareto-front. By contrast, MiniBrass operates on a more abstract level, relying on partial orders and their combinations. This higher level of abstraction is particularly useful for situations that do not offer a clear-cut numeric objective such as, e.g., a makespan in scheduling or monetary costs. For example, then optimizing according to the set of satisfied or violated constraints with an appropriate set-based ordering can be an alternative. Example 1.1 showed this for he electric vehicle's preference structure. Inventing a numeric objective function that represents the modeler's ordering relation is then a cognitively demanding task in its own right – if more than just "maximize the number of satisfied soft constraints" is desired. If users fail to easily assign numeric values, the quantification should be distinct from the elicitation of preferences. Adequately modeling preferences then results in an exercise in "objective engineering". Encapsulating preferences into preference structures naturally leads to reuse and recombination possibilities (e.g., in case of changing priorities between structures).

## 1.2 Scientific Contribution

We discuss the foundations and applications to implement the vision outlined in Figure 1.1. More precisely, we reduce the gap between abstract soft constraint frameworks and available solvers by contributing:

- **MiniBrass**[2], a modeling language for soft constraint problems based on algebraic structures, that is compiled into MiniZinc [Nethercote et al., 2007] or MiniSearch [Rendl et al., 2015] code to inherit their support for a broad variety of solvers. Our language comes with an *extensible* type system for preference structures including common types in the literature (weighted constraints [Shapiro and Haralick, 1981], cost function networks [Allouche et al., 2012], fuzzy constraints [Ruttkay, 1994], probabilistic constraints [Fargier and Lang, 1993]) – see Chapter 4. We provide use cases and examples of existing soft constraint formalisms in MiniBrass, including structured types such as the free PVS or constraint preferences, combinations of PVS, and morphisms.
- The formalism *constraint preferences* originally presented by Schiendorfer et al. [2013]. It is a qualitative alternative to quantitative soft constraint formalisms using a numeric objective and only relies on a partial order over soft constraints. For instance, stating "$c_1 \to c_2$" indicates that soft constraint $c_1$ is less important to be satisfied than $c_2$. Chapter 5 presents several ways to lift such statements to sets of violated soft constraints based on dominance properties.
- A formal foundation for the semantics of the types implemented in MiniBrass which includes the systematic derivation of partially-ordered valuation structures from partial orders using category-theoretical arguments (see Chapter 6).[3] In the course of the derivation, we survey the adequacy of abstract frameworks in the literature with respect to model expressiveness and algorithmic efficiency – with an emphasis on expressiveness (see Section 6.5). Our results presented in Section 6.4 show how to extend any partially-ordered valuation structure to a c-semiring, if needed for a specific algorithm.
- In terms of theory, a completion and analysis of earlier attempts at expressing hierarchical soft constraints as lexicographic products of preference structures is subject of Chapter 7. The key observation is that a particular class of such constraint hierarchies give rise to problematic "collapsing elements" introduced by Gadducci et al. [2013]. These collapsing elements violate strict monotonicity, i.e., they make ordered, unequal elements equal upon multiplication. This can, in turn, violate the (weak) monotonicity for a product ordering. We provide the notion of "optima-simulation" to mitigate this restriction and provide two constructions along with necessary criteria. Moreover, we identify and implement suitable voting operators inspired by social choice theory in Chapter 8.
- An empirical evaluation using modified benchmark problems from the MiniZinc benchmark library that are supplemented with explicit soft constraints in different formalisms is offered in Chapter 9. We compare the solving performance of classical constraint solvers working on encoded soft constraint problems to that of a dedicated soft constraint solver (Toulbar2), the influence the used formalism has on solving time, and the efficiency of generic soft constraint search heuristics.

[2] MiniBrass pays tribute to the tradition of naming NICTA's G12 software after elements in the 12th group of the periodic table. *Brass* is an alloy containing *zinc* that is *softer* than zinc alone.

[3] In terms of language, the influence of category theory manifests also in the fact that morphisms (structure-preserving mappings between preference structures) are first-class citizens in MiniBrass.

Chapter **2**

# Application Scenarios

**Summary.** This chapter introduces the application scenarios that shaped the MiniBrass language. The main application scenario is a distributed energy system where individual consumers and producers have preferences regarding valid and high-quality schedules – besides the organizational goals. In addition, we present several other problems that benefit from a soft constraint modeling language. We conclude by tracing properties present in MiniBrass back to their origins in the application scenarios.

Constraints used to steer the decision making in autonomous systems—in their most general form—are found in various kinds. We first begin by motivating how distributed energy systems affect our considerations and then move on to other application scenarios. For each problem, we present a verbal specification in terms of decisions, (hard) constraints, objectives, and soft constraints/preferences.

## 2.1 Distributed Energy Systems

A fundamental problem of (especially electrical) energy systems is the lack of efficient storage capabilities. This requires that supply and demand of electrical power in a network are in balance in order to keep the AC grid's mains frequency close to a standardized value (50 Hz in continental Europe) [Heuck et al., 2010]. Future energy systems will likely transition from classical few-suppliers-to-many-consumers settings to distributed systems involving highly stochastic generation from weather-dependent producers, such as photovoltaics (PV) or wind power plants. This influx of renewable generation, in particular, PV, puts more strain on a grid's capabilities which became apparent in the so-called "50.2 Hz"-problem [von Appen et al., 2013]: The identical control strategies deployed simultaneously in various PV led to frequency oscillations as a high number of decentral units used a fixed cutoff-frequency of 50.2 Hz.

Problems like these motivate the vision of a "smart grid"(e.g., [Ramchurn et al., 2012; Dash et al., 2007; Miller et al., 2012]) that schedules power generation in a more proactive and coordinated fashion – despite uncertainties and the large scale. *Virtual power plants* (VPP) are a tool of paramount importance to accomplish this. But most visions even take a step further and include end-users into the process by implementing some form of demand-side-management. For instance, owners of an electric vehicle (EV) can offer their battery's capacity as storage to a utility in times of high power generation. This puts *end-users* and small plant

operators such as farmers in the loop. We use the term "prosumers" to collectively refer to energy producers and consumers. In most visions of this smart grid, prosumers partially lose their independence but have to stick to a coordinated and previously negotiated schedule in order to use resources most efficiently.

Due to the presence of end-users and small plant owners, not only technical properties matter in such systems. Owners of (small) power plants may know how they would like their system to be controlled – for instance, they would want to avoid frequent manual switching actions. Or, a consumer has to align the availability of their EV's battery with their personal schedules. Such preferences or wishes have to be submitted and considered in order to i) satisfy users' expectations and ii) keep users engaged at all. If they decide to withdraw consent for controlling their devices, a utility operator is faced with more uncertainties.

To express these wishes systematically, we need a rich variety of formalisms. This enables us to capture, e.g., rather boolean preferences such as "I never want to experience having a low battery when I need my car", rather metric preferences such as "the range of operation of my plant's output should be close to an optimal point of energy conversion efficiency", or even probabilistic goals such as "Make sure that the probability of me not reaching work is less than 5%". Moreover, devices in a VPP do not act in isolation but generate added value by coordination. Therefore, aggregation of individual and system goals is key.

Designing a scalable and trustworthy system accomplishing these aspects has been a major challenge in the OC-Trust project [Anders et al., 2016], including algorithms to partition and organize a system of prosumers hierarchically, predictive models to obtain more accurate schedules, and a decentralized, auction-based scheduling algorithm. For a comprehensive overview, see [Anders, 2017] and [Siefert, 2017]. For now, we will focus on one specific subproblem – the unit commitment or plant scheduling problem including individual preferences.

### 2.1.1 The Unit Commitment Problem

In order to balance supply and demand, the first step is usually to create a (cost/preference)-efficient schedule based on predictions for the environmental factors intermittent generation and demand. This problem is referred to as unit commitment since individual units commit to certain levels of generation beforehand [Padhy, 2004]. At runtime, contributions may be adjusted to better reflect the actual state of the environment.

**Example 2.1 – Scheduling in a VPP**
Figure 2.1 shows a simplified instance of the problem for clarity. For ease of presentation, we consider the (predicted) demand as given and are able to control the output of two power plants $a$ and $b$. Our goal is to match the summed output (the orange curve) to the demand (the blue curve). However, this has to be done under some adverse conditions: While the demand declines a little bit between the first and second time step, it is at a higher level at the third time step. Plant $a$ is in the middle of a ramp-up process (e.g., starting another turbine or other sources of inertia) so plant $b$ has to compensate by reducing its own output. This, in turn, prohibits that the demand is completely met in time step three since the ramp-up of plant $b$ is not fast enough after the previous compensation. Similarly, in time step four, the ramp down of plant $b$ is too slow to reduce the supply to the demand. Moreover, plant $a$ might need to wait for some time before further ramping-up, etc. These and other restrictions complicate the scheduling problem, as explained in [Schiendorfer et al., 2015a].

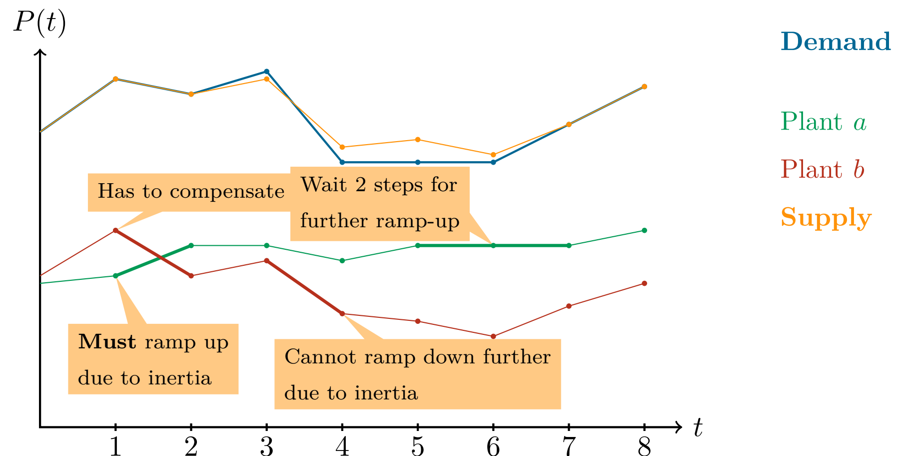



Figure 2.1: The problems faced in unit commitment demonstrated with example trajectories. Demand and supply of power must be aligned at all times $t$. Here, only plants $a$ and $b$ are controllable units, the demand is given.

Certainly, if we have controllable consumers at our disposal, their consumption can be considered as well. In times of low generation, loads can be shifted and vice versa. Then, personal preferences of the devices' owners become even more apparent. In conclusion, we can summarize the following problem aspects for future reference:

**Decisions** how much energy is produced/consumed at which point in time; when to switch on/off a power plant or schedule a consumption, i.e., a *schedule* for each prosumer in a VPP

**Constraints** limits on minimal and maximal production/consumption; start-up and cool-down times; battery capacities; tolerable discrepancies between supply and demand

**Objectives** minimizing the deviations between supply and demand; reducing operating costs

**Soft Constraints and Preferences** concerns about personal schedules: Having an electric vehicle ready at specific times; not having home appliances run at undesirable times; not having to manually switch on and off a biogas plant frequently.

### 2.1.2 Complexity of the Problem

The complexity of unit commitment depends on the level of detail that is modeled. In its simplest form, there are minimal and maximal bounds on scheduled supply and demand, as well as maximal changes per time step. Moreover, the outputs are modeled as real-valued decision variables. Since in that case, all constraints and the objective are linear functions of the decision variables, *linear programming* can be used to solve unit commitment [Piekutowski and Rose, 1985].

However, additional constraints stemming from more realistic models make the problem more complex to solve. Some power plants are not only subject to limited maximal but also

limited *minimal* capacities of production due to mostly technical but also economic reasons (e.g., a minimum generation of 20% of the nameplate capacity for gas turbines or 40% for coal based thermal plants [Hundt et al., 2009]). Moreover, for a virtual power plant, it is imperative to be able to switch plants on and off selectively to achieve more favorable aggregate partial load behavior compared to a single conventional generator [Karl, 2012] that suffers from increased costs when not operated at optimal energy conversion efficiency.

Consequently, having the binary option of switching on and off power plants changes the task at hand from a linear to a mixed integer linear program due to the involved 0/1 decision variables [Arroyo and Conejo, 2004]. Intuitively, now the solvers have to solve a knapsack problem of deciding which plants to active at all. Moreover, stochastic programming approaches that consider optimization over uncertain demands make the problem even more difficult to solve [Anders et al., 2013]. For all these cases, it is beneficial to have the support of a large variety of constraint solvers, including mathematical optimizers capable of solving mixed integer programs such as CPLEX [2013].

## 2.2 Self-organizing Robotic Systems

Another set of problems that inspire our research on soft constraints emerge from the class of self-organizing robotic systems [Hoffmann et al., 2011]. These systems are designed to be reconfigurable in order to prolong their lifetime of operation in case of hardware faults. Self-organization can take place on the individual level (i.e., changing components) as well as on the organizational level (i.e., reorganizing teams that cooperate).

### 2.2.1 Task and Resource Allocation in Reconfigurable Swarms

Robotics and embedded systems technology have made unmanned aerial vehicles (UAVs) readily applicable for both hobbyists and engineers. It is to be expected that they will serve important purposes in application areas that are otherwise hard to cope with. Consider for instance various inspection tasks such as those in agricultural practice (e.g., surveillance of cattle or fields) or scientific measurement missions. Many of these problems cannot be effectively solved by a single robot but require (optimally autonomous) cooperation of various robots to be more efficient in performing tasks, achieve higher fault-tolerance or even need the cooperation by the very nature of the problem (e.g., not all needed sensors can be placed on a single UAV due to weight restrictions, hence, multiple UAVs need to cooperate).

In terms of constraints, the mission plan might require that the estimated time of arrival (ETA) at a goal station for a UAV be earlier than the estimated time of battery outage (EBO):

$$\forall r \in \mathit{Robots} : r.\mathit{ETA} \leq r.\mathit{EBO} \tag{2.1}$$

But at the same time, one UAV $q$ might be more stable to control and be better suited for difficult terrain. It might thus be preferable to send $q$ to a difficult location than another robot, say $p$, even if $p$'s battery level would be higher and thus less likely to suffer an outage. We might state this preference regarding the mission plan.

Furthermore, Kosak et al. [2016] proposed a concept for reconfigurable ensembles of (e.g., flying) robots that are able to change their composition according to task requirements. Sensors or actuators are encapsulated into modular pieces of hardware that, enriched with semantic

information, become "semantic components" that can be attached dynamically to a UAV [Wanninger et al., 2018]. For instance, a task may indicate that it needs the capabilities "flying" and "measuring temperature" and a suitable reconfiguration makes sure that the semantic components attached to a UAV still preserve said capabilities, e.g., do not exceed the maximal take-off weight.
We can summarize the involved aspects as follows [Hanke et al., 2018]:

**Decisions** which semantic component is mapped to which hardware slot

**Constraints** maximal payloads; mutual geometric incompatibilities of UAVs and components

**Objectives** minimizing the number of reconfigurations required; minimizing the number of involved devices to reduce changeover times

**Soft Constraints and Preferences** knowledge about unreliable reconfigurations; "experience" factors such as avoiding sensor-pairings that influence each other

### 2.2.2 Self-organizing Resource-Flow Systems

Future production cells tend to be more tolerant towards changes in terms of the produced objects, as opposed to classical mass-production [Bickel and Schuster, 2005; Seebach, 2011]. Autonomous carts could transport workpieces between robot workstations, where tasks are performed on them. If a robot (or cart) fails, an existing task allocation and resource-flow has to be redesigned such that all tasks are applied to a workpiece in the correct order. Similar to the previous problem in reconfigurable swarms, an important requirement is that the tasks assigned to a robot are an actual subset of their capabilities:

$$\forall r \in \mathit{Robots} : r.\mathit{assignedTasks} \subseteq r.\mathit{capabilities} \tag{2.2}$$

Consider for instance that a robot might not have the right tooling mounted to perform certain tasks (e.g., drilling) or that tools break, rendering a previously perfectly fine allocation obsolete. Preferences in this scenario could lead to shorter distances traveled by the carts or differentiating among feasible allocation tasks based on tool status such that the time to the next reconfiguration is proactively minimized. Since the constraints and preferences of multiple robots have to be taken into account, we need mechanisms to combine them according to the organization structure. In summary:

**Decisions** what job is done by which robot;

**Constraints** valid resource flow including connections; tool capabilities for assigned jobs

**Objectives** maximizing the throughput; minimizing the number of reconfigurations required within a given time frame; minimizing the number of involved devices

**Soft Constraints and Preferences** precedence for certain carts; task allocation with respect to tool quality

## 2.3 Preference-oriented Systems

Besides these (socio-)technical systems, there are more mundane activities in the daily university routine that nevertheless strongly benefit from a soft constraint modeling language. In many systems, we expect adaptation according to users' preferences and wishes that are either learned or explicitly modeled. Often, not only a single user is involved in decision-making but several users each having their individual set of preferences.

### 2.3.1 Exam Appointment Scheduling

For oral exams, students are assigned appointments beforehand. In Augsburg (and perhaps many other universities), this is conventionally done using a first-come-first-served assignment on paper. Such a mechanism is however inefficient in the sense that conflicts with other exams or personal preferences are frequent simply due to the order in which students get to the list. There are, obviously, hard constraints determined by the available examiners and rooms. Other wishes should be respected as much as possible, in an unbiased way.

**Decisions** mapping students to appointments

**Constraints** maximal capacity per appointment; individual absences (e.g., other exams)

**Objectives** minimizing the number of needed exam days ("clustered" schedules); maximizing student satisfaction

**Soft Constraints and Preferences** having an early or late appointment within a week; no morning appointments; latest-possible, keep Friday free, etc.

### 2.3.2 Mentor Matching

The study program "software engineering" assigns students to company advisors (and/or professors) for mentoring purposes. This should be done for mutual benefit as to maximize the student's satisfaction with a possible internship as well as the company's benefit in terms of a productive, high-potential future employee. Again, this assignment is subject to hard constraints such as a minimal and maximal number of mentees per company in addition to the pure wishes of students and companies.

**Decisions** which student is assigned to which company

**Constraints** minimal and maximal numbers of students supervised by each company

**Objectives** satisfying students and/or companies; mutual preferences

**Soft Constraints and Preferences** company-perspective: matching knowledge and goals; student-perspective: demanding tasks, reputation, salary, etc.

A similar frequent problem is faced when assigning seminar topics or theses to students.

### 2.3.3 Multi-User Multi-Display Exhibitions

The OC-Trust project showed another example application where a set of displays is supposed to react to users' passing by showing appropriate contents [Schiendorfer et al., 2015b]. "Appropriate" refers to slides that are of interest to the users (based on a profile learned from previous contents they watched) and suitable in terms of prerequisite knowledge, i.e., there are dependencies between contents. The setting is targeted at exhibitions with multiple users interacting with multiple displays where compromise solutions are inevitable.

**Decisions** which content is displayed at which display

**Constraints** age restrictions; correct language; prerequisite knowledge

**Objectives** minimizing user disruption and discomfort

**Soft Constraints and Preferences** topical interests, preferred level of detail

## 2.4 Related Work

For most of the application scenarios, there exist specialized, bespoke algorithms that can solve a "clean" and well-structured version of the core problem. If we want to add specific side constraints emerging in practice to the formulations, these solutions often break down. Hence, we want to provide implementations in a rather high-level modeling language that is capable of delegating the search for optimal solutions to a variety of solvers.

The unit commit problem and its various facets have received considerable attention in the past due to its relevance in practice [Ramchurn et al., 2012]. Nieße et al. [2016] considered so-called *local* soft constraints for energy unit scheduling in VPPs. They included soft constraints such as avoiding frequent cold starts of combined heat and power plants into the distributed optimization heuristic COHDA [Hinrichs et al., 2013]. Their approach allows for a fuzzy-like specification of soft constraints but is rather domain-specific and tied closely to the overall COHDA system. Most approaches so far focused on cost-effective (monetary or environmental costs) scheduling in a distributed way [Dash et al., 2007; Miller et al., 2012] but ignored individual preferences.

Nafz et al. [2011] and Seebach et al. [2011] introduced the restore invariant approach for modeling self-organizing resource-flow systems based on a "corridor of correct behavior" which was described in more detail in [Seebach, 2011] and [Nafz, 2012]. Correct behavior is expressed by a conjunction of constraints on the system that always have to hold. If one constraint is violated, the system reconfigures, possibly by reusing the specified constraints for solving a constraint satisfaction problem [Nafz, 2012]. Steghöfer [2014] then proposed to extend this idea towards "soft corridors". MiniBrass has its roots in that line of research [Güdemann et al., 2006; Seebach et al., 2011].

Problems that involve matching, e.g., students and companies or students and their exam appointments are studied in matching theory [Lovász and Plummer, 2009]. Seminal results such as the Gale-Shapley algorithm [Gale and Shapley, 1962] provide solutions to the so-called stable "marriage" problem where we essentially search for pairings that are stable in the sense that no assigned couple would "run off", i.e., mutually prefer each other to their assigned partner. There are variants for 1:1 as well as for 1:n matchings. Both problems

can be solved in polynomial time. However, these variants address clean problem formulations without additional side constraints (such as, e.g., restricting a company's number of mentees or enforcing a minimal number of mentees per company). For such constraints, we want to benefit from a more general formulation that is amenable to constraint technology. Moreover, the addition of real-world constraints can easily transform the clean, polynomially solvable problem into an NP-complete one, as is the case with the hospitals/residents problem with couples where, e.g., couples can indicate that they want to be assigned to the same hospital [Gusfield and Irving, 1989]. In such cases, the efficient specialized algorithms are not valid anymore and it pays off to use a generic solution within a soft constraint framework.

## 2.5 Challenges for MiniBrass

Concluding, these and similar application scenarios pose challenges to conventional constraint frameworks which we need to consider during the design of MiniBrass:

**Heterogeneous Specification** Depending on the task at hand, a problem's objective might be better specified in terms of costs (or penalties), probabilities (probability of a solution to be actually useful), or satisfied constraints.

**Modularity** Since for some case studies, the set of participating agents (devices, etc.) is subject to change at runtime, individual preference structures have to be formed in a modular way such that we can form combinations dynamically.

**Compositionality** Organization structures require different forms of combination, such as a lexicographic one to give precedence to superiors and a direct one for peers' decisions in a Pareto-style.

**Extensibility** Since it is impossible to anticipate all formalisms needed to conveniently model a given problem, the framework needs to allow for easy extension within a suitable algebraic base type.

Besides these requirements, MiniBrass should be user-friendly, i.e., easy to understand and discuss, as this is one of the main criticisms we have for conventional soft constraint systems. This includes that the specified order for optimization should semantically be as transparent as possible, i.e., users should *know* what they actually optimize for. This idea permeates both the design of the language as well as our own formalism "constraint preferences" in Chapter 5.

CHAPTER 3

# Preliminaries and Related Work

**Summary.** This chapter introduces the reader to established concepts in constraint programming and optimization that are required for the following sessions. It also serves as a reference for the relevant notation. Although constraint preferences and partial valuation structures are discussed extensively in subsequent chapters, we already introduced them for the sake of understanding the MiniBrass language.

**Publication.** Constraint preferences and the overview on related work have been published in [Schiendorfer et al., 2013; Knapp et al., 2014; Schiendorfer et al., 2018].

We review our notation for conventional constraint satisfaction and optimization problems as well as soft constraint problems and then discuss common algebraic structures used for soft constraints and those underlying MiniBrass. Our definitions closely follow standards in the literature [Rossi et al., 2006] except for a functional style of defining constraints instead of a relational one, not necessarily restricting (mathematical) models to finite domains, and considering optimization over *partial* orders. Please be aware that, contrary to mathematical logic, constraint programming definitions tend to be made entirely in a (relational) semantic regime without considering the syntax of a formal language first. This can be traced back to the origins where constraint satisfaction problems were reduced to search problems on so-called constraint graphs [Dechter, 2003] that do not require any proof calculus.

## 3.1 Classical Constraint Satisfaction and Optimization

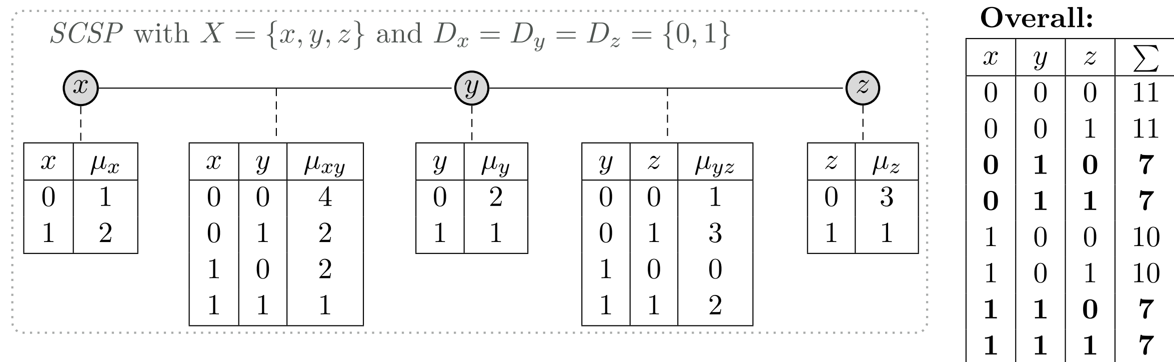
A *constraint (satisfaction) problem* $\mathit{CSP} = (X, D, C)$ is described by a set of (decision) variables $X$, their associated family of domains $D = (D_x)_{x \in X}$ of possible values, and a set of (hard) constraints $C$ that restrict valid assignments. For a $\mathit{CSP}$, an assignment $\theta$ over scope $X$ is a mapping from $X$ to $D$, written as $\theta \in [X \to D]$, such that each variable $x$ maps to a value in $D_x$. A (hard) constraint $c \in C$ is understood as a map $c : [X \to D] \to \mathbb{B}$ where we write $\theta \models c$ to express that $\theta$ satisfies $c$ (i.e., $c(\theta) = \mathit{true}$) and $\theta \not\models c$ to express that $\theta$ violates $c$. Each constraint has a scope $\mathit{sc}(c) \subseteq X$, i.e., the variables that actually influence its truth value. An assignment $\theta$ is a solution if $\theta \models c$ holds for all $c \in C$. In the literature, relational definitions view a hard constraint simply as a relation over its scope's variables' domains, i.e., $c \subseteq \prod_{x \in \mathit{sc}(c)} D_x$. The restriction of an assignment $\theta$ to a scope $X' \subseteq X$ is written as $\theta{\downarrow}X'$.

**Example 3.1 – A CSP for Robotic Task Assignment**
Consider a set of robots $R = \{r_1, r_2, r_3\}$ that have to solve tasks $T = \{drill, insert, tighten\}$ where no robot is allowed to perform more than one task ($c_{\mathsf{singleTask}}$) and robot $r_1$ can do *drill* and *insert* whereas robots $r_2$ and $r_3$ can do *insert* and *tighten*. We reduce this to a *CSP* by having a variable $x_t$ for each task $t \in T$. The respective domains are $D_{drill} = \{r_1\}$, $D_{insert} = \{r_1, r_2, r_3\}$, and $D_{tighten} = \{r_2, r_3\}$. The assignment $\theta_1 = \{x_{drill} \mapsto r_1, x_{insert} \mapsto r_2, x_{tighten} \mapsto r_3\}$ is a solution whereas $\theta_2 = \{x_{drill} \mapsto r_1, x_{insert} \mapsto r_2, x_{tighten} \mapsto r_2\}$ is not since $\theta_2 \not\models c_{\mathsf{singleTask}}$. Moreover, $sc(c_{\mathsf{singleTask}}) = \{x_t \mid t \in T\}$ and $\theta_2{\downarrow}\{x_{drill}\} = \{x_{drill} \mapsto r_1\}$.

We move from satisfaction to *constraint optimization problems* (*COP*) by adding an objective function $f : [X \to D] \to P$ where $(P, \leq_P)$ is a partial order, that is, $\leq_P$ is a reflexive, antisymmetric, and transitive relation over $P$. Elements of $P$ are interpreted as *solution degrees*, denoting quality. Without loss of generality, we interpret $m <_P n$ as solution degree $m$ being strictly *worse* than $n$ and restrict our attention to *maximization problems* regarding $P$. Hence, a solution degree $m$ is *optimal* with respect to a constraint optimization problem *COP* if for all solutions $\theta$ it holds either that $f(\theta) \leq_P m$ or $f(\theta) \parallel_P m$, expressing incomparability with respect to $\leq_P$. It is *reachable* if there exists a solution $\theta$ such that $f(\theta) = m$. Non-reachable optimal solution degrees appear, e.g., as upper bounds. Finally, a solution $\theta^*$ is called optimal if $f(\theta^*)$ is optimal.

## 3.2 Over-Constrainedness – From Correctness to Optimality

Pioneering attempts to generalize hard constraints were discussed in *partial constraint satisfaction* [Freuder and Wallace, 1992]: If no solution can be found, i.e., no assignment satisfies all constraints, a metric measures the distance of an assignment to the solution space of the original problem. Proposed distance choices include the number of required domain items to "patch" the assignment, the number of added tuples to a constraint's defining relation, or the number of *violated constraints*. The latter is better known as *Max-CSP*. We may note that a Max-CSP can be formulated as a *COP* by either counting the number of violated constraints, i.e. $f^{\text{int}}(\theta) = |\{c \mid \theta \not\models c\}|$, or by mapping each assignment directly to its violation set, i.e., $f^{\text{set}}(\theta) = \{c \mid \theta \not\models c\}$ (cf. [Bistarelli et al., 2004]). The former instantiates the natural ordering over natural numbers (i.e., $(P, \leq_P)$ is actually instantiated with $(\mathbb{N}, \geq)$) whereas the latter uses the standard inclusion relation over violation sets (i.e., $(P, \leq_P)$ becomes $(2^C, \supseteq)$). Certainly, the integer-based variant is conceptually straightforward whereas the set-based variant leaves more valuations incomparable. For instance, assume that we have $C = \{c_1, c_2, c_3\}$ and two assignments $\theta_1$ and $\theta_2$ with $\theta_1 \not\models c_1$ and $\theta_2 \not\models \{c_2, c_3\}$. Then $f^{\text{int}}(\theta_1) < f^{\text{int}}(\theta_2)$ but $f^{\text{set}}(\theta_1) \parallel f^{\text{set}}(\theta_2)$ since $\{c_1\} \not\subseteq \{c_2, c_3\}$. Using sets is beneficial if we care for strict improvement in terms of violating a strict subset of violated constraints and perhaps get *all* optimal solutions instead of merely looking for one with the *fewest* violations. This line of reasoning gives also rise to *constraint preferences* that we explore in more detail in Chapter 5 and briefly in the following section.




| $x$ | $\mu_x$ |
|---|---|
| 0 | 1 |
| 1 | 2 |

| $x$ | $y$ | $\mu_{xy}$ |
|---|---|---|
| 0 | 0 | 4 |
| 0 | 1 | 2 |
| 1 | 0 | 2 |
| 1 | 1 | 1 |

| $y$ | $\mu_y$ |
|---|---|
| 0 | 2 |
| 1 | 1 |

| $y$ | $z$ | $\mu_{yz}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 3 |
| 1 | 0 | 0 |
| 1 | 1 | 2 |

| $z$ | $\mu_z$ |
|---|---|
| 0 | 3 |
| 1 | 1 |

| $x$ | $y$ | $z$ | $\sum$ |
|---|---|---|---|
| 0 | 0 | 0 | 11 |
| 0 | 0 | 1 | 11 |
| **0** | **1** | **0** | **7** |
| **0** | **1** | **1** | **7** |
| 1 | 0 | 0 | 10 |
| 1 | 0 | 1 | 10 |
| **1** | **1** | **0** | **7** |
| **1** | **1** | **1** | **7** |

Figure 3.1: A toy example of a cost function network with soft constraints $\mu$. Note how cost functions do not classify assignments as "violated" or "satisfied" but map every (partial) assignment to an integer.

### 3.2.1 Specific Soft Constraint Formalisms

Partial constraint satisfaction, in particular Max-CSP obviously lacks any form of distinction between soft constraints regarding their importance. To introduce this distinction, various formalisms have been proposed. In weighted constraint problems (WCSP), each constraint has an assigned penalty value, to be "paid" if it is violated. Toulbar2 is a dedicated WCSP solver using search strategies (e.g., [Allouche et al., 2015; Sánchez et al., 2009]) and soft constraint propagation and filtering [Cooper et al., 2010], see Section 3.3. The notion of weighted CSP has been extended to the so-called "cost function networks" which replace constraints by cost functions that map an assignment to their scope's variables to some integer. The overall valuation is the sum of the individual valuations. Figure 3.1 illustrates how a cost function network is structured and evaluated. While the mathematical model of weighted CSP may use $\infty$ as a non-reachable maximal violation, existing implementations cap the natural numbers at some defined maximal value $k \in \mathbb{N}$ – which can then also be used to emulate hard constraints. Instead of the conventional summation, we write $+_k$ (defined as $a +_k b = \min\{a + b, k\}$) for the capped summation. Cost function networks may also use the maximum operator for aggregation instead of the sum.

Similar to WCSP, constraints can also be placed qualitatively: either in layers of importance, such as in *constraint hierarchies* proposed by Borning et al. [1992] or only comparatively, as done in *constraint preferences* [Schiendorfer et al., 2013] using a preference graph over constraints. In constraint hierarchies, any constraint on layer $i$ is more important than another on a layer $j > i$, i.e., the assignments are ordered lexicographically by their satisfaction degrees of the individual layers. The values for the layers themselves are found by choosing the desired aggregation function as a parameter: Borning et al. [1992] assumed that each constraint maps to a numeric value (0 or 1 for boolean soft constraints, or a metric error function). The proposed aggregation functions are (weighted) sums, (weighted) sum of squares, or taking the maximum of all constraints' values – in essence, different $p$-norms (1, 2, and $\infty$ in these examples, see Section 7.2.2). Constraint preferences aim to prioritize constraints in a less rigid fashion. Figure 3.2 illustrates the idea in a nutshell: Users specify an importance graph over soft constraints; we lift this ordering to sets of soft constraints and search for most satisfying solutions. In that example, not all three constraints can be satisfied simultaneously, e.g., $c_1$ forces that

either $n_2$ or $n_3$ take the night shift which conflicts with $c_2$ or $c_3$. There are solutions satisfying two out of three constraints. The graph depicts a partial ordering of the constraints with $c_1$ being most important and $c_2$ being incomparable to $c_3$. It is lifted to sets using the so-called *Smyth*-ordering which is defined according to the center part of the figure. The Smyth-ordering is also referred to as "single-predecessor-dominance" (SPD) since a single soft constraint can be traded for exactly one other. If one soft constraint is more important than a whole set of less important constraints, this lifting is called "transitive-predecessors-dominance" (TPD). The difference will be explored in more detail in Chapter 5, in particular in Figure 5.3. Petit et al. [2000] introduced the notion of meta-constraints to explicitly talk about constraints in other constraints, such as "B has to hold only if A is violated". Using meta-constraints which are now better known as "reified" constraints, constraint hierarchies and constraint preferences can be implemented, e.g., in MiniZinc. Solvers provide reified variants for several cost values, MiniBrass relies on that technique.

But there are other semantic variants of softened constraint formalisms in the literature: Instead of placing weights on constraints themselves, fuzzy constraints [Ruttkay, 1994] consider a constraint's defining relation as a fuzzy set with a membership function ranging from 0 to 1 declaring how strongly an assignment satisfies the constraint. The overall valuation is obtained by taking a fuzzy intersection value, i.e., typically the minimal membership value. Dually to fuzzy constraints where we maximize the minimal membership degree, possibilistic constraints [Schiex, 1992] assign a priority value to each constraint and aim at minimizing the maximal degree of all violated constraints. Another formalism suggested to interpret soft constraints probabilistically, leading to *probabilistic constraints* [Fargier and Lang, 1993]: For every soft constraint, we have a probability $p_i$ that the constraint is *actually* present. An assignment $\theta$ is judged by the probability of it being an "actual solution". For example, if the constraint that the energy demand in a virtual power plant exceeds 800 GW is only present in 10% of all cases, a schedule satisfying 790 GW might be just as fine in 90% of all cases. Assuming independence, we obtain the validity probability by multiplying $1-p_i$ for all violated soft constraints (i.e., all violated constraints have to be absent if $\theta$ still counts as a solution).

### 3.2.2 Algebraic Structures for Soft Constraints

This plethora of seemingly different soft constraint and preference formalisms shares many commonalities. Multiple soft constraints map to a value in an ordered set and their valuations are combined to an overall valuation which then serves as the ordering criterion. For example in weighted CSP, each soft constraint maps an assignment to either 0 or to its weight, all weight valuations are summed, and the sum value is used to rank the assignments in ascending order. With fuzzy constraints, every soft constraint maps to a real number that is combined by taking the minimum and assignments are ranked in descending order. These similarities have been recognized for many years, leading to a unified theory of *soft constraints* that subsumes over-constrained problems and preferences [Meseguer et al., 2006].

It offers a more general treatment of satisfaction (or violation) degrees as *algebraic structures*: an ordered set accompanied by a binary combination operation over that set and dedicated top and bottom elements. Instead of working with well-known *specific* orderings, such as $(\mathbb{N}, \leq)$, calculations and orderings are studied from an *abstract* algebra perspective. The leading frameworks are *c(onstraint)-semirings* [Bistarelli et al., 1997] and (totally ordered) *valuation structures* [Schiex et al., 1995], i.e., ordered monoids. These abstractions serve to both find general complexity-theoretic results and devise solving algorithms (search and inference) for

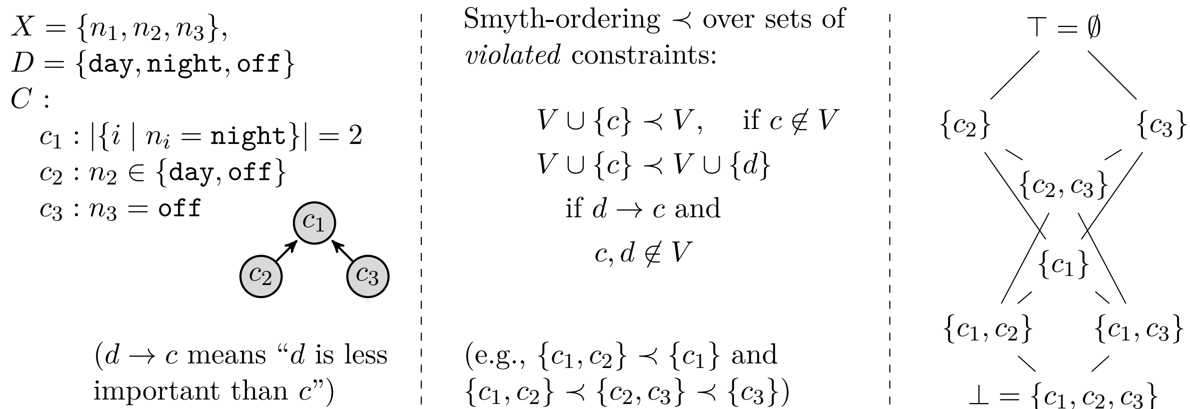



Figure 3.2: Left: A rostering problem involving three nurses $n_i$ with (comparative) constraint preferences. Center: An ordering over sets of violated constraints defined inductively by the two above rules (called the *Smyth*-ordering). Right: The Hasse diagram of $\prec$ over the valuation space: No violation ($\emptyset$) is best, and, e.g., $\{c_2, c_3\}$ is better than $\{c_1, c_2\}$ since it violates the *less important* constraint $c_3$ instead of $c_1$.

a broad class of problems. Moreover, product operators such as a direct product (for Pareto-orderings) and a lexicographic product allow for complex valuation structures composed of elementary ones. This enables *modular* specification and runtime combinations [Schiendorfer et al., 2015c; Gadducci et al., 2013], as we show in Example 1.1. For these purposes, Gadducci et al. [2013] introduced *partial valuation structures* (PVS) that relax the totality requirement and existence of a minimal element from valuation structures (see Definition 3.1). PVS only require that solution degrees obtained from soft constraints are combined using a binary operation, called multiplication, that there should be a neutral element representing *complete satisfaction*, and that combination should be monotone with respect to multiplication to denote that additional violation can only harm the quality further (a more elaborate example justifying monotonicity is presented in Chapter 7). See Definition 3.1 for its formal definition.

Valuation structures and c-semirings are obviously similar in nature and their relationship is well explored [Bistarelli et al., 1999]. Being purely algebraic, c-semirings prescribe an additive operation in addition to the combination operator in order to induce a partial order (i.e., $a \leq b \leftrightarrow a + b = b$) for *comparing* solutions. The additive operator acts as the supremum of this induced partial order, i.e., $a + b$ is the least upper bound of $a$ and $b$. Having a supremum operation available is particularly useful for variable elimination approaches, as we discuss in Section 6.5. However, many relevant partial orders do not admit a least upper bound such as, e.g., a Pareto-ordering of two orders or the Smyth-ordering presented in Figure 3.2.

Table 3.1 shows how the previously shown formalisms can be cast as partial valuation structures with appropriate mappings. For constraint preferences, finite multisets have to be used for a PVS-embedding although we showed the Smyth-ordering over (possibly infinite) *sets* in Figure 3.2. We present a detailed technical explanation for this in Section 6.1. For constraint hierarchies with their lexicographic elements, the situation is a bit more complicated and remained unclear for some time. Hosobe [2009] presented an encoding of some types of constraint hierarchies as c-semirings – except for a "worst case" semantics, i.e., the $\infty$-norm. Later, Schiendorfer et al. [2015c] provided a mathematical explanation for the impossibility of

Table 3.1: Different soft constraint formalisms presented as PVS. $C_s$ is the set of soft constraints, $\preceq$ is the Smyth-ordering on finite multisets over of soft constraints ($\mathcal{M}_{\text{fin}}(C_s)$) analogously to Figure 3.2.

| **Specific PVS types** | $M$ | $\cdot_M$ | $\leq_M$ | $\varepsilon_M$ | $\bot_M$ |
|---|---|---|---|---|---|
| Weighted CSP (WCSP) | $\mathbb{N}/\{0,\ldots,k\}$ | $+/+_k$ | $\geq$ | 0 | $\infty/k$ |
| Cost Function Network (CFN) | $\mathbb{N}/\{0,\ldots,k\}$ | $+/+_k/\max$ | $\geq$ | 0 | $\infty/k$ |
| Fuzzy CSP | $[0,1]$ | min | $\leq$ | 1 | 0 |
| Possibilistic CSP | $[0,1]$ | max | $\geq$ | 0 | 1 |
| Set-based Max CSP | $2^{C_s}$ | $\cup$ | $\supseteq$ | $\emptyset$ | $C_s$ |
| Constraint Preferences (CP) | $\mathcal{M}_{\text{fin}}(C_s)$ | $\uplus$ | $\preceq$ | $\{\!\!\}$ | – |

directly encoding this missing constraint hierarchies type by noting the presence of so-called "collapsing elements" (introduced by Gadducci et al. [2013]) that equalize distinct elements (i.e., $a < b$ but $c \cdot a = c \cdot b$) and are prohibitive for lexicographic products – as we cover in Chapter 7. Instead of directly encoding the worst-case semantics, the idea is to pick a sufficiently high $p$ such that $\|\cdot\|_p \approx \|\cdot\|_\infty$ for all relevant cases such that the optimization problem behaves similarly with respect to its optima in the non-collapsing $p$-norm and the collapsing $\infty$-norm. We revisit these constructions in more detail in Chapter 7.

### 3.2.3 Soft Constraint Satisfaction Problems

Due to their generality in capturing a wide range of specific formalisms as well as their modularity in terms of direct and lexicographic products, we set PVS as the default algebraic structure in the following. Moreover, total valuation structures are obviously instances of PVS, any c-semiring gives rise to a PVS, and, conversely, any PVS can be "augmented" to a c-semiring via a free construction (see Section 6.4). Consequently, a *soft constraint satisfaction problem* (SCSP) is defined as a COP where i) the objective is decomposable into multiple objectives (i.e., soft constraints) defined on their respective scopes and ii) the codomain of the objective is the underlying set of a partial valuation structure, formally defined below:

**Definition 3.1 – PVS**
A PVS $(M, \cdot_M, \varepsilon_M, \leq_M)$ is a *partially-ordered commutative monoid* where the multiplication $\cdot_M$ is monotone w.r.t. the partial ordering $\leq_M$ and $\varepsilon_M \in M$ is both the neutral element w.r.t. $\cdot_M$ and the top element w.r.t. $\leq_M$. That is, $(M, \leq_M)$ is a partial order and the following axioms hold for all $m, n, o \in M$: (1) $m \cdot_M n = n \cdot_M m$; (2) $m \cdot_M (n \cdot_M o) = (m \cdot_M n) \cdot o$; (3) $m \cdot_M \varepsilon_M = m$; (4) $m \leq_M \varepsilon_M$; (5) $m \leq_M n \rightarrow m \cdot_M o \leq_M n \cdot_M o$.

A PVS $M$ is *bounded* if there also exists a minimal element $\bot \in M$ to represent complete dissatisfaction. Hence, a total valuation structure as used by Schiex et al. [1995] is a bounded PVS where $\leq_M$ is a total ordering. A bounded PVS is further *weakly strict* if $m <_M n$ implies $m \cdot_M o <_M n \cdot_M o$ for all $m, n \in M$ and $\bot_M \neq o \in M$. Boundedness as a requirement does not reduce the generality of PVS operators since each PVS $M$ can be *augmented* with a "fresh" element $\bot \notin M$ to a bounded PVS $M_\bot = (M \cup \{\bot\}, \cdot_{M_\bot}, \varepsilon_{M_\bot}, \leq_{M_\bot})$ and setting $m \cdot_{M_\bot} \bot = \bot$, $\varepsilon_{M_\bot} = \varepsilon_M$, and $\bot \leq_{M_\bot} m$ for all $m \in M \cup \{\bot\}$.

If $M$ and $N$ are PVS, so are $M \times N$, the direct (Cartesian) product, and $M \ltimes N$, the lexicographic product – as long as some conditions on $M$ hold that will be subject of Chapter 7. Both products have pairs of elements of the underlying sets of $M$ and $N$ as their elements and combination is applied component-wise. For two PVS $M$ and $N$, we can construct the ordering of the direct product as follows:

$$(m, n) \leq_{M \times N} (m', n') \leftrightarrow m \leq_M m' \wedge n \leq_N n'$$

which is a Pareto-ordering over the underlying orderings of $M$ and $N$. The Pareto-ordering leads to a "fair" but not decisive aggregation of several PVS. A satisfaction degree (i.e., a pair of underlying satisfaction degrees) is only better than another one if both PVS approve of or prefer it. By contrast, the ordering of the lexicographic product is defined as

$$(m, n) \leq_{M \ltimes N} (m', n') \leftrightarrow (m <_M m') \vee (m = m' \wedge n \leq_N n')$$

It allows us to express hierarchical relationships between PVS to distinguish, e.g., organizational from individual goals.

Furthermore, to allow for structure-preserving mappings between PVS, we define a PVS-homomorphism from a PVS $(M, \cdot_M, \varepsilon_M, \leq_M)$ to a PVS $(N, \cdot_N, \varepsilon_N, \leq_N)$ as a mapping $\varphi : M \to N$ such that $\varphi(\varepsilon_M) = \varepsilon_N$, $\varphi(m \cdot_M n) = \varphi(m) \cdot_N \varphi(n)$, and $m \leq_M n \to \varphi(m) \leq_N \varphi(n)$ (order-preservation). Finally, we define a soft constraint $\mu$ over a PVS $M$ as a map $\mu : [X \to D] \to M$, we denote the set of soft constraints by $C_\mathrm{s}$ and write an *SCSP* as $(X, D, C, (M, \cdot_M, \varepsilon_M, \leq_M), C_\mathrm{s})$ which can be seen as a *COP* $(X, D, C, (M, \leq_M), f)$ where

$$f(\theta) = \Pi_M\{\mu(\theta) \mid \mu \in C_\mathrm{s}\} \tag{3.1}$$

using $\cdot_M$ to aggregate solution degrees of all soft constraints evaluated on an assignment.

**Example 3.2 – Rostering**
Consider again the rostering problem in Figure 3.2 (Left) and let $(X, D)$ be as depicted and use $\mathsf{U} = (\{c_1, c_2, c_3\}, \leq_\mathsf{U})$ with $\leq_\mathsf{U} = \{(c_2, c_1), (c_3, c_1)\}^*$ as a partial order denoting urgency of constraints. For $C = \{c_1, c_2, c_3\}$ as hard constraints, the solution space is empty. Instead, we can convert each hard constraint $c_i$ into a soft constraint $\mu_i$ by choosing a suitable PVS $M$. For instance, we could use the PVS $(\mathbb{N}, +, 0, \geq)$ and interpret each valuation as a penalty incurred for a violated soft constraint. The sum of penalties ought to be minimized. With weights $\vec{w} = [2, 1, 1]$, we define $\mu_i(\theta) = w_i$ if $\theta \not\models c_i$ and $\mu_i(\theta) = 0$ otherwise. Letting $C = \emptyset$ and $C_\mathrm{s} = \{\mu_1, \mu_2, \mu_3\}$, the solution $\theta = \{n_1 \mapsto \texttt{night}, n_2 \mapsto \texttt{night}, n_3 \mapsto \texttt{off}\}$ is optimal with $f(\theta) = \sum_{\mu_i \in C_\mathrm{s}} \mu_i(\theta) = 1$. The solution degree 0, being top in $M$, is not reachable.

## 3.3 Algorithms to Solve (Soft) Constraint Problems

For solving soft constraint problems in particular, many approaches are borrowed from traditional constraint solving – either the soft constraint (optimization) problem is split into a sequence of conventional constraint problems with ever-tightening constraints on the objectives or the search and inference techniques that make classical solvers successful have been generalized to the soft constraint case [Meseguer et al., 2006]. Therefore, we briefly survey the existing

techniques to keep the dissertation self-contained but refer to introductory literature [Marriott and Stuckey, 1998; Rossi et al., 2006].

Algorithms that calculate assignments satisfying all hard constraints are called (constraint) *solvers*. A solver is *complete* if, given enough time, it is guaranteed to find a solution if there is one. Most problems considered in constraint reasoning are finite such that at least there exists a naïve, enumerative algorithm which is complete. Since the search space still grows exponentially in the domain sizes of the involved variables, there are also many *incomplete* solvers (heuristics) which tend to find solutions in a reasonable amount of time in practice but do not offer any guarantees about finding a solution at all.

Both classes of solvers essentially follow a "search-and-inference" paradigm where the search part leads to exploring possible solutions by trial & error and inference denotes that some facts (e.g., "$y \neq 5$") about the problem are inferred from other, perhaps previously made, decisions (e.g., "$x \leftarrow 5$"). We begin by exploring systematic search, then go on to describe inference (called "constraint propagation"), and conclude with hybrid methods.

For soft constraints in particular, generalized variants of branch-and-bound, soft arc consistency [Cooper et al., 2010], and non-serial dynamic programming techniques [Bertele and Brioschi, 1973] (such as bucket elimination or cluster tree elimination [Dechter, 2003]) use the algebraic structures presented in the previous section. In addition, some global constraints (including `alldifferent` and `gcc`) with dedicated propagators have been generalized to a soft variant by van Hoeve [2011], usually by considering one (integer) cost variable that measures violation (see Section 3.3.2).

### 3.3.1 Systematic Search

As mentioned before, due to the typical finiteness of constraint problems, it is, in principle, always possible to traverse the entire search space in a systematic way. The most elementary form of this approach is called backtracking [Wells, 1971] where only consistent partial assignments are extended with values for unassigned variables until either a solution is found or a previous assignment has to be undone. That way, a so-called search tree is constructed on the fly (see Figure 3.3) and depth-first-search on the tree leads to full assignments, i.e., solutions. Besides backtracking, there are also "backjumping" strategies that do not simply undo the last operations in chronological order but identify so-called "culprit variables" that caused the emerging consistency conflict [Dechter and Frost, 2002]. The solvers used for our implementation and evaluation do rely on conventional, chronological backtracking, however.

For optimization problems, enumerative search is often accompanied by appropriate relaxations, leading to algorithms similar to branch-and-bound. The idea is to exploit a bounding function on the objective $f$. It estimates the "best possible" value $\hat{f}(\theta)$ that can be achieved, given the current partial assignment $\theta$. If, even in the best case, the search at a given node cannot improve upon an already found value, we can prune at that point. Figure 3.3 illustrates branch-and-bound for the one-dimensional knapsack problem. The formulation has $\{0,1\}$-variables $x_i$ to represent that item $i$ should be taken into the knapsack iff $x_i = 1$. As usual, the objective is to maximize the summed values of all items in the knapsack. A simple bounding function for any partial assignment assumes that all items (that remain to be decided) can be placed in the knapsack – in their entirety. The idea of branch-and-and-bound is similar to branch-and-cut for *mixed integer programming* (MIP) which are optimization problems over real-valued decision problems involving only linear constraints, a linear objective function, and integrality constraints on some (or all) of the decision variables. The integrality constraint on

$$\begin{aligned} \underset{x_1,x_2,x_3}{\text{maximize}} \quad & 45 \cdot x_1 + 48 \cdot x_2 + 35 \cdot x_3 \\ \text{subject to} \quad & 5 \cdot x_1 + 8 \cdot x_2 + 3 \cdot x_3 \leq 10 \end{aligned}$$

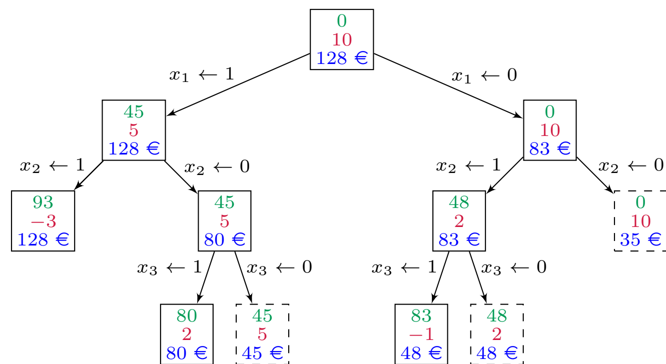



Figure 3.3: Top: A one-dimensional knapsack problem for illustration, taken from the online course described in [van Hentenryck and Coffrin, 2014]. Bottom: A possible search tree traversing the search space. At every search node, the first line indicates the already achieved value, the second line contains the remaining weight budget, and the third line has the upper bounding function $\hat{f}$ that indicates the optimally achievable summed value of the knapsack.

integer decision variables is relaxed and bounds are obtained by solving the resulting linear programs (LP) which can be done much more efficiently using, e.g., the simplex algorithm. It is worth noting that, despite the similarities of branch-and-bound and branch-and-cut, most solvers are developed for one class of problems or the other. Certainly though, there are significant efforts to integrate both technologies [Hooker, 2007].

### 3.3.2 Constraint Propagation and Global Constraints

Search alone is often too expensive for problems of even moderate size. Therefore a key ingredient to most constraint programming systems is to view constraints as procedures that can act upon a so-called domain store that holds possible items for all of the unassigned variables [Bockmayr and Hooker, 2005]. The goal is domain filtering, i.e., removing all values from domains that can never lead to feasible solutions. Since domain reductions can trigger additional domain reductions, this class of algorithms (which is prominent in constraint programming) is called *constraint propagation* [Bessiere, 2006] since information is propagated through constraints and a shared domain store.

**Example 3.3 – Simple Constraint Propagation**
Assume a constraint problem with $X = \{x, y\}$, $D_x = D_y = \{1, \ldots, 10\}$, and a single constraint in $C = \{``|x - y| > 5"\}$. We can deduce that $x$ and $y$ cannot take 5 or 6 as their value and update the domains accordingly: $D'_x = D'_y = \{1, \ldots, 10\} \setminus \{5, 6\}$.

Constraint propagation algorithms implement deductions such as those in the above example systematically. The first filtering algorithms were generic and graph-theoretically moti-

vated, i.e., operated on arbitrary finite relations to achieve *node consistency* (no domain value violates a unary constraint), *arc consistency* (in a binary relation, there is a "partner" for every remaining domain item), and *path consistency* (generalizing arc consistency to paths of length $k > 2$) [Mohr and Henderson, 1986]. For large domains though, these generic filtering algorithms suffer from the little information that is conveyed with a purely extensional representation. Therefore, so-called *global constraints* encapsulate more specific filtering algorithms for frequently recurring patterns in constraint models [van Hoeve and Katriel, 2006]. A dedicated filtering algorithm for a (global) constraint is also called a *propagator*. Exploiting a constraint's particular semantics, these propagators can achieve more efficient filtering in less time. The most prominent example is the `alldifferent` constraint which states that some variables $x_1 \ldots x_n$ should each take a different value which happens frequently if we have to decide some injective function in our models (e.g., assigning a different task to each worker). Régin [1994] presented the first dedicated filtering algorithm for `alldifferent` which is based on matching theory for bipartite graphs. The *global constraint catalogue* [Beldiceanu et al., 2007] lists many more global constraints for recurring problems such as cardinality constraints (for imposing limits such as "only five variables may take value $d$"), ordering constraints (the sequence $x_1 \ldots x_n$ must contain values in ascending order), or extensional representations of relations with a table constraint or a deterministic finite automaton (called `regular`) – each with specific propagators that existing solvers provide. The ideas of propagation and arc consistency have been extended to the case of soft constraints where a cost variable takes the measure of violation. There is, e.g., a soft `alldifferent` constraint that does not fully enforce `alldifferent` but has an additional variable take the number of equally assigned variables as a measure of violation [van Hoeve, 2011]. For instance, for variables $[x_1, x_2, x_3]$ and cost variable $\mu$, we have that **softalldifferent**$(\{x_1 \leftarrow 1, x_2 \leftarrow 2, x_3 \leftarrow 2, \mu \leftarrow 2\})$ or **softalldifferent**$(\{x_1 \leftarrow 2, x_2 \leftarrow 2, x_3 \leftarrow 2, \mu \leftarrow 3\})$ holds (see Section 4.3.1).

Most current constraint programming solvers blend search with domain filtering and constraint propagation to obtain good performance. For example, the order of propagators executed for constraints can drastically affect the runtime [Schulte and Carlsson, 2006]. Also, there can be several filtering algorithms for a constraint, each reaching different levels of consistency at different runtimes and a solver has to decide when to schedule which propagator. Constraint propagation is also imperative in steering the search in such systems since branching can be interpreted as augmenting an existing problem with a new constraint (e.g., "$x = 1$" or "$x \neq 1$"). Global constraints and efficient propagation are similarly important for *optimization* in constraint programming systems since they enable the fast solution of a sequence of satisfaction problems. The "getBetter" predicates that are used for search in MiniBrass rely on this principle (see Section 4.4.2).

### 3.3.3 Local and Large-Neighborhood Search

For many large-scale problems, despite efficient propagators and search, systematic enumeration can simply be too time and space consuming. In such cases, local search algorithms constitute incomplete solvers that may still offer a practical benefit, i.e., come up with solutions in less time. In contrast to the systematic search methods such as branch-and-bound which incrementally build up assignments in a search tree until a solution is found, local search algorithms start with a random initial assignment and try to improve or repair it until a valid (or, the best) solution is found [Hoos and Tsang, 2006]. They are called local since there is a neighborhood function $\mathcal{N}$ that maps each assignment $\theta$ to its set of neighboring assignments

$\mathcal{N}(\theta) \subseteq [X \to D]$. For instance, the neighborhood of an assignment $\theta$ can be the set of assignments $\theta'$ such that only one variable's assignment differs or that two variables swap their value. Local search methods come in different flavors and include hill-climbing, simulated annealing, or tabu search. A local search algorithm then typically proceeds as follows:

1. Find an initial assignment $\theta$.
2. Explore the neighborhood $\mathcal{N}(\theta)$.
3. Search for the "best" $\theta' \in \mathcal{N}(\theta)$ (fewer violated constraints, better objective, ...).
4. Repeat with $\theta'$ and continue these steps until convergence (solution found, (local) optimum found, timeout, ...).

Local search has proved to be successful for a variety of problems, beginning with the min-conflicts heuristic that enabled solving very large $N$-queens instances (up to $N = 10^6$ in [Minton et al., 1992]). Most often it is a starting point to tackle constraint optimization problems without using a constraint solver at all. However, due to the locality, the algorithm may suffer from getting stuck in local optima. In addition to breakout strategies such as the aforementioned simulated annealing, more recently, the benefits of systematic, propagation-powered solving and local search have been combined to create *large neighborhood search* (LNS, see [Shaw, 1998; Pisinger and Ropke, 2010]). The idea is to be able to explore a very large neighborhood of a given assignment by means of constraint propagation and search within a smaller subproblem than the overall one. During an iteration, the current solution $\theta$ is, e.g., constrained to have some variables fixed whereas others can take new values. The proportion of variables that can be reassigned and optimized defines the size of the neighborhood and has to be tailored to the specific problem instance at hand.

### 3.3.4 Existing Implementations

As previously stated, for soft constraint systems we noted the lack of solvers for general semiring or valued constraints. Most papers offer closed ad-hoc implementations focusing on one particular type such as [Rossi and Pilan, 2003] or [Bistarelli et al., 2003] for fuzzy CSP. By contrast, Leenen et al. [2007] provides a formulation of c-semiring-based soft constraint problems as "weighted semiring Max-SAT" that uses the semiring values and ordering as "weights". The encoded problems are solved with basic implementations of branch-and-bound and GSAT, outperforming the fuzzy solver CONFLEX (which is not available anymore). However, these algorithms do not rely on the supremum operator of a c-semiring and could be run as well with partial valuation structures (see Section 6.5). In addition, the approach remained rather prototypical (random instances with up to 120 variables and 20 constraints), only supported strict domination search for partially-ordered search spaces (see Section 4.4.2), and did not offer a public API to their system – which brings us to modeling languages.

## 3.4 Modeling Languages

Most existing constraint solvers offer an API to model constraint problems in imperative code. For higher layers of abstraction, there have been several proposals for domain-specific languages that allow users to express constraint models more declaratively. We begin by reviewing MiniZinc, the foundation of MiniBrass, as the most widespread constraint modeling language which gives rise to the annual MiniZinc challenge and then go on to describe other approaches.

All of these languages and platforms have in common that they strive at facilitating constraint modeling for domain experts and not necessarily constraint programmers alone.

### 3.4.1 MiniZinc and MiniSearch

MiniBrass is built on top of MiniZinc which itself is a subset of the Zinc language. MiniZinc is a high-level constraint modeling language that is compiled to the flat file format FlatZinc [Becket, 2014] which is understood by many constraint, MIP, or SAT solvers[1]. MiniZinc supports the primitive data types `bool`, `int`, `float`, and subtypes of those in addition to arrays and sets of them.

```
array[1..3] of var 1..3: x;                   % decision variables
constraint forall (i in 1..2) (x[i] <= x[i+1]);  % constraints

solve satisfy;                                % objectives (satisfy/minimize numExpr/maximize numExpr)
```

FlatZinc only consists of variable definitions and relational constraints. This process, compiling MiniZinc to FlatZinc, is also called *flattening* [Nethercote, 2014]. For instance, the above high-level model gets flattened to the FlatZinc file

```
array [1..2] of int: X_INTRODUCED_3 = [1,-1];
var 1..3: X_INTRODUCED_0;
var 1..3: X_INTRODUCED_1;
var 1..3: X_INTRODUCED_2;
array [1..3] of var int: x:: output_array([1..3]) = [X_INTRODUCED_0,X_INTRODUCED_1,X_INTRODUCED_2]
constraint int_lin_le(X_INTRODUCED_3,[X_INTRODUCED_0,X_INTRODUCED_1],0);
constraint int_lin_le(X_INTRODUCED_3,[X_INTRODUCED_1,X_INTRODUCED_2],0);
solve satisfy;
```

where we can see that the `forall`-loop has been unrolled to state $\texttt{variables}[1] \leq \texttt{variables}[2]$ and $\texttt{variables}[2] \leq \texttt{variables}[3]$ (represented by the low-level binary `int_lin_le` FlatZinc constraint) explicitly. MiniZinc strongly supports and encourages global constraints for modeling (see Section 3.3.2). On the one hand, this helps to keep models concise and makes modelers' intentions more obvious. On the other hand, solvers may handle global constraints completely differently – depending on the backing technology (e.g., constraint programming, SAT, or MIP, etc.) as well as the supported range of global constraints. We depend on this mechanism in our definitions of *soft global constraints* as well as *native cost functions* for Toulbar2 (see Section 4.3.1) which is why we explain the mechanism in more detail. We could rewrite the above example using the global constraint `increasing`:

```
array[1..3] of var 1..3: x;

include "increasing.mzn";
constraint increasing(x);
solve satisfy;
```

For a solver that understands `increasing` (e.g., Gecode by Schulte et al. [2006]), the FlatZinc output will be similar to

```
predicate increasing_int(array [int] of var int: x);
var 1..3: X_INTRODUCED_0;
var 1..3: X_INTRODUCED_1;
var 1..3: X_INTRODUCED_2;
array [1..3] of var int: x:: output_array([1..3]) = [X_INTRODUCED_0,X_INTRODUCED_1,X_INTRODUCED_2];
constraint increasing_int(x);
solve satisfy;
```

[1] At the time of writing, 18 solvers are listed on the MiniZinc website at `www.minizinc.org/software.html`.

where we see that `increasing` has merely been replaced by the predicate `increasing_int` that MiniZinc expects the FlatZinc parser of Gecode to understand and treat properly. A very elegant property of MiniZinc is that there are default decompositions for solvers that do not offer a dedicated implementation of, e.g., `increasing`. Those solvers, such as e.g. JaCoP by Kuchcinski and Szymanek [2013], will get a FlatZinc output identical to the above first FlatZinc output since the MiniZinc standard library defines `increasing` almost exactly as seen in the first listing. Similarly, the decompositions for MIP or SAT solvers amount to efficient encodings in the respective formalisms. The process of substituting solver-specific implementations or default decompositions of constraints is hidden from users who only specify the constraint model using MiniZinc's global constraints library.

There are variations and extensions such as stochastic MiniZinc [Rendl et al., 2014] for problems involving uncertainties, MiningZinc for constraint-based data mining [Guns et al., 2017], MiniSearch [Rendl et al., 2015] for customizable search, and extensions for large neighborhood search [Dekker et al., 2018]. MiniSearch provides facilities to access a search tree at the solution level, making queries such as "fetch the next solution; when found, constrain the next solution to have to improve" (in terms of, e.g., some partial order) – effectively resulting in a form of propagation-based branch-and-bound. For abstract soft constraint models, we found this to be the right level of granularity – as opposed to a fine-tuned programmable search since we can only rely on the existence of an ordering predicate and a way to combine individual valuations. Moreover, with MiniSearch, a search strategy has to be defined just once and can be used by any FlatZinc solver instead of implementing custom search for each solver. MiniSearch does so by generating multiple FlatZinc files. Additionally, there is native search tree interaction for Gecode.

### 3.4.2 Essence and Numberjack

Other constraint modeling languages include Essence [Frisch et al., 2008] or OPL [van Hentenryck, 1999]. While due to the existence of OPL script, OPL would be suited for a soft constraint modeling language such as MiniBrass as well, Essence does not offer search combinators or programmable search. We could only work with repeated solver calls or numeric (integer) objectives – effectively recreating the facilities that MiniSearch already offers, albeit having fewer compatible solvers. OPL, on the other hand, is tied to the CP/MIP solver IBM ILOG CPLEX whereas MiniZinc supports a broad variety of solvers – a property found useful in our evaluation in Chapter 9.

Numberjack follows a different path by not being designed as a domain-specific language but rather an object-oriented API in Python [Hebrard et al., 2010]. The models are then translated for various backend solvers without the flat file format present in MiniZinc. It is beneficial that users do not need to learn a new language but rather work with established technologies. Moreover, their platform also provides a FlatZinc parser such that it can serve as backend to MiniZinc as well. Supported solvers include Mistral [Hebrard, 2008] and Toulbar2 [Allouche et al., 2010].

## 3.5 Related Work

Probably closest to our overall approach of MiniBrass, Ansótegui et al. [2011] proposed the higher-level language concept "w-MiniZinc" which would extend MiniZinc to weighted CSP

(but only weighted CSP). Also, Ansótegui et al. [2013] offers WSimply, a specification language for weighted CSP with a transformation to SMT. However, their approach was never implemented. MiniBrass, by contrast, is designed to be easily extended with new types (fuzzy, probabilistic, constraint preferences, etc.) and puts a layer of abstraction on top of MiniZinc, being its target language of compilation. Moreover, modular specifications such as those offered by products in MiniBrass are not available in either w-MiniZinc or WSimply. In addition, the syntactical features offered by w-MiniZinc are also available in MiniBrass (see Section 4.3.1) – along with more pre-defined types to choose from, the ability to define new types, and complex preference structures assembled from smaller preference structures.

Clearly, other areas study preferences with different emphases, ranging from game theory, databases [Kießling and Köstler, 2002], the social sciences [Allen et al., 2015], mechanism design [Nisan and Ronen, 1999] to multiagent systems, in general [Shoham and Leyton-Brown, 2008]. Often, a preference relation is represented by numeric utilities that can be translated to weighted or fuzzy constraints. CP-nets [Boutilier et al., 2004] provide the most common *qualitative* preference language used in the above domains. Users specify total orders over the domain of a variable depending on an assignment to other variables. For instance, $x_1 = d_1, \ldots, x_n = d_n : y = w_1 \succ \cdots \succ y = w_k$ indicates that if variables $x_i$ are assigned to $d_i$, then variable $y$ should preferably be assigned $w_i$ than $w_{i+1}$. By applying these rules transitively under a *ceteris paribus* assumption (all other things being equal), generally a preorder over assignments is induced. In terms of solution ordering, it is well-known that soft constraints and CP-nets are formally incomparable [Meseguer et al., 2006]. Compared to *constraint* preferences, CP-nets require users to rank domain *values* whereas constraint preferences are settled on a coarser level: solutions satisfying an important constraint $A$ are better than solutions satisfying a less important constraint $B$ – ceteris paribus. The former is obviously better suited in problems involving rather small domains whereas the latter aims at ordering a large number of solutions in equivalence classes of manageable size. We discuss related approaches to constraint preferences in Section 5.3.

Regarding the specification and aggregation of preferences in multi-agent settings, (computational) social choice provides formal foundations by means of axiomatizing desirable properties and postulating appropriate voting rules [Brandt et al., 2013; Shoham and Leyton-Brown, 2008]. Little attention has yet been devoted to the combination of social choice with soft constraint problems consisting of $n$ preference structures [Dalla Pozza et al., 2011] even though the prevalent heterogeneity calls for such approaches (see Section 8.1.3).

At the intersection of constraint programming and multi-agent systems, much effort has been invested in the so-called distributed constraint optimization problems (DCOP) [Yokoo and Hirayama, 2000; Fioretto et al., 2016]. A paramount goal in DCOPs is to provide asynchronous and distributed variants of the classical constraint solving algorithms such as, e.g., asynchronous branch-and-bound [Modi et al., 2005; Yeoh et al., 2010]. The involved agents each take responsibility for some variables and communicate assignments and propagation results via message-passing. By contrast, our approach is concerned with modeling and specification, *in principle*, without making any assumptions or restrictions on the underlying solvers' architecture or distribution. Before we consider a distributed execution at all, we want to provide means to adequately specify and aggregate preferences of multiple agents. More technically, the overall objective in DCOP is usually a sum of local cost functions which amounts to the special case of a weighted CSP [Fioretto et al., 2016] as opposed to more generic frameworks such as those supported in MiniBrass. This weighted combination is moreover problematic in terms of fairness and bias (see Chapter 8). Still, research in DCOP and soft constraints

is widely orthogonal such that an integration of a DCOP solver as a backend of MiniBrass is possible. Among the most sophisticated DCOP solvers are FRODO [Léauté et al., 2009] and DisChoco 2 [Wahbi et al., 2011], which are strongly tied to JaCoP [Kuchcinski and Szymanek, 2013] and Choco [Jussien et al., 2008], respectively. Both of the latter (non-distributed) solvers support FlatZinc out of the box. Therefore, a FlatZinc parser for FRODO or DisChoco 2 is an interesting engineering effort.

Chapter **4**

# MiniBrass – A Soft Constraint Modeling Language

**Summary.** We present the syntactic elements of MiniBrass in both a graphical and textual form. This includes the definition of new PVS types, the instantiation of new preference structures, morphisms to transform preference structures, as well as their combinations. Also, concepts for data validation and usability play an important role.

**Publication.** The concepts presented in this chapter are published in [Schiendorfer et al., 2018].

Our considerations up to now have been mostly devoted to the motivating application scenarios as well as to existing formalisms in the literature and their generalizing algebraic structures. We now turn to the design of MiniBrass as an extension of MiniZinc and how it includes these existing formalisms. As already hinted (and explained in detail in Chapter 6), partial valuation structures are the adequate algebraic structure to *encode* soft constraint formalisms such that the theoretical constructions substantiate the MiniBrass language. As such, MiniBrass revolves around the concept of PVS as the main abstraction of concrete preference structures and can be either thought of mathematically as partial valuation structures or, closer to a user-perspective, as "preference (valuation) structures". They are thus the essential building blocks to form complex preference models of various types.

But not only are PVS a useful abstraction tool to subsume the specific formalisms lying beneath them. They are eminently suitable for *graphical representations*, including their compositional structure. That way, domain-experts need no background in formal modeling or optimization to discuss their objectives and preferences systematically. For example, constraint preferences are most easily represented by their preference graph, lexicographic hierarchies can be displayed in layers, or Pareto-products lead to equally placed concepts. Figure 1.1 visualizes Example 1.1 that summarizes this idea with a scenario taken from unit commitment in virtual power plants. Besides the graphical representation, MiniBrass also has a clean textual interface.

Mini*Zinc* in its own right offers a well-balanced compromise between expressive power and broad support by a variety of solvers including propagation engines such as Gecode or JaCoP but also other paradigms such as MIP or SAT solvers such as CPLEX. It is the only state-

```
include "hello-world_o.mzn"; % output of minibrass
include "soft_constraints/minibrass.mzn"; % for generic branch and bound

% the basic, "classic" CSP
set of int: NURSES = 1..3;
int: day = 1; int: night = 2; int: off = 3;
set of int: SHIFTS = {day,night,off};
array[NURSES] of var SHIFTS: n;

constraint (exists(i in NURSES) (n[i] = night) );

solve
search miniBrass(); % calls to a generic PVS-based branch-and-bound
```

Listing 4.1: `hello-world.mzn`: The conventional constraint model contains all variable definitions and hard constraints. It includes the compiled MiniBrass output (`hello-world_o.mzn`) which contains generated variables, linking constraints, search procedures relevant to MiniSearch (`miniBrass`, defined in `minibrass.mzn`), and the (optional) search annotation `pvsSearchHeuristic`.

of-the-art constraint modeling language that offers high-level concepts such as user-defined functions and predicates for encapsulation of recurring combinatorial substructures [Stuckey and Tack, 2013]. To smoothen the transition between conventional constraint models and soft constraint models, MiniBrass follows many MiniZinc conventions such as having a "solve" item, independence of order of statements and the notation, in general.

MiniBrass is designed to capture such PVS-based soft constraint problems. In MiniBrass, a model (resp., instance) is divided into a (hard) *constraint model* (see Listing 4.1) written in conventional MiniZinc, consisting of variable definitions and classical constraints, and a *preference model* (see Listing 4.2) which contains PVS type declarations along with instantiations, soft constraint definitions based on the variables in the constraint model, and combinations (Pareto and lexicographic) of instances. MiniBrass separates essential constraints of a problem from its objective for several reasons:

- Previously described soft constraint formalisms in the literature (weighted, fuzzy, constraint preferences, possibilistic, etc.) are available for a preference model using the predefined PVS types.
- Preferences can be elicited and specified using PVS type $\mathcal{A}$ (perhaps having a non-trivial (multi)set-based order such as in constraint preferences) which is then transformed to another PVS type $\mathcal{B}$ that is better supported by existing solvers (cost function networks or integer objectives) using morphisms (see Section 4.4).
- By exploiting modularity, users can combine several preference structures (e.g., stemming from different agents) at runtime (Pareto or lexicographic).
- Multiple preference models for the same hard constraint model can co-exist and be selected at runtime depending on other context factors.

Conceptually, the main idea of how to encode a soft constraint problem as a conventional constraint optimization problem that MiniZinc or MiniSearch can solve has been outlined in Meseguer et al. [2006] after being first described by Petit et al. [2000]: For a soft constraint problem $SCSP = (X, D, C, M, C_s)$ with PVS $M = (|M|, \cdot_M, \varepsilon_M, \leq_M)$, we define the classical constraint model $(X, D, C)$ as usual and for every soft constraint $\mu_i \in C_s$, we have a hard constraint along the lines of "$\texttt{valuation}[i] = \texttt{mznExpr}_i(X)$" where `valuation` is an array of generated variables of type $|M|$ and $\texttt{mznExpr}_i(X)$ stands for the MiniZinc expression of soft constraint

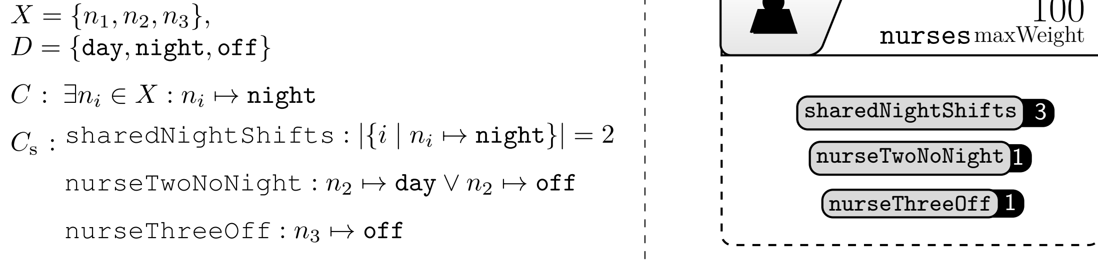



Figure 4.1: Left: The formal definition of a rostering problem (initially presented in Figure 3.2). Right: The PVS for this model, now given as a weighted CSP. Labels in black indicate associated weights.

```
include "defs.mbr"; % to get predefined type "WeightedCsp"

PVS: nurses = new WeightedCsp("nurses") {
   soft-constraint sharedNightShifts: 'sum(i in NURSES)(bool2int(n[i] = night)) = 2' :: weights('3');
   soft-constraint nurseTwoNoNight: 'n[2] in {day,off}' :: weights('1');
   soft-constraint nurseThreeOff: 'n[3] = off' :: weights('1');
};

output '["n = \(n)\nValuations:  topLevelObjective = \(topLevelObjective)\n"]';

solve nurses;
```

Listing 4.2: hello-world.mbr: A preference model to accompany the constraint model in Listing 4.1 with one PVS-instance of type WeightedCSP that also serves as the “solve"-item analogous to MiniZinc.

$\mu_i$, based on variables $X$. Additionally, there is an $|M|$-variable **overall** holding the overall valuation which is constrained such that “$\texttt{overall} = \texttt{valuation}[1] \cdot_M \ldots \cdot_M \texttt{valuation}[\texttt{nScs}]$” where **nScs** refers to the number of soft constraints. The partial ordering $\leq_M$ is used to generate constraints on future solutions such as “$\texttt{overall} <_M \texttt{overall}'$” to ask for the next solution's value $\texttt{overall}'$ to be *better* than the current one's value **overall**. MiniSearch-based branch-and-bound (see Section 4.4.2) is based on this predicate.

## 4.1 A Hello-World Example

To keep the exposition easily understandable, we turn to the simple rostering problem already shown in Figure 3.2 for our first MiniBrass example. Figure 4.1 repeats the problem statement and defines a weighted constraints PVS (in contrast to the earlier used constraint preferences). Each soft constraint $\mu_i$ maps to $w_i$ if violated and 0 otherwise.

Listing 4.1 first presents the conventional constraint model of the problem in standard MiniZinc. To capture the weighted variant of the problem, we use the predefined PVS type WeightedCsp from defs.mbr which defines integers as the underlying partial order of solution degrees, summation as the combination operation, and the usual $\geq$-ordering. Users, however, do not have to directly map to $\{0, \ldots, k\}$ but rather supply a boolean expression that is translated as above using a parameter weight (see “specification type” in Section 4.2). Listing 4.3 shows the part of defs.mbr that contains the mappings to MiniZinc functions and predicates in the file weighted_type.mzn, i.e., combination maps to weighted_sum and

```
type WeightedCsp = PVSType<bool, int> =
  params {
    int: k :: default('1000');
    array[1..nScs] of int: weights :: default('1');
  } in
  instantiates with "soft_constraints/mbr_types/weighted_type.mzn" {
    times -> weighted_sum;
    is_worse -> is_worse_weighted;
    top -> 0;
 };

%%% in "weighted_type.mzn" %%%
predicate is_worse_weighted(var int: x, var int: y,
  par int: nScs, par int: k, array[int] of par int: weights) =
  x > y;

function var int: weighted_sum(array[int] of var bool: b,
                               par int: nScs, par int: k, array[int] of par int: weights) =
  sum(i in index_set(b)) ( (1 - bool2int(b[i])) * weights[i]);
```

Listing 4.3: The type definition used in the hello world example in Listing 4.2 is found in `defs.mbr`.

ordering maps to `is_worse_weighted`. While we show the contents of `defs.mbr` here, we note that *users* do not *have* to know about this when defining a PVS-based model. They can rely on the predefined types and only have to supply the model in Listing 4.2.

More systematically, the toolchain needed for MiniBrass adds an additional preceding step to the conventional MiniZinc/MiniSearch workflow. Figure 4.2 illustrates the involved processes: First, the MiniBrass preference model (e.g., Listing 4.2) is compiled into MiniZinc or MiniSearch code using `mbr2mzn`. During this process, auxiliary variables taking soft constraint valuations and their aggregations, improvement and not-worsening constraints for branch-and-bound, as well as variable orderings for search heuristics and complex order predicates (in case of Pareto or lexicographic combinations) are generated. Finally, the classical constraint model (e.g., Listing 4.1) includes the compiled MiniBrass output and is solved by MiniZinc (`mzn2fzn`) or MiniSearch (`minisearch`) and a FlatZinc solver.

**Example 4.1 – Executing Rostering**

When we run this toolchain with our example from Listings 4.1 and 4.2, we get the following output:

```
Intermediate solution:n = [night, day, day]
Valuations:  topLevelObjective = 4
----------
Intermediate solution:n = [night, night, day]
Valuations:  topLevelObjective = 2
----------
Intermediate solution:n = [night, day, night]
Valuations:  topLevelObjective = 1
----------
==========
```

We start with the solution `[night, day, day]` that violates the soft constraints **`sharedNightShifts`** (since only one nurse is on night shift) and **`nurseThreeOff`** since nurse three has to work the day shift – totaling in a violation sum of 4. The solver provides an improved solution by having nurse two move to the night shift which satisfies **`sharedNightShifts`**

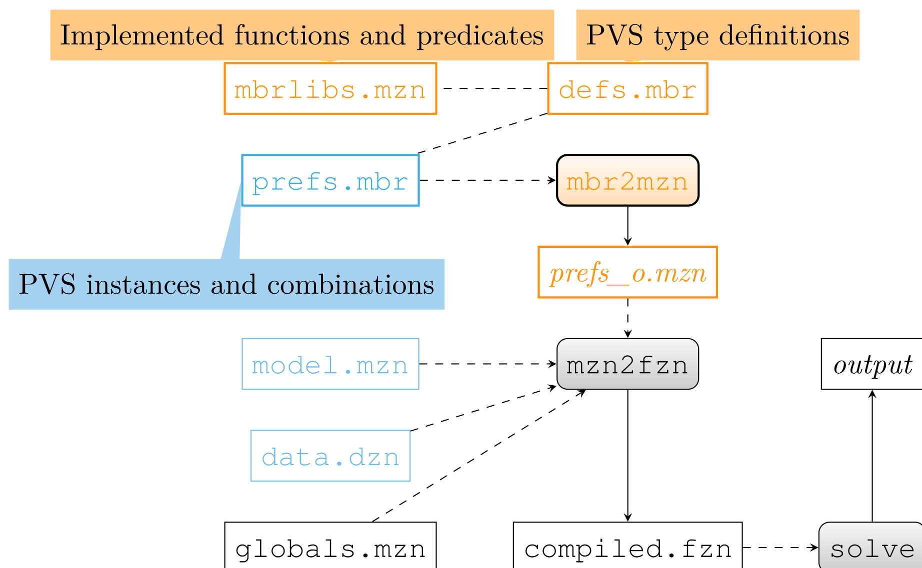



Figure 4.2: Toolchain of MiniBrass and its integration with MiniZinc. Blue elements indicate artifacts that have to be created for every new problem instance whereas orange ones refer to reusable (MiniBrass) library items that do not need to be modified by end users.

but violates `nurseThreeOff` and `nurseTwoNoNight`, leading to a penalty sum of 2. In fact, this can be seen as "trading" the violation of a less important constraint in favor of satisfying a more important constraint. Finally, the last found solution [`night`, `day`, `night`] only violates `nurseThreeOff` but satisfies the other two soft constraints which results in the optimal value 1. Qualitatively speaking, we got rid of the violation of `nurseTwoNoNight`. Since there is no solution satisfying all soft constraints, this is the best we can do.

## 4.2 PVS Types and Instantiations

As we have seen in the first example, specifying a preference model begins with defining appropriate PVS types – with most of the literature's proposals being already implemented in the MiniBrass standard library. Every PVS type definition has to specify the possible solution degrees, ordering predicate and combination operation in MiniZinc. The possible solution degrees are referred to as the *element type* $E$ of a PVS type. This can be any MiniZinc base type such as `int`, `float`, `bool`, or subtypes thereof as well as sets or arrays of a base type.

For instance, weighted constraints or cost function networks map to `int` or some integer range `0 .. k`, fuzzy constraints employ the float range `0.0 .. 1.0`, or the inclusion-based Max-CSP maps to sets of integers representing violated constraints. The element type $E$ corresponds to (a subset) of the carrier set $|M|$ of a resulting PVS instance $M = (|M|, \cdot_M, \varepsilon_M, \leq_M)$

and prescribes the signature of the ordering $\leq_M$, i.e., $\leq_M \subseteq E \times E$. Canonically, every soft constraint $\mu_i$ maps to $E$ and the combination operation has to be a function $\cdot_M : E \times E \to E$. However, there are cases where the essential model information is different from an $E$-element – consider our previous toy example where soft constraints map to `bool` but are wrapped by an embedding function that converts them to `int`. This becomes most evident in all PVS that rely on "violated soft constraints" such as, e.g., weighted constraints, constraint preferences, probabilistic constraints, or constraint hierarchies. Each soft constraint $\mu_c : [X \to D] \to |M|$ (parametrized by some conventional constraint $c$ and associated weight $w_c$) then has the form $\mu_c(\theta) = 0$ if $\theta \models c$ and $w_c$ otherwise: The *essential* information here is whether $\theta \models c$. Wrapping every boolean expression obviously leads to cluttered and less readable constraint and preference models. For instance, in Listing 4.2, we would have to write

```
soft-constraint sharedNightShifts: '(1 - bool2int(sum(i in NURSES)(bool2int(n[i] = night)) = 2)) * 3';
```

instead of

```
soft-constraint sharedNightShifts: 'sum(i in NURSES)(bool2int(n[i] = night)) = 2' :: weights('3');
```

if we wanted to directly map an assignment to the underlying PVS $(\{0, \ldots, 100\}, +_{100}, \geq, 0)$.

To avoid this clutter on the syntactic level, PVS types can be augmented with a *soft constraint type* $S$ that defines the type of each soft constraint expression (here `bool`). We semantically define a mapping $g_i : S \to E$ that assigns the $S$-expression of the soft constraint $\mu_i$ to an $E$-element. All soft constraints' mapped $E$-elements are combined with $\cdot_M$. However, there are cases when we do not want or need to have $n$ $E$-expressions but rather emulate the above combination using an embedding $S^n \to E$ – especially if there are beneficial global constraints involved (see Section 4.3.2). In both cases, we end up with an overall valuation of type $E$ that is ordered using $\leq_M$ during the search. As a consequence, we have the combination operation map several $S$-expressions to one element in $E$ (where $S = E$ is, of course, valid) and have it mimic the $\cdot_M$ operation on $E$.

Moreover, PVS types may declare parameters that need to be specified when constructing an instance in a preference model. Examples thereof are the maximally allowed weight $k$ in weighted constraints or the preference graph in constraint preferences. The MiniZinc implementation of the combination function and ordering predicate has to be supplied in a separate MiniZinc file that will be included by the compiled MiniZinc output. Although all presented types are included in the MiniBrass standard library, new definitions can likewise be added since MiniBrass keeps an eye on extensibility.

To sum up, for a PVS type parametrized by soft constraint type $S$ and element type $E$, the ordering predicate, combination, and top element have to implement these interfaces in MiniZinc:

```
predicate is_worse(var E: x, var E: y, par int: nScs [, ⟨PVS type parameters⟩]);
function var E: times(array[int] of var S: v, par int: nScs [, ⟨PVS type parameters⟩]);
par E: top;
```

The ordering of the PVS type parameters must match the order in the PVS type declaration. In a similar way, some PVS types offer generic search heuristics that can be provided (see the keyword `pvsSearchHeuristic` below). The interface is expected to be:

```
function ann: searchHeuristic(array[int] of var S: values, var E: overall,
                              par int: nScs [, ⟨PVS type parameters⟩]);
```

It should generally be noted that `is_worse` always corresponds to a predicate denoting *strict* worsening (this is the most common type of predicate used in branch-and-bound). The `top` element is beneficial for bounding search and having default soft constraints.

Besides the type declarations, there are a few "technical" MiniBrass keywords that are sure to be found in the compiled MiniZinc output (e.g., `prefs_o.mzn` in Figure 4.2) and can thus be accessed from the constraint model (e.g., `model.mzn` in Figure 4.2):

- `topLevelObjective`: contains a `var E`-expression for an *atomic* top level PVS (the instance specified in the `solve` item), with element type $E$; not applicable if the top level PVS is complex (e.g., a lexicographic product). It appears in the output in Listing 4.1 and could also be a MiniZinc objective: `solve minimize topLevelObjective` (only if $E$ is scalar).
- `pvsSearchHeuristic`: contains an annotation object for the top level PVS that holds a particular variable order (of the generated variables) that depends on the PVS type(s) involved (see Section 4.3.1). For complex PVS, multiple heuristics are concatenated sequentially.
- `postGetBetter`: contains a MiniSearch procedure that is used to post a constraint requiring the next solution to be better than the current one. The generic branch-and-bound procedure `miniBrass` used in Listing 4.1 (defined in `pvs_gen_search.mzn` and explained in Section 4.4.2) relies on `postGetBetter` being written by the compiler.
- `postNotGetWorse`: dually, this MiniSearch procedure only requires the next solution not to be dominated (important to find all optima of an instance).

Once PVS types are declared, we can use them to instantiate concrete PVS objects. A PVS object stores a specific set of parameters and includes the actual soft constraints mapping to $E$ (or $S$) as MiniZinc expressions – thereby connecting the constraint and preference model. In addition, the operators `pareto` and `lex` can be used to compose complex preference structures from elementary ones.

## 4.3 Examples of Soft Constraint Formalisms as PVS Types

For illustration purposes, we survey the most common soft constraint formalisms (see Section 3.2) presented as PVS types. We frequently use variants of the rostering problem as our running example to show how the concrete PVS types differ.

### 4.3.1 Integer-Valued: Weighted CSP or Cost Function Networks

We have already seen the PVS type for weighted constraints in the introductory example. Cost function networks are naturally very similar to weighted CSP. The latter are defined as integer-valued soft constraints that map any assignment to some value in the range $[0 \ldots k]$ for some parameter $k$ denoting maximal violation and top being 0 (recall Figure 3.1). Consequently, there is no distinct soft constraint type but just the element type `0..k`. Due to this generality, cost function networks are an excellent match to embed conventional numeric costs into the PVS-based MiniBrass universe.

```
type CostFunctionNetwork = PVSType<0..k> =
  params {
    int: k :: default('1000');
  } in
  instantiates with "soft_constraints/mbr_types/cfn_type.mzn" {
    times -> k_bounded_sum;
    is_worse -> is_worse_weighted;
    top -> 0;
};
```

Combination refers to adding individual costs, bounded by $k$. The the ordering relation is simply the integer greater-than ordering (consistently with literature, cost minimization is default):

```
% Inside soft_constraints/mbr_types/cfn_type.mzn
predicate is_worse_weighted(var int: x, var int: y, int: nScs, int: k) =
  x > y;

function var int: k_bounded_sum(array[int] of var int: b, int: nScs, int: k) =
  if sum(b) > k then k else sum(b) endif;
```

**Example 4.2 – Toy Cost Function Network in MiniBrass**
We can model Figure 3.1 using the PVS type `CostFunctionNetwork`, starting with the constraint model and omitting includes:

```
var 0..1: x; var 0..1: y; var 0..1: z;
% useful since we have a numeric objective and can use MiniZinc *or* MiniSearch
solve minimize topLevelObjective;
```

Based on this simplistic constraint model, we are ready to define the soft constraints which, recall, map directly to integers and are summed and minimized:

```
include "defs.mbr";

PVS: cfn1 = new CostFunctionNetwork("cfn1") {
   soft-constraint mu_x: 'x+1';
   soft-constraint mu_y: '2-y';
   soft-constraint mu_z: 'if (z == 0) then 3 else 1 endif';

   soft-constraint mu_xy: 'if (x == 0) then
                              if (y == 0) then 4 else 2 endif
                           else if (y == 0) then 2 else 1 endif
                           endif';
   soft-constraint mu_yz: 'if (y == 0) then
                              if (z == 0) then 1 else 3 endif
                           else if (z == 0) then 0 else 2 endif
                           endif';
};

output '["x = \(x); y = \(y); z = \(z)"]';

solve cfn1;
```

As we can see from this example, the enumerative definition of discrete cost functions can become rather unwieldy. Besides the core type definition, the MiniBrass library thus offers additional utility functions and predicates: For better access to *native cost function* implementations in Toulbar2, there is a global constraint (along with a default decomposition for other solvers, see Section 3.4 for how this MiniZinc mechanism operates) that is handled by Numberjack and properly given to Toulbar2. For instance,

```
predicate cost_function_binary(var int: x, var int: y,
                               array[int] of int: costs, var int: costVariable)
```

ties a cost function's valuation for variables `x` and `y` to a cost variable `costVariable`, depending on a given cost vector that contains the value for every element in the Cartesian product of the domains of `x` and `y`. In Example 4.2, we could have written

```
soft-constraint mu_xy: 'cost_function_binary_fn(x, y, [4, 2, 2, 1])';
```

The default decomposition for `cost_function_binary_fn` that all solvers except for Toulbar2 use, is implemented using a `table` constraint, i.e., an extensional representation. In a

similar spirit, *soft global constraints* are implemented in MiniBrass. Since the existing soft globals map to a numeric variable, they naturally lead to cost function networks. For instance, a soft variant of `alldifferent` counts the variables taking the same value as a measure of violation:

```
% a default decomposition for solvers that do not provide the soft global constraint "soft_alldifferent"
function var int: soft_alldifferent(array[int] of var int: x) :: promise_total =
  let { set of int: seenValues = dom_array (x); }
  in (sum(s in seenValues) (max(count(x, s) - 1, 0)));

% [...] Used in a MiniBrass preference model
include "soft_constraints/soft_alldifferent.mzn";

% [...] assume we assign students to projects
array[STUDENT] of var PROJECT: x;

% ideally, all work on their own project
% but we resort to a soft, optimization variant
soft-constraint studentsSharingAProject: 'soft_alldifferent(x)';
```

There are native implementations for `soft_alldifferent` (e.g., in JaCoP [Kuchcinski and Szymanek, 2013]) which can make use of dedicated propagation (JaCoP reduces it to a network flow constraint) instead of this provided decomposition.

In contrast to cost function networks, weighted constraints are binary – a soft constraint is violated or not, and if so, penalized by a fixed weight. This is reflected by using the soft constraint type `bool` that is mapped to the element type `0..k`.

```
type WeightedCsp = PVSType<bool, 0..k> =
  params {
    int: k :: default('1000');
    array[1..nScs] of int: weights :: default('1');
  } in
  instantiates with "soft_constraints/mbr_types/weighted_type.mzn" {
    times -> weighted_k_bounded_sum;
    is_worse -> is_worse_weighted;
    top -> 0;
  }
 offers {
    heuristics -> getSearchHeuristicWeighted;
 };
```

In addition to our previous definition of `WeightedCsp`, we now also consider the first example of a generic search heuristic annotation that comes with the PVS type. The MiniZinc function `getSearchHeuristicWeighted` (defined in `weighted_type.mzn`) provides a particular variable ordering for search: the generated variables containing the highest-weighted possible violation first (called *most important first* in Schiendorfer et al. [2014b]). That way, search can initially set the generated satisfaction variables of all soft constraints to *true* and let propagation take over to possibly find high-quality solutions early, optimally an assignment satisfying all soft constraints:

```
function ann: getSearchHeuristicWeighted(array[int] of var bool: scSatisfied,
                                         var int: overall,
                                         par int: nScs, % number of soft constraints
                                         int: k, array[int] of int: weights) =
let {
  set of int: sCs = 1..nScs;
  % find the sorted permutation of soft constraint instances
  array[sCs] of sCs: sortPerm = arg_sort(weights);
  % invert, since arg_sort uses <= and we need decreasing order
  array[sCs] of sCs: mostImpFirst = [ sortPerm[nScs-s+1] | s in sCs];
  array[sCs] of var bool: mifSatisfied = [ scSatisfied[mostImpFirst[s]] | s in sCs];
} in
int_search(mifSatisfied, input_order, indomain_max, complete);
```

Section 9.3 provides some insight in the effectiveness of the above search heuristic.

There is a double-usage of the PVS type `WeightedCsp`. As we set the weights' default value to 1, we get a Max-CSP instance (counting-based, as explained in Section 3.2) if no weights are supplied. As seen in the hello world example, we can add weights (more generally, parameter values tied to every soft constraint) by annotating a soft constraint during instantiation:

```
PVS: wcsp = new WeightedCsp("wcsp") {
  soft-constraint c1: 'x + 1 = y' :: weights('2');
  soft-constraint c2: 'z = y + 2' :: weights('1');
  soft-constraint c3: 'x + y <= 3' :: weights('1');
  k : '20';
  % alternatively, we could also write
  % weights : '[2, 1, 1]';
};
solve wcsp;
```

Weighted constraints and cost function networks share the fact that their top level objective function is, after all, a single integer variable. Therefore maximization or minimization is supported out of the box in MiniZinc. As we see in the next session, this is not true for other PVS types.

### 4.3.2 Comparative: The Free PVS and Constraint Preferences

The benefit of MiniBrass is that users are not bound to using numeric types, as in the previous examples, but can order their solutions with more abstract types that help to aid understanding. We begin with the *most general* type, i.e., the *free PVS* over a partial order (see Chapter 6 or [Schiendorfer et al., 2018] for a thorough presentation and motivation of the concept). Constraint preferences constitute a special case that can be formalized using the free PVS. In a nutshell, the most general way to lift any partial order $(P, \leq_P)$ to a PVS is by taking *finite* multisets of elements of $P$ as the underlying set (written as $\mathcal{M}_{\text{fin}}(P)$), multiset union as combination with $\wr\int$ as the neutral element, and the Smyth-ordering as the PVS ordering (written as $\preceq_P$, see Figure 4.3). Intuitively, multisets are required for the case that two soft constraints map to the same element $p$. Then $p \cdot_M p$ should be different from $p = p \cdot_M \varepsilon_M$, i.e., idempotency is not necessarily fulfilled in a PVS.

A partial order over a finite set of elements is most conveniently represented in MiniZinc as an adjacency list. We assume parameters `maxP` (denoting the highest index) and `orderingP` (the ordering relation as a list of edges). For example,

```
int: maxP = 3;
set of int: P = 1..maxP;
array[int, 1..2] of 1..maxP: partialOrdering = [| 1, 2 | 1, 3 |];
```

represents a directed graph $(\{1, 2, 3\}, \{(1, 2), (1, 3)\})$ that we convert to a partial ordering by taking the reflexive-transitive closure later on.

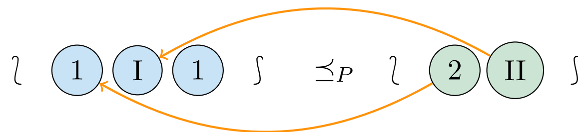



Figure 4.3: A visual depiction of the Smyth-ordering. Assuming $P$ is defined such that $1 < 2$ and $\text{I} < \text{II}$, a witness function pairs every element on the right-hand side with a dominated counterpart on the left-hand side (see Lemma 6.1).

The needed multisets are not natively supported by MiniZinc but have to be encoded with the existing MiniZinc types. The free PVS' underlying set $\mathcal{M}_{\text{fin}}(P)$ itself is clearly infinite as we can reach any finite multiset over $P$ by applying combination (i.e., multiset union) often enough. Since existing solvers operate on *finitely* many decision variables with finite domains, we never have to use the full range of $\mathcal{M}_{\text{fin}}(P)$, though, and always operate on a finite subset of it. Put differently, the maximal multiplicity of any element of $P$ is necessarily restricted for every multiset we face in the course of optimization. To put a meaningful upper bound on the multiplicities, we note that in an *SCSP*, the overall valuation is given by $\prod_M \{\mu_i(\theta) \mid \mu_i \in C_s\}$. If we can determine the maximal occurrence any $P$-element has in any individual $\mu_i(\theta)$, say $k$, we simply use **nScs** $\cdot\, k$ as the maximal occurrence for the overall valuation. For example, in constraint preferences, the maximal occurrence is, in fact, easy to determine since any soft constraint can only be violated once – we exploit this further below. With these restrictions, a multiset composed of $P$-elements with maximal multiplicity `maxOccurrences` is just encoded as an `array[1..maxP] of var 0..maxOccurrences`. For instance, `[1,0,0]` represents the multiset $\{\!|1|\!\}$ or `[1,0,2]` would be $\{\!|1,3,3|\!\}$. Due to the relevance of multisets for the free PVS, MiniBrass defines the keyword `mset` that abbreviates an array of integers representing a multiset. Consequently, the free PVS' element type is `mset[maxOccurrences] of 1..maxP` which is syntactic sugar for an `array[1..maxP] of var 0..maxOccurrences` that represents the overall solution degree.

```
type FreePVS = PVSType<mset[maxOccurrences] of 1..maxP> =
  params {
    array[int, 1..2] of 1..maxP: partialOrdering  ::
        wrappedBy('java', 'isse.mbr.extensions.preprocessing.TransitiveClosure');
5   int: maxP;
    int: maxPerSc;
    int: maxOccurrences :: default('mbr.nScs * mbr.maxPerSc');
  } in
  instantiates with "soft_constraints/mbr_types/free-pvs-type.mzn" {
10  times -> multiset_union;
    is_worse -> isSmythWorse;
    top -> [0 | i in 1..mbr.maxP]; % the empty multiset
};
```

For user convenience, we have the reflexive-transitive closure automatically calculated by MiniBrass during compilation (turning a DAG into a partial ordering) – as an example of a parameter wrapping method. Such parameter wrapping methods could either be MiniZinc functions for data transformation or Java methods. Ensuring correct user input (i.e., validations by means of MiniZinc assertions or Java exceptions) can be done here as well. For instance, if a cycle is supplied to MiniBrass, this error is detected and reported.

Each soft constraint maps to a multiset with bounded multiplicity, as indicated by the parameter `maxPerSc`. By default, the overall maximal multiplicity is simply the product of the number of soft constraints and `maxPerSc`. While the combination (`multiset_union`) is straightforward by just summing the individual soft constraints' element multiplicities, implementing a MiniZinc predicate for the Smyth-ordering is a bit more involved but showcases MiniBrass' abilities.

In essence, to establish $T \prec_P U$, the key idea is to apply Lemma 6.1 and have the witness function be *decided* by the solver using local decision variables of the predicate. To make up for the fact that multisets are used, a witness $h$ is defined as a map $h : \mathcal{S}(U) \to \mathcal{S}(T)$ where $\mathcal{S}(U)$ refers to a "set of pairs" representation of a multiset, e.g., $\mathcal{S}(\{\!|4,4,3|\!\}) = \{(1,4),(2,4),(1,3)\}$. Thus $h$ is defined on *pairs*, has to be injective, and satisfy the constraints $p \leq_P q$ whenever $h(j,q) = (k,p)$. Injectivity is best modeled by an `alldifferent`-constraint but there is none for pairs. We can mitigate this by defining a one-dimensional witness and apply a bijection

between the pairs and the one-dimensional array. The resulting one-dimensional array can then be constrained to be all different, as usual. We chose the bijective Cantor pairing function $\pi : \mathbb{N}^2 \to \mathbb{N}$ defined by

$$\pi(k_1, k_2) := \frac{1}{2}(k_1 + k_2)(k_1 + k_2 + 1) + k_2$$

```
predicate isSmythWorse ( array[int] of var int: T, array[int] of var int: U, % T < U
              int: nScs, array[int, 1..2] of int: edges, % adjacency list
              int: maxP, int: maxPerSc, int: maxOccurrences % parameters of free PVS type
          ) = let {
  set of int: P = 1..maxP; % multisets draw from the set of elements P
  par int: maxOcc = maxPerSc*maxOccurrences; % bounding multiplicities, see Sect. 4
  set of int: OCCS = 0..maxOcc;
  set of int: PosOCCS = OCCS diff {0};
  set of int: P0 = {0} union P; % 0 representing no assignment for the witness

  set of int: edgeIndices = index_set_1of2(edges);
  % for each element in P, we pre-calculate the set of ``smaller'' items
  % according to the adjacency list (edges)
  array[P] of set of P: lessThanOrEquals =
    [ {q} union {p | p in P where exists(e in edgeIndices)
         (edges[e,1] = p /\ edges[e,2] = q)} |  q in P];

  % We have to split the witness function h : S(U) \to S(T) into
  % two arrays of decision variables.
  array[OCCS,P] of var P0: witnessElem;  % element part of h
  array[OCCS,P] of var OCCS: witnessOcc; % occurrence part of h

  % First, we make sure all (j,q) tuples for occurrences j greater than the
  % actual number of q elements in U map to non-existence.
  constraint forall(q in P, j in OCCS where j > U[q]) (
    witnessElem[j,q] = 0 /\ witnessOcc[j,q] = 0
  );

  % Now, for all (j,q) tuples in S(U), they have to map
  % to a (k,p) tuple in S(T) such that p <= q.
  constraint forall(q in P, j in PosOCCS where j <= U[q])  (
    % p must not be 0 and p must be leq than q  (according to "edges")
    witnessElem[j,q] != 0 /\ witnessElem[j,q] in lessThanOrEquals[q] /\
    % k must be between 1 and the actual number of p-occurrences in T
    witnessOcc[j,q] >= 1 /\ witnessOcc[j,q] <= T[witnessElem[j,q]]
  );

  % Lastly, we have to assert injectivity of our witness, using the Cantor pairing
  % function to map S(U) to int and constrain the Cantorized witness to be alldifferent.
  array[OCCS,P] of var 0 .. maxP + (maxOcc) * (maxOcc + maxP + 1) div 2:
    cantoredWitness;
  constraint forall(i in OCCS, p in P) (
   cantoredWitness[i,p] = witnessOcc[i,p] + (witnessElem[i,p]+witnessOcc[i,p])
                         *(witnessElem[i,p] + witnessOcc[i,p] + 1) div 2);

  constraint alldifferent_except_0([cantoredWitness[i,p] | i in OCCS, p in P]);
  % A bit of symmetry breaking on the exchangeable occurrences
  constraint value_precede_chain(OCCS, [witnessOcc[i,p] | i in OCCS, p in P]);

  % Make sure we have inequality
  constraint exists(i in P) (T[i] != U[i]);
} in (  true  );
```

At this point, we want to emphasize that *end-users* (i.e., modelers) do not have to understand the implementation of the Smyth-ordering in MiniZinc in detail but only its (rather intuitive) inductive definition to apply it in their models. The above definition is fully *encapsulated* by the type `freePVS`.

Nevertheless, there is one subtle technicality about this definition in MiniZinc. The predicate relies on *local free variables* (e.g., `witnessOcc`) which prohibit its usage in a negative or mixed context such as “it must *not* be the case that the next solution is Smyth-worse than the current one” (cf. non-domination search in Section 4.4.2). To see why this is problematic for constraint solvers, consider that negating a predicate with local variables requires

all-quantification:

$$p(x_1, \ldots, x_n) :\Leftrightarrow \exists y : q(x_1, \ldots, x_n, y) \rightarrow$$
$$\neg p(x_1, \ldots, x_n) \Leftrightarrow \forall y : \neg q(x_1, \ldots, x_n, y)$$

Most constraint solvers and mathematical optimizers focus on existential quantification, though, i.e., they search for one particular assignment.[1] For the Smyth-ordering, we would basically state a higher-level constraint such as "there exists no witness function from $\mathcal{S}(T)$ to $\mathcal{S}(U)$" or "all functions from $\mathcal{S}(T)$ to $\mathcal{S}(U)$ violate the witness property". This exceeds the capabilities of most currently available constraint solvers. Although we cannot access the Smyth-predicate in a mixed or negated context, Zinc and MiniZinc are the only constraint modeling languages that support user-defined local free variables at all [Stuckey and Tack, 2013] – and they are paramount for the Smyth-ordering. A workaround for finite multisets is to use a bijective mapping $f$ from multisets to integers (perhaps $f(\wr i, \ldots, k \wr) = p_i \cdot \ldots \cdot p_k$ where $p_i$ is the $i$-th prime number, e.g., $f(\wr 1, 1, 3 \wr) = 2 \cdot 2 \cdot 5 = 20$ and model the Smyth-ordering with a table constraint on the encoded sets which requires us to precalculate the relation beforehand.

**Example 4.3 – Nurse Rostering with the free PVS**

By construction, `freePVS` is well-suited to transform *any* partial order into a PVS. We demonstrate this by encoding constraint preferences as a free PVS instance. Recall that we have an ordering over soft constraints and a soft constraint $\mu_i$ maps to $\wr i \wr$ if violated and $\wr\wr$ otherwise. We revisit the nurse scheduling running example as a free PVS (without soft constraint type `bool` for simplicity):

```
PVS: nurses = new FreePVS("nurses") {
   soft-constraint sharedNightShifts:
      'if sum(i in NURSES)(bool2int(n[i] = night)) = 2 then [0, 0, 0] else [1, 0, 0] endif';
   soft-constraint nurseTwoNoNight:
      'if n[2] in {day,off} then [0, 0, 0] else [0, 1, 0] endif';
   soft-constraint nurseThreeOff:
      'if n[3] = off then [0, 0, 0] else [0, 0, 1] endif';

   % indices 1, 2, and 3 are used for the soft constraints
   partialOrdering : '[| 2, 1 | 3, 1 |]';
   maxP: '3' ;
   maxPerSc : '1';
};
```

The assignment `[night, day, day]` would lead to the overall degree `[1,0,1]` $\triangleq \wr 1, 3 \wr$ and is inferior to `[night, day, night]` which leads to `[0,0,1]` $\triangleq \wr 3 \wr$ in terms of the Smyth-ordering.

In this PVS type, any soft constraint can map to an arbitrary multiset (respecting the multiplicity restrictions).[2] However, for constraint preferences alone, `freePVS` might seem too rich in generality. We only observe the multisets $\wr i \wr$ or $\wr\wr$ for distinct soft constraints $\mu_i$ which means that in every overall valuation, a $P$-element (i.e., a soft constraint in constraint preferences) can only occur at most once. An intermediate step would be to annotate every soft constraint $\mu_i$ with the multiset $\wr i \wr$ (as in weighted CSP) and use the soft constraint type `bool` (effectively defining a new PVS type):

[1] SMT-solvers constitute a notable exception but there are currently none with a FlatZinc interface

[2] This would be useful, e.g., when each user can rate a travel itinerary by placing one to five stars for quality and one to five "euros" for cost preference: $\mu_1(\theta) = \wr \star, \star, €, € \wr$ and $\mu_2(\theta) = \wr \star, \star, \star, € \wr$

```
soft-constraint sharedNightShifts: 'sum(i in NURSES)(bool2int(n[i] = night)) = 2' :: msetVal('[1,0,0]');
soft-constraint nurseTwoNoNight: 'n[2] in {day,off}' ::  msetVal('[0,1,0]');
soft-constraint nurseThreeOff: 'n[3] = off' ::  msetVal('[0,0,1]')';
```

Still, we can do better in encoding constraint preferences by noting that no element $i$ can occur more than once in any reachable overall valuation $m \in \mathcal{M}_{\text{fin}}(P)$. Each such $m$ can then simply be *represented* by a *set* of integers – a type that is natively supported with appropriate global constraints by MiniZinc and several constraint solvers. More specifically, we assume that each soft constraint is identified by an integer – simply its position in the PVS instantiation. MiniBrass offers the constant `nScs` for this purpose, so we can use `set of 1..nScs` as the element type of a new dedicated PVS type `ConstraintPreferences`. Since we want to specify the soft constraints as boolean expressions (again, not mapping directly to sets), we use `bool` as soft constraint type. Of course, an instance of `ConstraintPreferences` also needs a directed acyclic graph (the `crEdges` parameter represents the adjacency list) over soft constraints. The reflexive-transitive closure is applied, similar to the free PVS:

```
type ConstraintPreferences = PVSType<bool, set of 1..nScs> =
  params {
    array[int, 1..2] of 1..nScs: crEdges
      :: wrappedBy('java', 'isse.mbr.extensions.preprocessing.TransitiveClosure');
    bool: useSPD :: default('true');
  } in
  instantiates with "soft_constraints/mbr_types/cr_type.mzn" {
    times -> union_violateds;
    is_worse -> is_worse_constraint_pref;
    top -> {};
  }
  offers {
    heuristics -> getSearchHeuristicCR;
  };
```

Combination now has to map $n$ `bool` elements to the set of violated constraints. We achieve this by means of the global constraint `link_set_to_booleans` which is defined over an array of boolean variables and a set of integers. The constraint states that the set contains precisely those indices where the array evaluates to true. Since we state soft constraints in their positive form, this constraint is applied to the negations of the satisfaction variables:

```
include "link_set_to_booleans.mzn";
function var set of int: union_violateds(array[int] of var bool: b, par int: nScs) =
let {
  var set of index_set(b): violatedSet;
  constraint link_set_to_booleans(violatedSet, [ not b[i]  | i in index_set(b) ]);
} in violatedSet;
```

The Smyth-ordering is implemented in the predicate (`is_worse_constraint_pref`) and is selected by switching `useSPD` to true, thereby activating single-predecessor-dominance. Alternatively, transitive-predecessors-dominance (TPD) is applied which lets a single soft constraint dominate a whole set of less important ones. The appropriate choice of domination semantics is obviously a matter of modeling. But perhaps more usefully, the TPD predicate is easier to decide, i.e., no free local variables are involved since *all* less important ones can be eliminated instead of deciding an injective witness. Hence, TPD is suited for mixed and negative contexts such as non-domination search.

The restriction to `set of int` also eases the implementation of the Smyth-ordering as we do not have functions over pairs – as opposed to the multiset case. We shall see, as a corollary to Lemma 6.1, on two *sets* $T$ and $U$, $T \preceq_P U$ holds if and only if there exists an *injective* witness function $f : U \to T$ such that $f(p) \leq_P p$ for all $p \in U$. Similar to the multiset case, we enforce (and propagate) the injectivity of $f$ with `alldifferent` and make sure the witness

property is fulfilled. Moreover, we only have to focus on the symmetric difference of left-hand side (`lhs`) and right-hand side (`rhs`) since the shared violated soft constraints can be ignored.

```
% -----------------
% Implements single predecessor
% dominance on sets of constraints,
% i.e., the Smyth-ordering
% --------------------------------
include "alldifferent_except_0.mzn";

predicate smyth_worse(var set of int: lhs, var set of int: rhs,
                      set of int: softConstraints,
                      array[int, 1..2] of int: edges
) = let {
  int: le = min(index_set_1of2(edges));
  int: ue = max(index_set_1of2(edges));

  var set of int: lSymDiff = lhs diff rhs;
  var set of int: rSymDiff = rhs diff lhs;

  % we need 0 as a possible value for "non-assignment" from the right-hand side
  set of int: softConstraints0 = {0} union softConstraints;

  % all the soft constraints that are not on the right-hand side any more need to map to 0
  var set of int: rUndefined = softConstraints diff rSymDiff;

  % I need to make the dominance explicit by a function
  array[softConstraints] of var softConstraints0: witness;

  % collect all less important soft constraints in an array
  array[softConstraints] of set of softConstraints: lessThanOrEquals =
    [ {succ | succ in softConstraints where exists(e in le..ue)
         (edges[e,1] = pred /\ edges[e,2] = succ)} | pred in softConstraints];
} in (
  lhs != rhs /\
  alldifferent_except_0(witness) /\
  % be sure to use an over-approximation (upper bound ub) here
  forall(s in ub(rUndefined)) (s in rUndefined -> witness[s] = 0) /\
  forall(s in ub(rSymDiff)) (s in rSymDiff -> (witness[s] in lSymDiff /\
                                               witness[s] in lessThanOrEquals[s]))
);
```

The PVS type `ConstraintPreferences` leads to a more intuitive instantiation as we see when revisiting the rostering example. Note that `mbr.`*softConstraintName* can be used as a shortcut to a soft constraint's ID.

**Example 4.4 – Rostering with `ConstraintPreferences`**
Modelers simply state their soft constraints and provide a DAG of importance.

```
PVS: nurses = new ConstraintPreferences("nurses") {
   soft-constraint sharedNightShifts: 'sum(i in NURSES)(bool2int(n[i] = night)) = 2';
   soft-constraint nurseTwoNoNight: 'n[2] in {day,off}';
   soft-constraint nurseThreeOff: 'n[3] = off';

   crEdges : '[| mbr.nurseTwoNoNight, mbr.sharedNightShifts |
                 mbr.nurseThreeOff, mbr.sharedNightShifts |]';
   useSPD: 'false' ;
};
```

Finally, we consider the set-based Max-CSP definition (see Table 3.1), i.e., $\{c_1\}$ is not better than $\{c_2, c_3\}$ because it is not a subset – although the number of violated constraints is smaller. This type is very similar to `ConstraintPreferences` except that it uses the normal superset relation instead of the Smyth-ordering. Parameters are not required. To formulate this as a PVS type, we set the element type to `set of int` although we only need a small fraction of all sets of int. Combination leads to set union and the ordering is the normal superset relation – instead of the Smyth-ordering.

In MiniBrass:

```
type SetBasedMaxCsp = PVSType<bool, set of 1..nScs> =
  instantiates with "set-max-defs.mzn" {
    times -> union_violateds;
    is_worse -> super_set;
    top -> {};
  };
```

### 4.3.3 Real-Valued: Fuzzy CSP and Probabilistic CSP

A third class of soft constraint formalism is best characterized by the element type being the reals over $[0.0, 1.0]$. Starting with fuzzy constraints, each soft constraint maps to $[0.0, 1.0]$ with the combination being defined as the minimum operator – in this case, soft constraint type and element type coincide. Admittedly, floating-point decision variables are not among the strengths of the propagation-based solvers supporting MiniZinc, although interval-based constraint propagation and search exist [Benhamou and Granvilliers, 2006]. Nevertheless, we can model fuzzy constraint problems in MiniBrass as follows:

```
type FuzzyConstraints = PVSType<0.0 .. 1.0> =
  instantiates with "soft_constraints/mbr_types/fuzzy_type.mzn" {
    times -> min;
    is_worse -> is_worse_fuzzy;
    top -> 1.0;
};
```

The resulting soft constraints of element type `0.0 .. 1.0` could directly be defined as MiniZinc functions as we showed for cost function networks but MiniBrass provides some support with a global constraint that is included in `fuzzy_type.mzn`. For instance, consider a soft constraint $\mu_1$ defined over two boolean variables `mainCourse` (representing vegetarian or not) and `wine` (red or white): $\mu_1 = \{(0,0) \mapsto 0.3, (0,1) \mapsto 0.8, (1,0) \mapsto 1.0, (1,1) \mapsto 0.7\}$ (inspired by Rossi et al. [2008a]). This user is maximally satisfied with the combination "vegetarian" and "white wine" and least by "non-vegetarian" and "white wine". In MiniBrass:[3]

```
PVS: fz1 = new FuzzyConstraints("fz1") {
   soft-constraint mu1: 'fbinary_fuzzy([0.3, 0.8, 1.0, 0.7], mainCourse, wine)';
   soft-constraint mu2: 'fbinary_fuzzy([1.0, 0.8, 0.8, 1.0], mainCourse, wine)';
};
solve fz1;
```

On the other hand, probabilistic constraints bear similarities to both weighted and fuzzy constraints. We use `bool` as soft constraint type to denote violated constraints and again `0.0 .. 1.0` for probabilities as element type. Formally, the objective is $\prod_{\mu_i : \theta \not\models \mu_i} 1 - p_i$. The "constraint presence" probabilities $p_i$ are, analogously to weights, supplied as parameters.

```
type ProbabilisticConstraints = PVSType<bool, 0.0 .. 1.0> =
  params {
    array[1..nScs] of float: probs :: default('1.0');
  } in
  instantiates with "soft_constraints/mbr_types/probabilistic_type.mzn" {
    times -> prod;
    is_worse -> is_worse_prob;
    top -> 1.0;
};
[...]
% usage example
soft-constraint c2: 's1 + s2 >= 10' :: probs('0.7');
% which means "in 70% of all cases we expect that s1 + s2 must be greater than 10
% in 30% of all cases, no such restriction exists
```

[3] Note that the encoding employs a table constraint for floats which is not supported well by many solvers. Therefore a workaround using integers is also provided in the MiniBrass library (`fbinary_fuzzy_rational`).

Both fuzzy and probabilistic constraints aim at *maximization* of the solution degree such that `is_worse_prob(x,y)` corresponds to `x < y`. One can anticipate hybrid versions of weighted and probabilistic such as "minimizing the expected violation degree", given the "constraint presence" probabilities and penalties faced in case of violation. These probabilistic constraints, however, do not capture dependencies well and are only suited for simple probabilistic models.

## 4.4 Morphisms to Switch PVS

In addition to defining PVS types and instantiating them, PVS instances can be derived from existing ones by means of *morphisms*. There are at least two reasons why users would specify their *SCSP* using one PVS type but *solve* the problem using another: If the original PVS shows many incomparable optimal solutions, we might want or have to totalize the ordering – if only for testing and debugging. But perhaps more frequently, solvers that have to be used due to performance reasons or the target software environment do not support the data types required to represent a PVS type directly. For instance, set-based types for constraint preferences or real-valued domains with suitable global constraints for fuzzy constraints are not implemented equally well by all MiniZinc solvers. In such a case, a modeler would certainly only agree to transform the *SCSP in a structure-preserving way*: at least, existing strict "is better than" decisions in the original ordering are not to be contradicted; at most, incomparable solutions may become comparable – precisely what PVS-homomorphisms $\varphi : M \to N$ offer.

For example, when comparing Figure 3.2 and Figure 4.1, we see that the weighted version of the constraints (assigning $[3, 1, 1]$ to [`sharedNightShifts`, `nurseTwoNoNight`, `nurseThreeOff`]) is a consistent, monotone mapping. As explained in more detail in Section 5.1.2 and proved in Lemma 5.1, we can calculate a weight for each constraint (by making use of the instance parameters such as the supplied graph) to transform a constraint preferences problem into a weighted CSP instance. The simplest way would be to make sure a constraint has a higher weight than the highest of its predecessors (which would actually lead to $[2, 1, 1]$ in the above example). In MiniBrass, we first define a morphism

```
% defined in the MiniBrass library "defs.mbr"
morph ConstraintPreferences -> WeightedCsp: ToWeighted =
  params {
    k = 'mbr.nScs * max(i in 1..mbr.nScs) (mbr.weights[i])';
5   weights = calculate_cr_weights;
  } in id;
```

using a function that is applied to each original soft constraint expression (here just the identity `id`) and a parameter transforming function (here `calculate_cr_weights` in MiniZinc). The defined morphism can then be applied to a specific PVS instance:

```
PVS: cr1 = new ConstraintPreferences("cr1") {
  soft-constraint c1: 'x + 1 = y';
  soft-constraint c2: 'z = y + 2';
  soft-constraint c3: 'x + y <= 3';
5
  crEdges : '[| mbr.c2, mbr.c1 | mbr.c3, mbr.c1 |]';
  useSPD: 'true' ;
};
solve ToWeighted(cr1); % assigns weight 1 to c3 and c2, and 2 to c1
```

In this particular example of a morphism from constraint preferences to weighted CSP, delegating the calculation of weights directly to the MiniZinc compiler might take some time since the weight assignment proceeds recursively and cannot make use of precalculated results in a

dynamic programming (or memoization) style. Still, the weights are parametric information that can be determined before any actual solving occurs. Therefore, morphisms can refer to an external method in Java, similar to parameter processing methods:

```
morph ConstraintPreferences -> WeightedCsp: ToWeightedExt =
  params generatedBy('isse.mbr.extensions.weighting.SingleWeighting') {
%    k = 'mbr.nScs * max(i in 1..mbr.nScs) (mbr.weights[i]) ';
    k = generated ;
5   weights = generated ;
  } in id; % the "generated" keyword indicates that the external method sets those parameters
```

By devising similar morphisms for other PVS types, we can also integrate the previously mentioned fact that many soft constraint formalisms can be (monotonically) encoded as cost function networks in polynomial time [Schiex et al., 1995], the type for which Toulbar2 offers efficient dedicated algorithms. For example, a probabilistic PVS that has a multiplicative maximization objective $f(\theta) = \prod_{\mu_i : \theta \not\models \mu_i} 1 - p_i$ can be transformed into an additive minimization problem by taking the negative logarithm of $f$: $-\log f(\theta) = \sum_{\mu_i : \theta \not\models \mu_i} -(1 - \log p_i)$ where we can calculate the $1 - \log p_i$ terms as weights. Here again, we benefit from the fact that the transformation can be done in the more expressive Java language rather than in MiniZinc:

```
% a morphism converting a probabilistic CSP to weighted CSP using log
morph ProbabilisticConstraints -> WeightedCsp: ProbToWeighted =
  params generatedBy('isse.mbr.extensions.weighting.ProbWeighting') {
    k = 'mbr.nScs * max(i in 1..mbr.nScs) (mbr.weights[i])';
5   weights = generated;
  } in id;
```

The above-mentioned calculation here takes place in the Java class `ProbWeighting`. While this morphism definition is mathematically sound, we have to round the terms to the nearest integer in the implementation.

There is an interesting technical subtlety in terms of propagation when using morphisms. Since $r \leq_P p \rightarrow \varphi(r) \leq_Q \varphi(p)$ for a PVS-homomorphism $\varphi$, we can propagate $\varphi(r) \leq_Q \varphi(p)$ *in addition to* $r \leq_P p$, i.e., as a redundant constraint. This can, in some cases, speed up the solving process due to increased pruning and propagation (see Section 9.2).

### 4.4.1 Products of PVS

One particular advantage of using a PVS-based soft constraint language is their inherent modularity which is agnostic towards concrete types. We can form composite PVS from elementary ones by means of products. MiniBrass provides the keyword `pareto` for the direct (Cartesian) product and the keyword `lex` for the lexicographic product. We can combine these two operators and morphisms to form complex PVS. Consider these exemplary use cases:

```
solve cfn1 pareto cfn2;
solve cfn1 lex cfn2;
solve ToWeighted(cfn1) pareto (cfn2 lex cfn3);
```

Pareto and lexicographic combinations are an elementary tool to combine preference relations [Andréka et al., 2002] but especially with partial orders, these product orderings can become too indecisive for practical situations. For example, assuming a Pareto-combination of twenty agents' PVS deciding on a working schedule, an assignment is deemed better if and only if *all* agents agree that it is better or equally good. Otherwise, two assignments end up being incomparable. In reality, it is rarely the case that all agents uniformly agree on an is-better-relation – at least one agent could spoil things for all others.

Therefore, a very important and interesting application area of MiniBrass relies on *voting*, borrowed from the theory of social choice. Chapter 8 is devoted to the underlying principles. For instance,

```
solve vote([agent1,agent2,agent3], majorityTops);
```

searches for an assignment that maximizes the number of preference structures (here PVS representing individual agents) that map to the respective top value. Other possibilities for preference aggregation include `approval` (allowing only boolean PVS and maximizing the number of approving PVS) or `condorcet` (applying a pairwise elimination moving from assignment to assignment if a majority prefers the new one).

### 4.4.2 PVS-based Search

With the tools presented so far, we are able to define new PVS types or include them, instantiate PVS, and combine or morph PVS instances to more complex structures. The overall goal is the PVS passed in the `solve`-item. This goal is decoupled from the search algorithm employed to find optima since preference modelers would typically not be concerned with the "how" but only the "what" should be searched for. Only in a second implementation step, engineers could switch, e.g., to a local (large-neighborhood) search if branch-and-bound fails to timely find satisfactory solutions. For solving problems, MiniBrass relies on classical systematic constraint solving and optimization based on constraint propagation and search, as outlined at the beginning of Chapter 4. The necessary facilities are provided by MiniZinc (esp. the MiniSearch extension) and the underlying solvers to actually carry out the optimization. If the element type is numeric and a standard ordering ($\leq$ or $\geq$) is used, the problem can be solved in *MiniZinc* by minimizing (or maximizing) `topLevelObjective`. However, the full strength of abstract soft constraint formalisms shows in the presence of partial and product orders. *MiniSearch* offers blueprints for several classical searches that can be customized for MiniBrass. For example, consider branch-and-bound natural maximization for a numerical objective in MiniSearch:

```
% Branch and Bound when maximizing
function ann: bab_max(var int: obj) =
   repeat(
       if next() then
5        commit() /\
          post(obj > sol(obj))
       else break endif
   );
```

Intuitively, the algorithm proceeds as follows: "Fetch the next solution (satisfying all hard constraints). Then impose that the next possible solution has a higher `obj` value. Repeat this until you find no solution." Bounds are implicitly given by the upper and lower bounds on the possible values the generated overall valuation may take. If the solver deduces that no better valuation is possible, it can prune the search tree.

In fact, we can abstract from the specific predicate (`post(obj > sol(obj))`) to *any* predicate that indicates how a solution improves:

```
% Only declare minisearch function; implementation generated during MiniBrass compilation
function ann: postGetBetter();

% PVS-based branch-and-bound
5 function ann: pvs_BAB() = % ask for *strict* improvement
  repeat(if next()
         then print("Intermediate solution:") /\ print() /\ commit() /\
              postGetBetter()
```

```
         else break
10       endif);

% synonym for easier usage
function ann: miniBrass() = pvs_BAB();
```

MiniBrass is responsible for compiling the top level PVS passed in the `solve` item to a definition of the `postGetBetter` predicate which is only declared for `pvs_BAB`.

While the procedure `pvs_BAB` yields optimal solutions in the classical (totally ordered) sense, it is not ideal for partially ordered objectives since another optimum clearly does not have to be *better* than the current solution. Instead, it *must not already be dominated* by *any* solution seen so far [Junker, 2009]. When solving for a PVS $M$, we thus have a set of lower bounds (the valuations of previous solutions) $L = \{l_1, \ldots, l_m\} \subseteq |M|$ and require that it cannot be that $\exists l \in L : \texttt{obj} \leq_M l$ where `obj` is the generated MiniZinc variable that holds the overall objective. The next solution must be strictly better than any one of the maxima of $L$ or incomparable to all of them.

```
function ann: postNotGetWorse();

function ann: pvs_BAB_NonDom() = % ask not to be dominated by any previous solution
  repeat(if next()
5        then print("Intermediate solution:") /\ print() /\ commit() /\
          postNotGetWorse()
         else break
         endif);
```

We can generate a MiniSearch procedure "`postNotGetWorse`" during compilation as well from the `is_worse` predicates that PVS types offer, similar to `postGetBetter`. Still, there is a caveat to this solution. We have to *negate* this predicate, i.e., change its boolean context. This is problematic if the predicate shows free local variables [Stuckey and Tack, 2013]. We have seen this in Section 4.3.2 for the witness function necessary to decide the Smyth-ordering which is not compatible with `postNotGetWorse`. For constraint preferences, we have to resort to the TPD-ordering instead. Since we expect future non-trivial PVS types to rely on local variables, we need modelers to be aware of this restriction.

**Example 4.5 – Non-Domination versus Domination Search**
Consider the following simplified example to illustrate the difference:

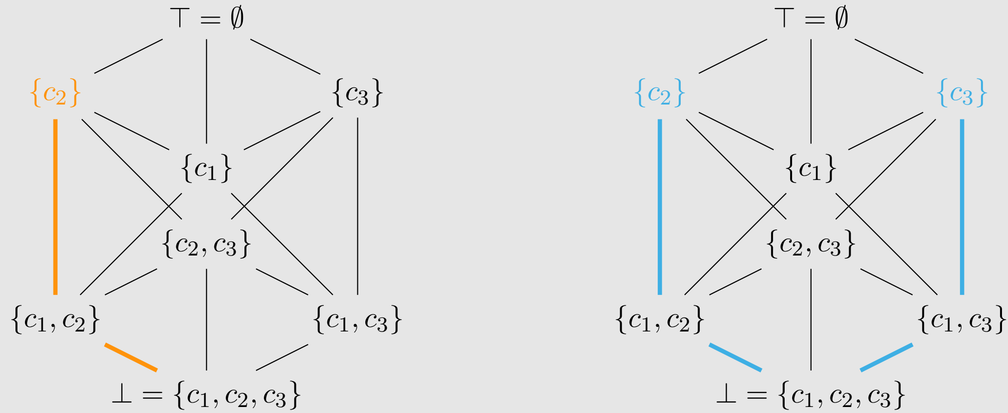
```
% In the classical constraint model:
var 1..3: x;
solve :: int_search([x], input_order, indomain_max, complete)
% traverses x = 3 then x = 2 then x = 1
5 search pvs_BAB_NonDom();

% In the preference model
PVS: cr1 = new ConstraintPreferences("cr1") {
  soft-constraint c1: 'x in {2,3}'; % violated iff x = 1
10 soft-constraint c2: 'x in {1,3}'; % violated iff x = 2
  soft-constraint c3: 'x in {1,2}'; % violated iff x = 3

  crEdges : '[| mbr.c2, mbr.c1 | mbr.c3, mbr.c1 |]';
  useSPD: 'false';
15 };

solve cr1;
```

It uses the familiar search space for set-based constraint preferences:




We explore x in a decreasing order. Each assignment to x violates precisely one soft constraint. This results in the sequence $\langle \{c_3\}, \{c_2\}, \{c_1\} \rangle$ of solution degrees. $\{c_3\}$ and $\{c_2\}$ both dominate $\{c_1\}$ but are incomparable using TPD-ordering (and Smyth, too). The reachable optima of this problem are clearly $\{\{c_2\}, \{c_3\}\}$ but `pvs_BAB` (left, orange) would stop after $\{c_2\}$ since $\{c_3\}$ is not better. By contrast, `pvs_BAB_NonDom` (right, blue) returns both optimal solution degrees.

MiniSearch actually offers much more flexibility in crafting problem-specific searches than just branch-and-bound. For instance, designing large-neighborhood-search (as described in Section 3.3.3) for PVS-based models can be done using their concepts of scopes, as described in Rendl et al. [2015].

```
% Adapted from lns_max an objective value
function ann: pvs_LNS(array[int] of var int: x,
                      int: iterations, float: d, int: exploreTime) =
  repeat (i in 1..iterations) (
    print("Starting iteration ... \(i)\n") /\
    scope(post(neighbourhoodCts(x,d)) /\
          time_limit(exploreTime, pvs_BAB()) /\ commit() /\
          print("Intermediate solution\n") /\ print()) /\ postGetBetter()
  );
```

In a similar way, we can anticipate many variants of search algorithms with `postGetBetter` or `postNotGetWorse`. By separating concerns between constraint and preference model, the preference model in MiniBrass can be tested with various searches.

## 4.5 Modeling Case Studies in MiniBrass

We conclude this chapter by demonstrating MiniBrass' abilities in two more involved examples, unit commitment, and mentor matching.

### 4.5.1 Unit Commitment

First of all, reconsider the scheduling example faced in (autonomous) virtual power plants presented in Section 2.1. This example most heavily influenced the idea of modular preference structures that are linked by lexicographic and Pareto-products, as well as voting operators.

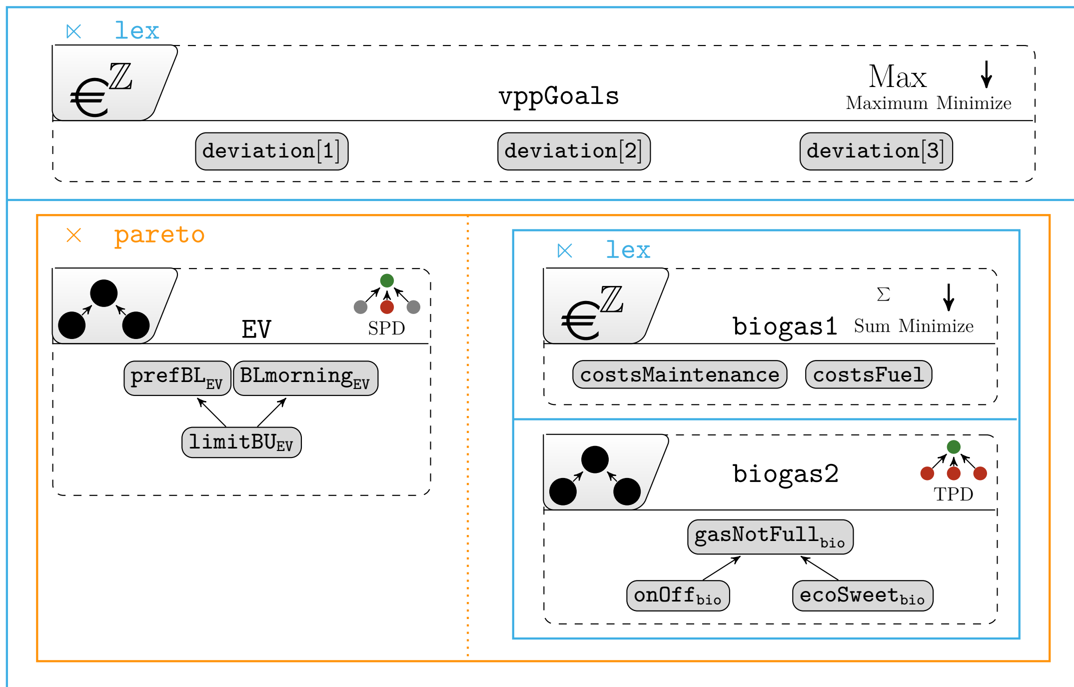



Figure 4.4: A graphical depiction of the overall complex preference structure for a simplified instance in the unit commit problem, as described in Example 1.1. Reprinted from Figure 1.1.

Consequently, it comes as no surprise that MiniBrass is well equipped to model situations such as those presented in Figure 1.1 which we reprint in Figure 4.4 for better readability.

We present the underlying constraint model, inspired by Figure 2.1. We assume that there is a finite scheduling horizon and the task is to only adapt the prosumers' power level as to match the existing demand (of course, in demand-side management for the EV, the demand is adjusted by allowing negative power values).

```
int: T = 5; set of int: WINDOW = 1..T;
array[WINDOW] of int: demand = [20, 21, 25, 30, 29];

int: P = 3; set of int: PROSUMERS = 1..P;

% only prosumer 2 (the EV) can consume
array[PROSUMERS] of int: pMin  = [12, -5, 7];
array[PROSUMERS] of int: pMax  = [15, 11, 9];

array[WINDOW, PROSUMERS] of var -5..15: power;
constraint forall(p in PROSUMERS, w in WINDOW)
     (power[w,p] in pMin[p]..pMax[p]);

array[WINDOW] of var int: deviation =
    [ abs( sum(p in PROSUMERS) (power[w, p]) - demand[w] ) | w in WINDOW];

solve search miniBrass();
```

This constraint model establishes the elementary decision variables for prosumer scheduling and defines minimal hard constraints – as it serves as an illustrative example. Real-world models are certainly more involved. Nevertheless, we are able to express the PVS displayed in Figure 1.1, beginning with the top-level (collective) objective:

```
PVS: orga = new CostFunctionNetwork("Orga") {
    soft-constraint vio_1: 'deviation[1]';
```

```
   soft-constraint vio_2: 'deviation[2]';
   soft-constraint vio_3: 'deviation[3]';
5  isWorstCase: 'true';
};
```

In MiniBrass cost function networks, cost minimization is default although we could define a PVS type that makes minimization or maximization parameterizable. Each soft constraint refers to the deviation between supplied power and demand at a certain point in time. We take the maximum deviation as aggregation operator since, e.g., a schedule that deviates $[0, 0, 9]$ is worse than one with deviations $[3, 3, 3]$. A better deviation at some points in time cannot make up for a larger deviation at another point in time. Recall that Chapter 7 was mostly concerned with PVS taking maximum as aggregation. Next, we can define the individual prosumers' PVS:

```
PVS: biogas = new ConstraintPreferences("biogas") {
   soft-constraint gasFull:
     'forall(w in WINDOW) (power[w,biogas] >= 13)';
   soft-constraint ecoSweet:
5    'forall(w in WINDOW) (power[w,biogas] >= 14)';
   soft-constraint onOff:
     'forall(w in 1..T-1) (
        abs(power[w,biogas] - power[w+1,biogas]) <= 1)';

10 crEdges : '[| mbr.ecoSweet, mbr.gasFull | mbr.onOff, mbr.gasFull |]';
   useSPD: 'true' ;
};

PVS: ev = new ConstraintPreferences("ev") {
15 soft-constraint prefBL: % never exceed 5 in production
     'forall(w in WINDOW) (power[w,ev] <= 5)';
   soft-constraint BLmorning: % no prosumption in the morning
     'power[1,ev] = 0)';
   soft-constraint limitBU: % reduce production from EV
20   'sum(w in WINDOW) ( bool2int(power[w,ev] > 0) ) < 2';

   crEdges : '[| mbr.limitBU, mbr.prefBL | mbr.limitBU, mbr.BLmorning |]';
   useSPD: 'false' ;
};
25
% thermal plant consists of two layers
PVS: therm1 = new CostFunctionNetwork("therm1") {
  soft-constraint ecoOpt:
    'sum(w in WINDOW) ( abs(power[w,thermal] - 8) )';
30 soft-constraint inertia:
    'sum(w in 1..T-1) ( abs(power[w,thermal] - power[w+1,thermal]))';
};

PVS: therm2 = new CostFunctionNetwork("therm2") {
35 soft-constraint ecoGood:
    'sum(w in WINDOW) ( abs(power[w,3] - 9) )';
};
```

The soft constraints are rather straightforward to define in this small example and we note two different PVS types. Attentive readers may notice that state and inertia information such as the battery level of the EV is not adequately represented in this constraint model. Indeed, there are more aspects involved when it comes to modeling a variety of prosumers such as the notorious binary on/off setting discussed in Section 2.1.2. Schiendorfer et al. [2015a] presented *supply automate* that are automatically translated into constraint models which are able to capture such system behavior. For the sake of brevity, however, we omit to discuss them here. Finally, we can tell MiniBrass the overall goal by means of lexicographic and Pareto-products:

```
solve orga lex ( biogas pareto ev pareto (therm1 lex therm2) );
```

We can see that most structural information (including PVS types and parameters) is preserved in the graphical representation of the preference model in Figure 4.4. Only the (formal) definition of soft constraints with respect to the underlying constraint model needs to be

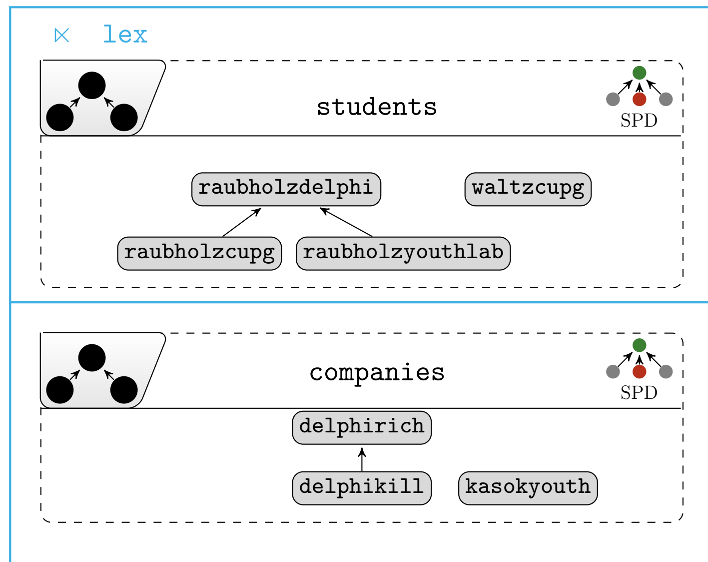



Figure 4.5: A graphical representation of the mentor matching problem modeled with two PVS representing students and companies

added. This step can be done independently of (re)-prioritizing the soft constraints and PVS. Related to models such as this example, we investigated the influence of the selected dominance property of constraint preferences in [Schiendorfer et al., 2014a]. As can be expected, TPD semantics had more (unimportant) constraints violated than SPD. In particular, TPD led to an average dissatisfaction of about 40% of all soft constraints, whereas SPD only dissatisfied 30%. This shows that a relevant question for preference elicitation and requirements engineering is to find out whether the problem's constraints are more hierarchical or egalitarian.

Although MiniBrass is well-suited for the unit commitment problem, it is certainly not a tool & theory crafted for a single problem. In addition to the subsequent example of mentor matching, several practical problems show different aspects of MiniBrass throughout the dissertation. Chapter 9 also provides a diverse set of standardized benchmark problems.

### 4.5.2 Mentor Matching

Next, we present a MiniBrass model for mentor matching in a software engineering graduate program, as described in Section 2.3.2. Our main task is to assign every student to a company acting as their mentor. To ensure fairness among the participating companies, there are cardinality constraints restricting the minimal and maximal number of mentees per company. The constraint model reflects this as follows:

```
int: n; set of int: STUDENT = 1..n;
int: m; set of int: COMPANY = 1..m;

% assign students to companies
array[STUDENT] of var COMPANY: worksAt;

int: minPerCompany = 1; int: maxPerCompany = 3;
constraint global_cardinality_low_up (
           worksAt, [c | c in COMPANY],
           [minPerCompany | c in COMPANY],
           [maxPerCompany | c in COMPANY]);
```

```
solve search miniBrass();
```

On top of this model, we can have one PVS for the students and one for the companies. Students submitted their wishes as a ranked list. Figure 4.5 depicts the overall preference model, with the soft constraints being specified as follows:

```
PVS: students = new ConstraintPreferences("students") {
   soft-constraint raubholzdelphi: 'worksAt[raubholz] = delphi';
   soft-constraint raubholzyouthlab: 'worksAt[raubholz] = youthlab';
   soft-constraint raubholzcupg: 'worksAt[raubholz] = cupgainini';
5  soft-constraint waltzcupg: 'worksAt[waltz] = cupgainini';

   % only intra-student preferences are allowed, no inter-student preferences
   crEdges : '[| mbr.raubholzyouthlab, mbr.raubholzdelphi |
                 mbr.raubholzcupg, mbr.raubholzdelphi |]';
10 useSPD: 'true' ;
};

PVS: companies = new ConstraintPreferences("companies") {
   soft-constraint delphi_kill: 'worksAt[kill] = delphi';
15 soft-constraint delphi_rich: 'worksAt[rich] = delphi';
   soft-constraint kasokyouth: 'worksAt[kasok] = airtrain';

   crEdges : '[| mbr.delphi_kill, mbr.delphi_rich |]';
   useSPD: 'true' ;
20 };
```

We can formulate the overall objective with a lexicographic product that either prioritizes the students or companies. To illustrate the difference, we employ a morphism to convert constraint preferences to weighted constraints, as described in Section 4.4. For instance, stating

```
solve ToWeighted(students) lex ToWeighted(companies);
```

could lead to

```
Intermediate solution:worksAt = [1, 1, 1, 2, 3]
Valuations: pen_companies = 2; pen_students = 1
----------
==========
```

which assigns two penalty points to the companies and one to the students whereas

```
solve ToWeighted(companies) lex ToWeighted(students);
```

would lead to

```
Intermediate solution:worksAt = [1, 2, 1, 1, 3]
Valuations: pen_companies = 0; pen_students = 2
----------
==========
```

which optimally satisfies the companies but yields higher costs for the students. Having interpretable cost values clearly is beneficial, although using a single instance of constraint preferences for different students is not ideal. We face the problem of introducing bias, as explained in Chapter 5.

Besides this illustrative toy problem, we experimented with the model for the mentor assignment in the winter term 2015. There were 16 students to be assigned to seven companies which leads to a search space of $7^{16} = 3.3233 \cdot 10^{13}$ assignments. There were 77 soft constraints submitted by the 16 students and 37 soft constraints submitted by the seven companies, totaling 114 soft constraints. The proven optimum (lexicographically favoring students) was found in six minutes and successfully replaced the manual solution proposed by the graduate program's coordinator.

**Chapter Summary and Outlook**

In this chapter, we presented the core elements of the MiniBrass language for soft constraint programming on top of MiniZinc and MiniSearch. Based on partial valuation structures (PVS) as its basic building block, it enables modular and concise expression of soft constraints of various types. New PVS types can be declared with a mapping for the element type, the combination operation, and the ordering. We presented the most common soft constraint formalisms in the literature as MiniBrass PVS types and highlighted their strengths and weaknesses. Finally, we highlighted how to apply MiniBrass concepts in two of our application scenarios.

Chapter 6 explains why we used PVS as the underlying algebraic structure, in the first place. It can be seen as the discussion of MiniBrass' semantics. Chapter 8 provides more background information on the implemented social choice functions to enable voting among several PVS and Chapter 5 revisits the rationale underlying the PVS type "constraint preferences".

Chapter **5**

# Constraint Preferences for Soft Constraints

**Summary.** To elicit preferences in a qualitative (as opposed to quantitative) way first, we devised a novel soft constraint formalism, called constraint preferences, that relies on a partial order over constraints to denote their relative importance. This relieves us from making constraints of different agents comparable. If not all soft constraints can be satisfied, we aim to achieve as many (important ones) as possible. We present several ways of specifying the dominance semantics as to *how* much more important a soft constraint is related to its less important counterparts – which results in different liftings to orderings in violation sets. Each of them can be transferred to a weighted representation resulting in a numeric objective function or be optimized directly according to the chosen lifted ordering in MiniBrass.

**Publication.** The concepts and results outlined in this chapter have been published in Schiendorfer et al. [2013]; Knapp et al. [2014].

Most of the conventional preference formalisms discussed in Section 3.2 are quantitative in nature. In weighted constraints, each soft constraint is assigned a penalty integer, numeric cost functions are minimized or maximized, or a satisfaction degree ranging from 0 to 1 is maximized in fuzzy constraints. However, there are good reasons not to start with the numeric objective in the first place. Consider again the solution trace from Example 4.1 that is obtained from executing the left, weight-based preference model in Figure 5.1:

```
Intermediate solution: n = [night, day, day]
Valuations:  topLevelObjective = 4
----------
Intermediate solution: n = [night, night, day]
Valuations:  topLevelObjective = 2
----------
Intermediate solution: n = [night, day, night]
Valuations:  topLevelObjective = 1
----------
==========
```

While we obviously see the `topLevelObjective` decreasing in this minimization problem, the intuition *why* an assignment dominates another one can become rather obfuscated instead

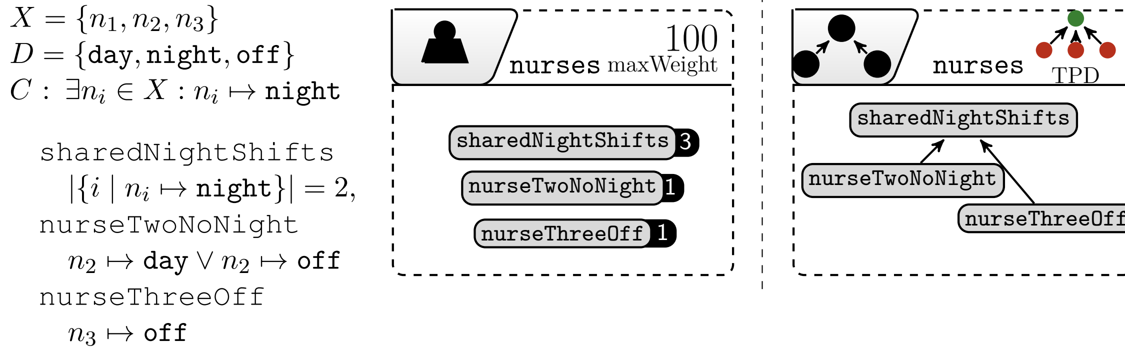



Figure 5.1: Two preference models based on one constraint model (left). Either constraints are assigned weights (quantitatively) or importance relations are given qualitatively.

of explanatory. Perhaps, a user would be more convinced by the following trace:

```
Intermediate solution: n = [night, day, day]
Valuations:  violatedSoftConstraints = { sharedNightShifts, nurseThreeOff }
----------
Intermediate solution: n = [night, night, day]
Valuations:  violatedSoftConstraints = { nurseTwoNoNight, nurseThreeOff }
Better since it only violates 'nurseTwoNight' instead of 'sharedNightShifts'
----------
Intermediate solution: n = [night, day, night]
Valuations:  violatedSoftConstraints = { nurseThreeOff }
Better since it does no longer violate 'nurseTwoNoNight'
----------
==========
```

We argue that the latter trace can be much easier to understand, in particular, since the underlying rationale is revealed and preserved. Kaci et al. [2014] point out that comparative statements may be easier to cognitively process for users. A suitable weighting function, such as the one used in Figure 5.1, quantifies an inherently qualitative relationship such as "`sharedNightShifts` is more important than `nurseTwoNight`". Moreover, in Section 4.4 we have discussed that we can automatically convert any constraint preferences instance to appropriate weights using a morphism, i.e., a monotone function (see Lemma 5.1).

Conversely, from a practical engineering perspective, consider the situation where a modeler would want to add a number of soft constraints that should be even less important than **`nurseTwoNoNight`** and **`nurseThreeOff`**. Had we modeled the preferences directly in weighted constraints, we would have to increase the weights of all soft constraints to reflect the new ordering. If the constraint and preference model is automatically compiled at runtime (e.g., two graphs are combined below some overall goals), we also benefit from a qualitative ordering before we generate weights – if necessary.

Moreover, the totalization that follows from weighting partially-ordered soft constraints makes incomparable elements comparable (such as that **`nurseTwoNoNight`** and **`nurseThreeOff`** take the *same* weight 1) or, even worse, elements from incomparable branches become ordered, as Figure 5.2 shows: There should not be a preference relationship between **`bestA`** and **`bestB`** but the solver would be biased to favor satisfying **`bestA`**.

To sum up, we identify the following motivations to specify preferences qualitatively:

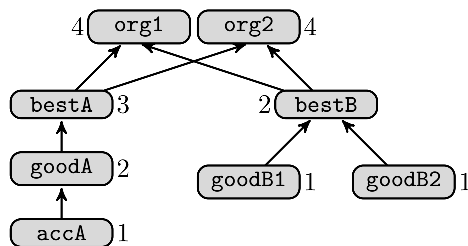



Figure 5.2: A constraint preferences graph with associated valid weights for a set of soft constraints. Assume that the subgraphs beneath `bestA` and `bestB` stem from different sources and are assembled to the overall graph at runtime.

- Better explanations
- Less cognitive overhead
- No weight-introduced bias
- Less refactoring effort

Although the previous chapters in part relied on some knowledge about constraint preferences as published in [Schiendorfer et al., 2013; Knapp et al., 2014] (then called *constraint relationships*), we present their underlying design intentions more explicitly in this chapter.

## 5.1 Qualitative Specification using Constraint Preferences

The syntactic features of constraint preferences are inherently simple. A set of constraint preferences over the soft constraints $C_\mathrm{s}$ refers to a relation $\rightarrow_P \subseteq C_\mathrm{s} \times C_\mathrm{s}$ that is a directed acyclic graph. By stating $\mu_1 \rightarrow_P \mu_2$, users express that soft constraint $\mu_1$ is *less important* to them than $\mu_2$. We can immediately augment this graph to a partial ordering over constraints by taking the reflexive-transitive closure $\rightarrow_P^*$ (also, the transitive closure $\rightarrow_P^+$ appears if we explicitly need to exclude reflexive edges). Hence, we would describe the constraint preferences from Figure 5.1 as $\{\texttt{nurseTwoNoNight} \rightarrow_P \texttt{sharedNightShifts}, \texttt{nurseThreeOff} \rightarrow_P \texttt{sharedNightShifts}\}$ and add the missing reflexive and transitive edges implicitly. By convention, we refer to $\mu_1$ as a "predecessor" of $c_2$, i.e., a less important soft constraint.

### 5.1.1 Semantics of Dominance Properties

So far, constraint preferences syntactically simply express that a soft constraint is more important than another one. Assignments, however, satisfy and violate *sets* of soft constraints. To grade them, it is important to address the question of *how* much more important a single soft constraint is with respect to its less important predecessors. From an *egalitarian* standpoint, a constraint should not be much more important than another one, whereas a *hierarchical* setting could judge a single soft constraint to be much more important than even a set of predecessors. Technically, we need a lifting of the ordering $\rightarrow_P^*$ over soft constraints to an ordering $\sqsubseteq_P$ over sets of violated constraints, called a *violation set*. For an assignment $\theta$ in an *SCSP*, its violation set is simply $V(\theta) = \{\mu \in C_\mathrm{s} \mid \theta \not\models \mu\}$. We read $T \sqsubseteq_P U$ as "$T$ is

worse than $U$, given $P$" to be consistent with the PVS-specific notation used throughout other chapters.[1]

**Example 5.1 – Lifting by Weighting**
One possible lifted ordering is obtained by assigning every soft constraint a weight as we have seen for Figure 5.1 and sum up the violated constraints' weights: For sets $T, U \subseteq C_\mathrm{s}$ and weighting $w : C_\mathrm{s} \to \mathbb{N}$ that is monotone with respect to $P$:

$$T \sqsubseteq_P U \Leftrightarrow \sum_{t \in T} w(t) \geq \sum_{u \in U} w(u)$$

With respect to Figure 5.2, $\{\mathtt{bestA}, \mathtt{goodA}\} \sqsubseteq_P \{\mathtt{goodB1}\}$ since $5 > 1$, for instance.

In the example above, we clearly introduce many "strange" ordering relations between unrelated sets that are not logically justifiable. We therefore ask for meaningful semantics that we ascribe to constraint preferences and begin with minimal requirements. We will characterize the orderings inductively as rule-based specifications. While in this chapter we follow an "intuitive" path, Chapter 6 approaches this question from a purer, category-theoretical angle.

What are the ordering relations over violation sets that we can assume from a given partial ordering over soft constraints? The only conclusion we can safely draw from a constraint preference $\mu_1 \to_P \mu_2$ is that $\{\mu_2\} \sqsubseteq_P \{\mu_1\}$ should hold since it should be *worse* to violate the *more important* soft constraint $\mu_2$. Similarly, if both violate the same other soft constraints $T \subseteq C_\mathrm{s}$ with $\mu_1, \mu_2 \notin T$, we would prefer a solution that violates the less important $\mu_1$, i.e., $T \cup \{\mu_2\} \sqsubseteq_P T \cup \{\mu_1\}$. Hence, the second case reduces to the first one if $T$ is empty.

Moreover, since we define an ordering on *violation* sets, having a larger set is inherently worse. If we can choose an assignment that only violates a subset of another one, we should always choose the first one. Formally, $T \cup \{\mu_1\} \sqsubseteq_P T$, still assuming $\mu_1 \notin T$. We obtain a partial ordering over sets if we apply both rules transitively. In fact, these two rules provide the semantics that a soft constraint can dominate a single other soft constraint but no more. With respect to Figure 5.2, $\{\mathtt{goodB1}, \mathtt{goodB2}\} \sqsubseteq_P \{\mathtt{bestB}\}$ holds since $\{\mathtt{goodB1}, \mathtt{goodB2}\} \sqsubseteq_P \{\mathtt{goodB1}\}$ by the first rule and $\{\mathtt{goodB1}\} \sqsubseteq_P \{\mathtt{bestB}\}$ by the second rule. On the other hand, neither $\{\mathtt{bestB}\} \sqsubseteq_P \{\mathtt{bestA}\}$ nor $\{\mathtt{bestA}\} \sqsubseteq_P \{\mathtt{bestB}\}$ is valid, hence these two sets are incomparable. We cannot establish an "is-worse-than" relation between the sets $\{\mathtt{org1}, \mathtt{goodB1}\}$ and $\{\mathtt{bestB}, \mathtt{goodB2}\}$ either: While a violation of `org1` can be "traded" for either `bestB` or `goodB2`, it cannot make up for both at the same time and `goodB1` cannot be exchanged for `goodB2` (incomparable) or `bestB` (a more severe violation). This feature is responsible for the dominance property's name "single-predecessor-dominance". The resulting ordering (that we write as $\preceq_P$, as before, or $\preceq^P_{\mathrm{SPD}}$ if we want to make the dominance property explicit) is referred to as the Smyth-ordering due to the eponymous ordering being used in powerdomain constructions.

$$\begin{aligned} &T \uplus \{\mu\} \prec^P_{\mathrm{SPD}} T \\ &T \uplus \{\mu\} \prec^P_{\mathrm{SPD}} T \uplus \{\mu'\} \quad \text{if } \mu' \to^+_P \mu \end{aligned} \tag{SPD}$$

The proof that $\preceq^P_{\mathrm{SPD}}$ is indeed a partial ordering relation over sets follows from a more general proof that we present for multiset-based PVS in Section 6.3. Still, we present a shorter proof

[1] The symbol $\sqsubseteq_P$ refers to a generic ordering whereas specific orderings will have their dedicated symbols.

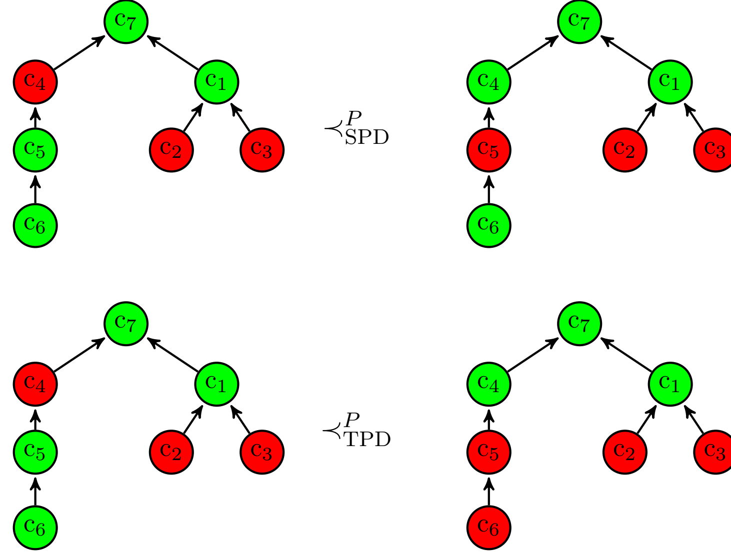



Figure 5.3: Single-predecessor-dominance (SPD) allows trading a single constraint; transitive-predecessors-dominance (TPD) makes a constraint more important than the set of all its predecessors. Left and right side represent violation sets of two different solutions. In the upper situation, the left is worse since the more important $c_4$ is violated, the rest being equal. In the lower situation, satisfying $c_4$ is even more valuable than both $c_5$ and $c_6$ which would *not* be true for SPD.

for the set-based ordering alone. Since we apply these rules inductively to obtain the full partial ordering $\preceq^P_{\text{SPD}}$, transitivity follows immediately. Similarly, reflexivity is obtained by the reflexive closure. It remains to show antisymmetry: As we will show in Section 5.1.2, there exists a strictly monotone mapping $w^P_{\text{SPD}} : 2^{C_s} \to \mathbb{N}$ such that $T \prec^P_{\text{SPD}} U \Rightarrow w^P_{\text{SPD}}(T) > w^P_{\text{SPD}}(U)$. For antisymmetry, assume two sets $T$, $U$ such that both $T \preceq^P_{\text{SPD}} U$ and $U \preceq^P_{\text{SPD}} T$ hold. We need to show that $T = U$ must hold. If $T \preceq^P_{\text{SPD}} U$ and $U \preceq^P_{\text{SPD}} T$ hold with equality, we are done, hence we focus on the case where $T \prec^P_{\text{SPD}} U$ and $U \prec^P_{\text{SPD}} T$. But then, we would simultaneously have $w^P_{\text{SPD}}(T) > w^P_{\text{SPD}}(U)$ and $w^P_{\text{SPD}}(U) > w^P_{\text{SPD}}(T)$ which cannot happen due to the totality of $<$ in the natural numbers. Hence, $\preceq^P_{\text{SPD}}$ is indeed a partial ordering relation.

But what about situations where modelers would indeed like the previously seen relation, $\{\texttt{org1}, \texttt{goodB1}\}$ is worse than $\{\texttt{bestB}, \texttt{goodB2}\}$, to hold, i.e., a single constraint's violation is strictly more detrimental than those of several or all of its predecessors? For this purpose, one may turn to the so-called *transitive-predecessors-ordering* (TPD), introduced by Schiendorfer et al. [2013], that defines a more important constraint to dominate a set of less important ones:

$$T \uplus \{\mu\} \prec^P_{\text{TPD}} T$$
$$T \uplus \{\mu\} \prec^P_{\text{TPD}} T \uplus \{\mu_1, \ldots, \mu_n\} \quad \text{if } \mu_i \to^+_P \mu \text{ for all } 1 \leq i \leq n \qquad \text{(TPD)}$$

Reflexivity, antisymmetry, and transitivity are shown using analogous arguments to SPD. Clearly, if we can establish $T \prec^P_{\text{SPD}} U$, we also have $T \prec^P_{\text{TPD}} U$ since the SPD rule is a special case of the TPD rule. That means, TPD only *adds* ordering relations to the existing ones in SPD. Section 6.1 actually helps us to formalize this distinction between "necessary" and "arbitrary" ordering relations. However, for adequately modeling a user's preferences,

additional well-defined relations may certainly be offered. Furthermore, since the rule $T \uplus \{\mu\} \prec^P_{\text{xPD}} T$ is present in both properties, an obvious consequence follows immediately:

$$T \supset U \Rightarrow T \prec^P_{\text{xPD}} U \tag{XPD}$$

This subsumes a set-based definition of Max-CSP. Figure 5.3 visualizes the difference between SPD and TPD.

### 5.1.2 Transforming Constraint Preferences to Weighted Constraints

Once constraint preferences and the desired dominance property are specified, we obtain a valid partial ordering on violation sets that can be used for optimization, e.g., in MiniBrass. Still, there are situations where users might prefer a numerical objective function – e.g., if a solver/system can only optimize according to an integer expression but is otherwise well-equipped for the problem class at hand (cf. Section 4.4 or Section 9.2). Then, we can derive a strictly monotonic conversion of constraint preferences to a weighted constraint representation, i.e., we assign a weight to every soft constraint according to the given relational preferences.

Formally, for an *SCSP* with soft constraints $C_{\text{s}}$ and preference graph $\rightarrow_P$, we devise weighting functions $w^P_{\text{xPD}}(U) : C_{\text{s}} \rightarrow \mathbb{N} \setminus \{0\}$ which we prove to be strictly monotonic. That means for any two violation sets $T$ and $U$, we have $T \prec^P_{\text{xPD}} U \Rightarrow W^P_{\text{xPD}}(T) > W^P_{\text{xPD}}(U)$ where $W^P_{\text{xPD}}(T) = \sum_{\mu \in T} w^P_{\text{xPD}}(\mu)$. If a violation set is judged worse by lifted constraint preferences, it will get a higher accumulated weight. Clearly though, the total order $(\mathbb{N}, \geq)$ is not able to express partiality which necessarily results in incomparable violation sets becoming comparable – as mentioned in the introduction to this chapter.

### 5.1.3 Concrete Weight Functions

Each weight function $w^P_{\text{xPD}}$ will be defined recursively based on the weights of its predecessors w.r.t. $\rightarrow_P$. This is well-defined since $C_{\text{s}}$ and thus $\rightarrow_P$ are finite and $\rightarrow_P$ is acyclic. Algorithmically, the less important constraints get their weights assigned *before* more important ones which can be done in a bottom-up fashion in a depth-first traversal.

Since $\prec^P_{\text{xPD}}$ is defined inductively by rules, it suffices to demonstrate strict monotonicity for each rule application. More precisely, if we have $T \prec^P_{\text{xPD}} U$ then there is actually a sequence $T = T_1 \prec^P_{\text{xPD}} \ldots \prec^P_{\text{xPD}} T_n \prec^P_{\text{xPD}} U$ of length $n \geq 1$ where each pair $(T_i, T_{i+1})$ is obtained by applying either of the previous rules.

We begin by examining the "superset" rule that is shared by both SPD and TPD which states that $T \uplus \{\mu\} \prec^P_{\text{xPD}} T$; and indeed, $w^P_{\text{xPD}}(T \uplus \{\mu\}) > w^P_{\text{xPD}}(T)$, since all weights are in $\mathbb{N} \setminus \{0\}$. We need to proceed to prove the strict monotonicity of (SPD) and (TPD).

**Single predecessor dominance.**
Since in SPD, a soft constraint only has to exceed any single predecessor, we propose to use the maximum weight of its predecessors and add 1 (by convention, $\max(\emptyset) = 0$).

$$w^P_{\text{SPD}}(\mu) = 1 + \max\{w^P_{\text{SPD}}(\mu') \mid \mu' \in C_{\text{s}} : \mu' \rightarrow_P \mu\} \quad \text{for } \mu \in C_{\text{s}} \ .$$

This mapping is indeed strictly monotonic for applications of (SPD): Assume $T \uplus \{\mu\} \prec^P_{\text{SPD}} T \uplus \{\mu'\}$ because $\mu' \rightarrow_P \mu$. By the above definition, $w^P_{\text{SPD}}(\mu) > w^P_{\text{SPD}}(\mu')$ and hence:

$$W^P_{\text{SPD}}(T \uplus \{\mu\}) = W^P_{\text{SPD}}(T) + w^P_{\text{SPD}}(\mu) > W^P_{\text{SPD}}(T) + w^P_{\text{SPD}}(\mu') = W^P_{\text{SPD}}(T \uplus \{\mu'\})$$

**Transitive predecessors dominance.** For TPD, the most straightforward choice is to take the sum of weights of all (transitive) predecessors and add 1. Even if all predecessors are violated, the violation of the more important constraints is higher than their sum:

$$w^P_{\mathrm{TPD}}(\mu) = 1 + \sum_{\mu' \to^+_P \mu} w^P_{\mathrm{TPD}}(\mu') \quad \text{for } \mu \in C_{\mathrm{s}} \ .$$

Assume that there are sets $\{\mu_1, \ldots, \mu_n\}$, $\{\mu'\}$, and $T$ such that $T \uplus \{\mu'\} \prec^P_{\mathrm{TPD}} T \uplus \{\mu_1, \ldots, \mu_n\}$ holds, i.e., $\mu_i \to_P \mu$ for all $1 \leq i \leq n$. Then, by the above definition of $w^P_{\mathrm{TPD}}$, the weight of $\mu'$ is still strictly higher than $W^P_{\mathrm{TPD}}(\{\mu_1, \ldots, \mu_n\})$ which proves the strict monotonicity since $T$ contributes equally to both sides.

However, this definition has to visit every transitive predecessor explicitly and repeatedly. We can exploit the graph structure of the preference specification to reduce the computational steps by only visiting immediate predecessors (referring to $\to_P$ instead of $\to^+_P$). In essence, since the weight of each soft constraint encodes the summed weight of all its predecessors, we can double this value to achieve the necessary growth. This slightly changes the definition:

$$w^P_{\mathrm{TPD}}(\mu) = 1 + \sum_{\mu' \to_P \mu} 2 \cdot w^P_{\mathrm{TPD}}(\mu') - 1 \quad \text{for } \mu \in C_{\mathrm{s}} \ .$$

We can establish that $w^P_{\mathrm{TPD}}(\mu) > \sum_{\mu' \to^+_P \mu} w^P_{\mathrm{TPD}}(\mu')$ with this definition by induction over the number $k$ of transitive predecessors of $\mu$: If $k = 0$, then $w^P_{\mathrm{TPD}}(\mu) = 1$ and thus the claim holds. If $k > 0$, assume that $w^P_{\mathrm{TPD}}(\mu') \geq 1 + \sum_{\mu'' \to^+_P \mu'} w^P_{\mathrm{TPD}}(\mu'')$ holds for all $\mu' \in C_{\mathrm{s}}$ with $\mu' \to^+_P \mu$. Then

$$\begin{aligned} w^P_{\mathrm{TPD}}(\mu) &= 1 + \sum_{\mu' \to_P \mu} (2 \cdot w^P_{\mathrm{TPD}}(\mu') - 1) \\ &= 1 + \sum_{\mu' \to_P \mu} (w^P_{\mathrm{TPD}}(\mu') + (w^P_{\mathrm{TPD}}(\mu') - 1)) \\ &\geq 1 + \sum_{\mu' \to_P \mu} (w^P_{\mathrm{TPD}}(\mu') + \sum_{\mu'' \to^+_P \mu'} w^P_{\mathrm{TPD}}(\mu'')) \\ &\geq 1 + \sum_{\mu' \to^+_P \mu} w^P_{\mathrm{TPD}}(\mu') \end{aligned}$$

where the last inequation holds since every summand of the last line is a summand of the next-to-last line but could be counted more than once if it is a predecessor of multiple soft constraints.

For completeness and future reference, we also present a third, intermediate type of semantics called "direct-predecessors-dominance" (DPD) that has been presented by Schiendorfer et al. [2013]. It essentially does not rely on transitiveness of $\to_P$ and assigns smaller weights than TPD but larger ones than SPD by stating that a soft constraint's weight should exceed the sum of its *immediate* predecessors:

$$w^P_{\mathrm{DPD}}(\mu) = 1 + \sum_{\mu' \to_P \mu} w^P_{\mathrm{DPD}}(\mu') \quad \text{for } \mu \in C_{\mathrm{s}} \ .$$

Obviously, if $\to_P$ is transitively closed, DPD and TPD are identical. Otherwise it is possible that a large enough set of (transitively) violated predecessors may outweigh a more important soft constraint which would be impossible in TPD. This leaves the semantics harder to interpret than for SPD and TPD which is why we presented no set-based ordering $\prec^P_{\mathrm{DPD}}$. Both of them assume transitivity but address the "egalitarian-hierarchical" conflict from an extremal perspective. DPD, on the other hand, offers a numeric compromise between the two and can be seen as a heuristic alternative to SPD and TPD. Figure 5.4 provides numeric examples.

In summary, we have

**Theorem 5.1.** *If* $T \prec^{P}_{\mathrm{xPD}} U$*, then* $W^{P}_{\mathrm{xPD}}(T) > W^{P}_{\mathrm{xPD}}(U)$ *for* $\mathrm{xPD} \in \{\mathrm{SPD}, \mathrm{TPD}\}$. □

In the next section, we present a toy example inspired by a ski-day planning application. In MiniBrass, recall that these weight functions are encapsulated and provided by the morphism `ToWeighted` that takes a `ConstraintPreferences` instance to a `WeightedCSP` instance.

## 5.2 Illustrating Constraint Preferences: A Ski-Day Planner

To get a better intuition about how preference specifications with constraint preferences work in practice, we review a typical simplified real-world scenario – originally presented in [Schiendorfer et al., 2013]. First, we show how varying sets of preferences lead to different preferred assignments. Then we discuss how changing constraints or preferences can be included more conveniently than with pure weighted constraints.

### 5.2.1 Personas & Preferences in the Ski-Day Example

Consider an application that provides guidance to travelers exploring a new ski area by offering a plan for a ski day. The problem might include real-time lift occupancy data as well as weather forecasts and snow conditions that result in a different set of valid itineraries. Each skier has different priorities that can be set interactively.

Assume the following soft constraints are defined on the set of possible itineraries (that need to respect hard constraints such as weather induced blockages, valid lift connections, timing issues, restaurant opening hours, and daylight time):

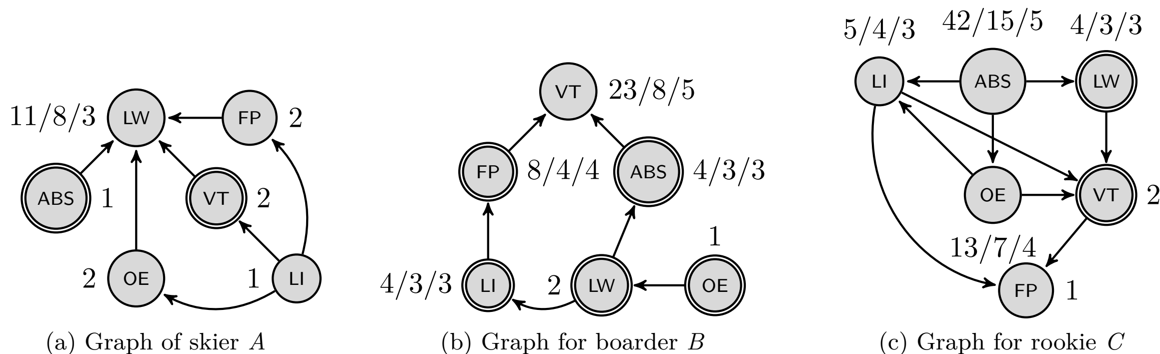
- Avoid black slopes (ABS): Beginners might want to avoid difficult (marked "black") slopes.
- Variety (VT): Different slopes need to be explored rather than staying on the same tracks.
- Fun-park (FP): Freestyle fans want to include a fun-park.
- Little Wait (LW): Impatient visitors prefer not to wait too long at a lift.
- Only Easy Slopes (OE): People can restrict their tours to easy ("blue") slopes only.
- Lunch Included (LI): Whereas some travelers enjoy the comfort of a good mountain dish, others prefer to spend their day out without longer rest.

For clarity and focus, we abstract from actual constraint models as well as the structure of the assignments and leave hard constraints aside by assuming the following three assignments satisfy all hard constraints but differ in their performance on soft constraints.

- $\theta_1 \models \{\mathsf{LW}, \mathsf{OE}, \mathsf{LI}, \mathsf{FP}\} \wedge \theta_1 \not\models \{\mathsf{VT}, \mathsf{ABS}\}$
- $\theta_2 \models \{\mathsf{VT}\} \wedge \theta_2 \not\models \{\mathsf{FP}, \mathsf{ABS}, \mathsf{LW}, \mathsf{LI}, \mathsf{OE}\}$
- $\theta_3 \models \{\mathsf{OE}, \mathsf{FP}, \mathsf{ABS}, \mathsf{LI}\} \wedge \theta_3 \not\models \{\mathsf{VT}, \mathsf{LW}\}$

Assume three personas as prototypical customers.

- Skier $A$ is rather impatient, skilled in skiing, wants to explore a fun-park but is not particularly afraid of difficult slopes or needs lunch since she perceives skiing primarily as physical activity.




(a) Graph of skier $A$ (b) Graph for boarder $B$ (c) Graph for rookie $C$

Figure 5.4: The constraint preferences for each persona. Double borders indicate that this constraint was violated in the TPD-preferred assignment according to Figure 5.5. Weights are printed for TPD/DPD/SPD, only one number indicates that the weights are equal for all dominance semantics.

| | Skier $A$ | Boarder $B$ | Rookie $C$ |
|---|---|---|---|
| $\theta_1$ | **3** | 8 | 7 |
| $\theta_2$ | 9 | 13 | 16 |
| $\theta_3$ | 5 | **7** | **5** |

(a) SPD semantics

| | Skier $A$ | Boarder $B$ | Rookie $C$ |
|---|---|---|---|
| $\theta_1$ | **3** | 27 | 44 |
| $\theta_2$ | 17 | **19** | 65 |
| $\theta_3$ | 13 | 25 | **6** |

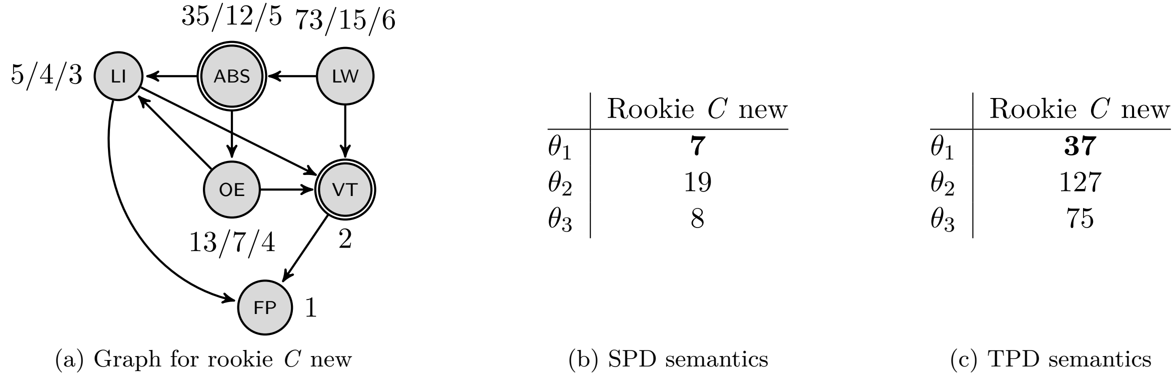
(b) TPD semantics

Figure 5.5: Different tours rated by different constraint preferences. As can be seen, DPD returns intermediately scaled weights compared to SPD and TPD.

- Boarder $B$ is an explorer, she prefers to see a large number of different slopes (not necessarily black ones) but accepts to wait longer if that is required.

- Rookie $C$ just started skiing and therefore wants to avoid black slopes and appreciates a tour consisting of easy slopes. He prefers a good meal to extraordinary adventures.

Figure 5.4 depicts corresponding constraint preference graphs extracted from this description and Figure 5.5 shows how the sample assignments are evaluated using the weight representation of the example preferences. Every preference graph results in a different winner assignment and we need to judge whether those decisions fit the preferences. $A$ indicated little interest in variety, avoiding black slopes or easy tracks and got the only assignment that does not require waiting. Similarly, we get a match for $B$'s requirements and do not force $C$ to take difficult slopes. The calculated assignment winners thus fit the preference specifications and show how the different graphs influence the decision process. Interestingly for $B$, the selected dominance property affects the preferred solution. Since $\theta_2$ only satisfies VT and violates the five other constraints, it is considered worse than $\theta_3$ in SPD and DPD semantics. However, as VT is the most important constraint for $B$ and is only satisfied by $\theta_2$, in a TPD semantics this solution is still preferred over the others satisfying more constraints.




|  | Rookie $C$ new |
|---|---|
| $\theta_1$ | **7** |
| $\theta_2$ | 19 |
| $\theta_3$ | 8 |

|  | Rookie $C$ new |
|---|---|
| $\theta_1$ | **37** |
| $\theta_2$ | 127 |
| $\theta_3$ | 75 |

(a) Graph for rookie $C$ new (b) SPD semantics (c) TPD semantics

Figure 5.6: After adaption, rookie $C$ now prefers $\theta_1$ instead of his initial favorite assignment $\theta_3$ in Figure 5.5.

### 5.2.2 Changing Preferences

In multi-agent systems, agents change their goals depending on their perceived environmental situation [Doyle and McGeachie, 2003]. Similarly, our personas may change their constraint relationships given new circumstances.

Assume, for instance, rookie $C$ has gotten enough practice such that avoiding black slopes is not as important as before — but he is not keen on waiting long anymore. Hence, the edge $\mathsf{ABS} \succ_R \mathsf{LW}$ gets inverted such that $\mathsf{LW} \succ_R \mathsf{ABS}$, making LW the most important constraint. Assignment $\theta_1$ is the only one that has a route without much waiting — we expect this as the new favored outcome.

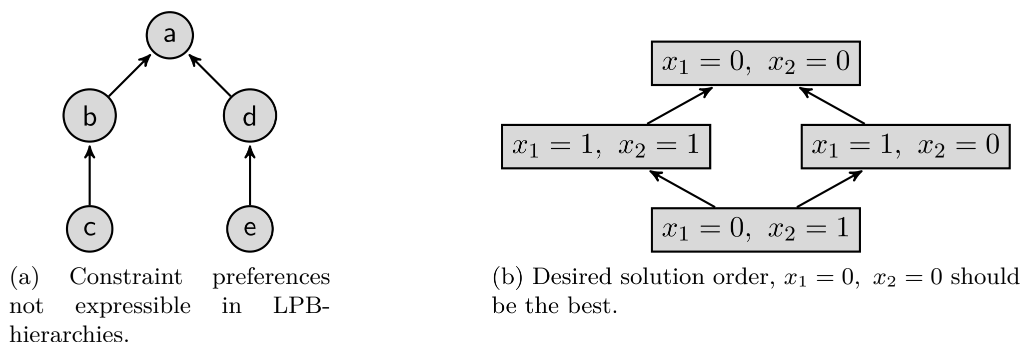
Figure 5.6 shows that this is indeed the case. One inverted edge led to a revisited weighting that influences the solution preference.

### 5.2.3 Changing Constraints

A similar, frequently arising use case is that of a change in available preferences and constraints. For instance, stakeholders may impose new desirable properties or others become obsolete. In our example, assume that slopes and lift endpoints have been evaluated with regards to beautiful landscapes (BL is true if one of those locations are part of the itinerary). Additionally, currently foggy slopes (FS) should be avoided.

Assuming $A$ does not care too much for either landscapes or foggy slopes, it is safe to assume that those constraints would be ranked even less important than LI. Boarder $B$, however, does care about BL about as much as variety and marks them more important than FS, FP, and ABS. Assume further that $\theta_1 \models \{\mathsf{BL}, \mathsf{FS}\}$, $\theta_2 \models \{\mathsf{FS}\}$ and $\theta_3 \models \{\mathsf{FS}\}$.

It is easy to calculate that $A$ still ranks $\theta_1$ before $\theta_3$ and that one before $\theta_2$ even though different numeric values are placed. To keep this order while including the new constraints, *six* weights would need to be changed while adding *two* new preference edges is enough. With larger preference models, these savings in maintenance effort could be more substantial. For $B$ the situation is different, as BL is only satisfied by $\theta_1$ which is why it would then be the preferred solution in all dominance properties.




(a) Constraint preferences not expressible in LPB-hierarchies.

(b) Desired solution order, $x_1 = 0,\ x_2 = 0$ should be the best.

Figure 5.7: Constraint preferences properly generalize a subclass of constraint hierarchies and target other application areas than CP-nets.

## 5.3 Constraint Preferences and Related Formalisms

Constraint preferences are an inherently simplistic formalism that only elicits the actual preferences and the dominance property from the user. Lifting these preferences to violation sets requires some consideration but can be carried out behind the scenes. Still, there are other preference formalisms that offer *qualitative*, i.e., relational specifications such as constraint hierarchies or conditional preference nets (CP-nets). We discuss their relationship to constraint preferences.

### 5.3.1 Reducing Constraint Hierarchies to Constraint Preferences

Constraint hierarchies [Borning et al., 1992] offer different *comparators* that aggregate the soft constraint valuations of a layer and order them. There, each soft constraint is given by an error function (most elementary, just a mapping from *false* to 1 and *true* to 0) and additional weights. Comparators are divided into *locally better* and *globally better*. Locally-better is based only on the error functions, whereas globally-better offers different aggregation functions such as summing valuations, the squared errors, or taking the maximal error. It also takes constraint-specific weights into account. Both comparator types are applied lexicographically over multiple layers in the sense that an ordering is found at the first level of the hierarchy where one solution performs better than the other. Since each layer is ordered totally, the resulting lexicographic order for the whole hierarchy is also total (Chapter 7 devotes more space to the formal definition of constraint hierarchies).

Error functions are either *predicate* functions converted from a conventional constraint $c$, i.e., $e(c, \theta) = 0$ if $\theta \models c$, and 1 otherwise; or *metric* functions that give a continuous degree of violation (e.g., for $c \triangleq (x = y)$, $e(c, \theta)$ could simply return $|\theta(x) - \theta(y)|$). Since constraint preferences focus on boolean soft constraints that are ranked by importance rather than combining metric error functions, we restrict ourselves to comparison with *locally predicate better* (LPB) and show that these hierarchies can be encoded in constraint preferences. We then demonstrate that constraint preferences properly generalize LPB-hierarchies by providing an example that cannot be expressed in LPB-hierarchies.

First, consider the definition of LPB, given a constraint hierarchy $H = \{H_0, \ldots, H_n\}$ where each $H_i$ is a set of boolean (soft) constraints. The operator $<_{LPB}$ compares two assignments $\theta$ and $\theta'$ (the constraints in $H_0$ are taken to be the hard constraints) and $\theta' <_{LPB} \theta$ should be

read as "$\theta'$ is worse than $\theta$"; Borning et al. [1992] define it as:

$$\theta' <_{LPB} \theta \leftrightarrow \exists k > 0 : (\forall i \in 1..k-1 : \forall \mu \in H_i : e(\mu,\theta) = e(\mu,\theta')) \wedge (\forall \mu \in H_k : e(\mu,\theta) \leq e(\mu,\theta')) \wedge (\exists \mu \in H_k : e(\mu,\theta) < e(\mu,\theta'))$$

Note that on layers lower than $k$, the satisfaction/violation values must be *exactly* the same for every soft constraint in locally-better (i.e., it does not suffice to satisfy the same number of constraints on every layer). Our encoding of the constraint hierarchy $H$ in constraint preferences $\rightarrow_P$ is then defined as follows: $C = H_0$, $C_s = \bigcup_{i \in 1..n} H_i$, and

$$c' \rightarrow_P c \Leftrightarrow c \in H_i \wedge c' \in H_{i+1} \ .$$

Hence, we construct a graph that is fully connected between the constraints of adjacent layers but has no additional edges. We write $T_i$ for all constraints in hierarchy level $i$ that are violated by an assignment $\theta$, i.e., $T_i = V(\theta) \cap H_i = \{c \in H_i \mid \theta \not\models c\}$ and, analogously, $U_i = V(\theta') \cap H_i$; we abbreviate $\bigcup_{k \leq i \leq l} T_i$ by $T_{k..l}$.

**Theorem 5.2.** *If $\theta' <_{LPB} \theta$, then $V(\theta') \prec^P_{\mathrm{TPD}} V(\theta)$.*

*Proof.* Observe that for locally-predicate-better $e(c,\theta) < e(c,\theta')$ iff $\theta \models c \wedge \theta' \not\models c$; and $e(c,\theta) \leq e(c,\theta')$ iff $\theta' \models c \Rightarrow \theta \models c$.

Assume $\theta' <_{LPB} \theta$; we have to show that $T = V(\theta)$ is worse than $U = V(\theta')$ by application of TPD-rules. Let $k > 0$ be such that (*) $\forall i \in 1..k-1 : \forall c_i \in H_i : \theta \models c_i \leftrightarrow \theta' \models c_i$, (**) $\forall c_k \in H_k : \theta' \models c_k \rightarrow \theta \models c_k$, and pick $c \in H_k$ such that $\theta \models c \wedge \theta' \not\models c$, hence $c \in U_i$ but $c \notin T_i$. Those all need to exist due to our assumption of $\theta' <_{LPB} \theta$.

By (*) we have that $T_{1..k-1} = U_{1..k-1}$. Furthermore, $T_{1..k} \subseteq U_{1..k} \setminus \{c\}$ since $\forall c_k \in H_k : \theta \not\models c_k \Rightarrow \theta' \not\models c_k$ by (**), and $c \notin T_{1..k}$. In particular, $T_{1..k} \subseteq U_{1..k} \setminus \{c\} \subseteq U_{1..n} \setminus \{c\}$. If $T_{1..k} = U_{1..n} \setminus \{c\}$, then $U = T_{1..k} \uplus \{c\} \prec^P_{\mathrm{TPD}} T_{1..k} \uplus T_{k+1..n} = T$ by (TPD), since all constraints in $T_{k+1..n}$ are transitively dominated by $c \in H_k$. If on the other hand $T_{1..k} \subsetneq U_{1..n} \setminus \{c\}$, then $U_{1..n} \setminus \{c\} \prec^P_{\mathrm{TPD}} T_{1..k}$ by (XPD) and $U = (U_{1..n} \setminus \{c\}) \uplus \{c\} \prec^P_{\mathrm{TPD}} T_{1..k} \uplus \{c\}$ since $c$ does not appear in either side. Combining that with rule (TPD) and transitivity of $\prec^P_{\mathrm{TPD}}$, we get $U \prec^P_{\mathrm{TPD}} T_{1..k} \uplus \{c\} \prec^P_{\mathrm{TPD}} T_{1..k} \uplus T_{k+1..n} = T$ since $T_{k+1..n}$ are all predecessors of $c$. □

Conversely, Figure 5.7a shows a simple constraint preferences example that is not expressible with LPB hierarchies – as we can confirm by quick reasoning: Let $H : \{\mathsf{a},\mathsf{b},\mathsf{c},\mathsf{d},\mathsf{e}\} \rightarrow \mathbb{N} \setminus \{0\}$ be a mapping from the constraints to their respective hierarchy levels. We consider solutions that only satisfy *one* constraint and violate all others and write a for "a solution satisfying only a". We show that every admissible choice of $H$ introduces too much ordering: The constraint preferences require a to be better than b which in turn should be better than c, thus we have to have $H(\mathsf{a}) < H(\mathsf{b}) < H(\mathsf{c})$. Since we expect a to be more important than d as well, but require b and d to be incomparable, $H(\mathsf{d})$ has to be equal to $H(\mathsf{b})$. Similarly, $H(\mathsf{e})$ has to be $H(\mathsf{c})$ as e and c should be incomparable. But then b would be better than e, a preference relation that is explicitly not modeled in the underlying constraint preferences.

Besides this encoding of a class of constraint hierarchies in constraint preferences, the attentive reader will notice that MiniBrass already offers lexicographic products of PVS to express lexicographic orders (see Section 4.4.1). Indeed, by having every hierarchy layer mapping to a PVS (perhaps cost functions) and combining them lexicographically, we can capture every proposed constraint hierarchy type and many more. Since this construction does not involve constraint preferences but operates on more abstract PVS types, we present its discussion in Section 7.1.

### 5.3.2 Relationship with CP-Nets

Similar to ordering constraints in constraint preferences, CP-nets [Boutilier et al., 1997] specify total orders over the *domain* of a variable depending on an assignment to other variables in a so-called conditional preference table. A CP-net preference statement for a variable $y$ is written as:

$$x_1 = d_1, \ldots, x_n = d_n : y = w_1 \succ \cdots \succ y = w_k$$

where $x_1, \ldots, x_n$ are the *parent* variables of $y$ and $w_1, \ldots, w_k$ are all domain values of $y$ given in a total order $\succ$. Such an order needs to be specified for all assignments to $x_1, \ldots, x_n$. The assignment to the parent variables frame the “context” in which another variable’s domain should be ordered. A preference statement should be interpreted as “Given that $x_1 = d_1, \ldots, x_n = d_n$, all other variables being equally assigned, prefer a solution that assigns $w_i$ to $y$ over one that assigns $w_j$ to $y$ if $i > j$” which is the *ceteris paribus* assumption. The change of value for $y$ from $w_i$ to $w_j$ is then called a “worsening flip”. A complete assignment $\theta$ to the variables of a CP-net is preferred to another one, say $\theta'$ , if $\theta'$ can be obtained from $\theta$ via a sequence of worsening flips [Meseguer et al., 2006].

On the one hand, the induced “better-as” relation on assignments needs not be a partial order since cycles may arise [Boutilier et al., 2004]. By contrast, constraint preferences always lead to a partial order $\preceq^P_{\text{xPD}}$ on violation sets of assignments. On the other hand, CP-nets cannot express all partial orders on assignments [Rossi et al., 2008b]. Consider the minimal example depicted in Fig. 5.7b: $X = \{x_1, x_2\}$, $D_1 = D_2 = \{0, 1\}$. The proposed solution order cannot be expressed in CP-nets since $\{x_1 \mapsto 1,\ x_2 \mapsto 0\}$ and $\{x_1 \mapsto 1,\ x_2 \mapsto 1\}$ differ only by the assignment of $x_2$ and have to be comparable because of the total order requirement and *ceteris paribus* semantics in CP-nets. But this solution ordering is easily expressible in constraint preferences defining a constraint for each possible assignment.

Thus, the two frameworks are incomparable regarding the solution order. An extension, however, of the constraint preferences formalism with conditional statements as in CP-nets might turn out to be fruitful. Apart from the formal viewpoint, CP-nets might be better suited for smaller, fine-grained decision situations such as configuration tasks where users are able to lay out their preferences on domain items. Constraint preferences, on the other hand, work best if a large class of solutions can be characterized by their violation or satisfaction of a certain soft constraint and these soft constraints can be ranked – especially if the decisions are combinatorial.

## 5.4 Solving “Constraint Preferences” Problems

Solving any soft constraint satisfaction problem generally results in a set of several optimal solutions due to the partiality of the induced order (see Section 3.2.3). This is clearly the case with constraint preferences – regardless of whether SPD or TPD lift the preferences. We can apply the well-known branch-and-bound algorithm (see, e.g., Section 3.3) to soft-constraint problems specified using constraint preferences. This algorithm is similar to the MiniBrass-based branch-and-bound implementation shown in Section 4.4.2 but does not rely on an underlying constraint solver or modeling language such as MiniZinc and serves as a basic implementation of problems specified with constraint preferences. In isolation, we can investigate the influence of variable ordering heuristics such as “assign the variables involved in the most important constraints first”.

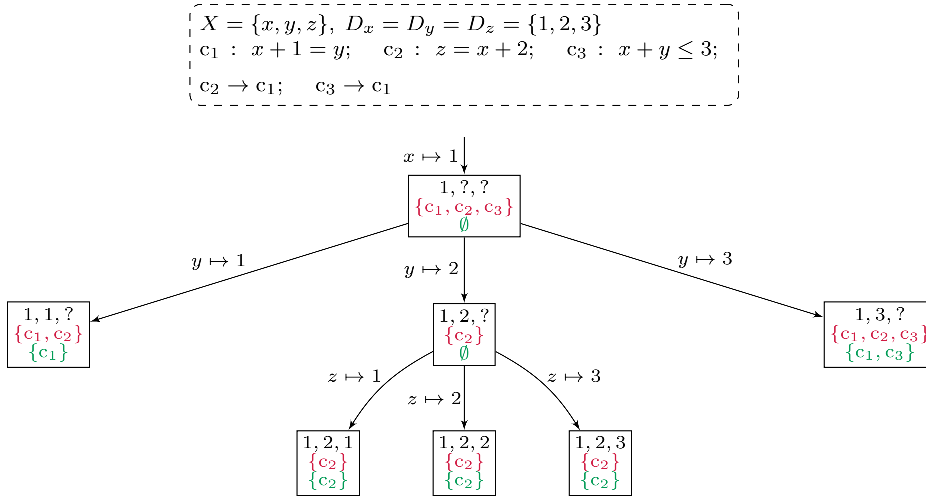



Figure 5.8: An example constraint preferences problem with a search space as traversed by branch-and-bound. In every node, the first line indicates the partial assignment so far, the middle line refers to the worst-case estimation (i.e., the upper bound on the violation set), and the third line contains the best-case estimation (i.e., the lower bound on the violation set).

In essence, the instantiation of branch-and-bound proceeds analogously to Figure 3.3: We select variables in some order and try to assign each domain value in a systematic way. At every node in the search degree, we already know the satisfaction/violation of soft constraints whose scope is fully assigned. Therefore a lower bound on the violation sets is given by the already assigned and violated soft constraints. All "open" soft constraints, i.e., those that are not yet fully assigned and have some more variables in their scope downstream the search tree, give rise to (obvious) lower and upper bounds on the violation sets:

- **Lower bound** (best case) *no* additional soft constraints will be violated
- **Upper bound** (worst case) *all* additional soft constraints will be violated

Figure 5.8 visualizes branch-and-bound for constraint preferences on a toy problem.

It is well known that the success of systematic tree search methods is greatly influenced by the variable and value ordering and several heuristics have been devised for classical constraint problems [Gent et al., 1996]. Successful heuristics typically take into account the domain cardinality or the effect assignments make [Levasseur et al., 2007]. We propose heuristics for the investigated soft-constraint problems over constraint preferences:

1. *Most important first* (MIF) variable ordering: A topological ordering of a constraint relationship lists most important constraints first and assigns values to their variables first.

2. *Local consistency* (LC): We improve lower and upper bounds inspired by soft-arc consistency [Cooper et al., 2010]. For each partially-assigned (binary) constraint, we include constraints that are violated irrespective of the value of the remaining variable into the best-case estimation and exclude those that are certainly true from the worst-case estimation.

Table 5.1: Modeling influence: Comparison of mean runtimes in seconds and recursive calls for SPD vs. TPD and weighted CSP vs. constraint preferences with standard deviations given in parentheses.

| | WCSP-RT | WCSP-RC | CPCSP-RT | CPCSP-RC |
|---|---|---|---|---|
| SPD | 0.86 (2.79) | 7183.78 (22617.88) | 2.08 (5.77) | 13323.36 (32050.35) |
| TPD | 0.83 (2.81) | 6888.38 (22622.73) | 2.00 (5.73) | 12721.54 (31996.98) |

## 5.5 Evaluation

In a first test-bed for constraint preferences without MiniZinc/MiniBrass, we implemented a branch-and-bound algorithm in Java. For replicability of our experiments, the source code is available online.[2] In particular, we wanted to compare our heuristics to a naïve branch-and-bound implementation that is intractable for a larger number of variables.

We test several parameters regarding algorithmics (variable orderings and local consistency) and modeling (using SPD or TPD semantics) on randomly generated CSP instances and measure running time as well as the number of recursive branch-and-bound calls. Constraints are simple arithmetical expressions including $+$, $-$, $*$, $/$, and $<$, $\leq$, $=$, $\geq$, $>$ as relational operators. In each experiment, $n$ random problems were solved by differently parametrized solvers.

### 5.5.1 Modeling Influence

First, we investigate how the selected dominance property and choice of partial valuation structure (using constraint preferences directly or weights according to Section 5.1.3) affect the solver's efficiency in order to select the formalism for which to evaluate the proposed heuristics in more detail. We used the bounds as previously described in Section 5.4, the variable ordering to use MIF (most important first), and the usage of a local consistency check (LC). We compare weighted CSPs (WCSP) and constraint preference CSPs (CPCSP) with regard to runtime (RT) and recursive calls (RC). The concrete weights for the constraints ranked by a set of constraint preferences are computed recursively, as described in Section 5.1.3:

$$\begin{aligned} w^P_{\mathrm{SPD}}(\mu) &= 1 + \max\{w^P_{\mathrm{SPD}}(\mu') \mid \mu' \in C_\mathrm{s} : \mu' \to_P \mu\} \ , \\ w^P_{\mathrm{TPD}}(\mu) &= 1 + \textstyle\sum_{\mu' \to_P \mu} 2 \cdot w^P_{\mathrm{TPD}}(\mu') - 1 \ . \end{aligned}$$

This experiment consists of 50 runs with 7 variables having domains of 5 to 15 values and 10 constraints arranged in a random set of constraint preferences.

As expected, Table 5.1 shows that weighted CSPs require less runtime and recursive calls to find the maximum solution degrees. This is mostly due to the presence of incomparable elements in constraint relationships which reduce the possible pruning and to the more complicated evaluation of the upper (Smyth) ordering. However, the stronger dominance property TPD only improved the pruning by about 5% in the constraint preferences case and 4% in the weighted CSP case. This is still a considerable gain but does not completely rule out SPD

[2] `https://github.com/isse-augsburg/constraint-preferences-csemiring`

Table 5.2: Algorithmic efficiency: Comparison of mean number of recursive calls ($\cdot 10^3$) for different combinations of using bounding pairs, local consistency and the most-important-first variable heuristic for a fixed weighted CSP with standard deviations given in parentheses.

| | ¬ WCB + ¬ LC | ¬ WCB + LC | WCB + ¬ LC | WCB + LC |
|---|---|---|---|---|
| RO | 1,914 (1,359) | 472 (993) | 1,914 (1,359) | 472 (993) |
| MIF | 1,264 (1,290) | 218 (649) | 1,264 (1,289) | 217 (649) |

in terms of performance. Consequently, we focus on weighted CSP instances to examine the proposed solver heuristics.

### 5.5.2 Algorithmic Efficiency

With a weighted CSP based on constraint relationships and TPD semantics, we investigate how variable ordering using the most-important-first heuristic, bounding pairs, as well as local consistency affect the performance. The CSPs for this experiment had 6 variables, 10 constraints and domains with 10 to 30 values, and 150 runs were made. We compare values for random order (RO) versus most-important-first (MIF) with flags for worst-case boundaries (WCB) and local consistency (LC).

For clarity, we discuss the number of recursive calls to measure the pruning performance of the algorithm. Table 5.2 clearly shows that using the most-important-first heuristic significantly improves the performance. Similar improvements can be achieved using local consistency checks for improved boundaries. Using a worst-case boundary only does not lead to better results in our experiments. These results hint that variable ordering heuristics using "important" variables should be researched further in conjunction with traditional ordering heuristics based on, e.g., the cardinality of the domains.

While these experiments confirm our intuition about meaningful heuristics and demonstrate that constraint preferences including the Smyth-ordering can be implemented in a solver, we replicate this experimental question in our overall evaluation of MiniBrass in Section 9.3.

**Chapter Summary and Outlook**
In this chapter, we laid out the motivations for specifying importance over soft constraints (first) in a qualitative way. Users only have to specify a directed acyclic graph over constraints and can convert the specification to weighted CSP, if needed. We went on to provide two kinds of dominance properties that lift a given set of constraint preferences to violation sets as reported by solutions. We demonstrated a simplified use case to show the benefits of qualitative soft constraint specifications. Still, we have not provided a PVS representing constraint preferences, yet. The embedding of constraint preferences into the realm of algebraic structures is subject of Chapter 6.

Chapter 6

# Designing Algebraic Structures for Soft Constraints

**Summary.** This chapter raises the question "what is the minimal amount of structure that has to be added to any given partial order to properly model soft constraint problems?". This question is relevant as any required axioms for a soft constraint framework possibly exclude specific formalisms. We provide a free construction (in categorical/algebraic terms) of a partial valuation structure from a partial order and argue that this is the natural choice as the underlying formalism for MiniBrass. Many formalisms are indeed already instances of PVS – without any surrounding construction. Moreover, we offer another free construction, that of a c-semiring from a partial valuation structure which creates "artificial suprema", if needed for a particular algorithm or solving technology. Throughout the chapter, questions of adequacy are discussed.

**Publication.** The concepts and results outlined in this chapter have been published in [Schiendorfer et al., 2015c, 2018; Knapp et al., 2014].

Many soft constraint formalisms are subsumed under algebraic structures that only prescribe which axioms the involved orderings and combination operations have to satisfy. Based on those axioms, generic solvers can be developed. However, imposing a certain structure requires justification – as this decision might exclude important concrete formalisms from a generic framework. If that is the case, there better be good reasons for doing so.

The previous chapter highlighted constraint preferences as a qualitative alternative to the otherwise quantitative formalisms frequently used in soft constraints. MiniBrass furthermore applies the concept of partial valuation structures as the least common denominator for a broad variety of soft constraint formalisms. While it is convenient that many proposed formalisms indeed match the specifications of valuation structures or c-semirings, from a theoretical perspective, it is interesting to ask for the *least* amount of structure that has to be added to a partial order to model soft constraint problems. Once the minimal requirements are known, we can still propose additional axioms (such as totality for valuation structures or the existence of suprema for c-semirings) for modeling purposes. Although these sections are rather formal, the presented constructions and orders are implemented in the MiniBrass library (see Chapter 4).

## 6.1 Looking for *Free* Partial Valuation Structures

Recalling Example 3.2, we had a soft constraint problem with soft constraints $c_1$, $c_2$, and $c_3$ such that $c_1$ was more important than $c_2$ or $c_3$ whereas the latter two were incomparable, from a user's perspective (forming the partial order $\mathsf{U}$). Chapter 5 provides constraint preferences for such situations but left open what the appropriate PVS would be. Without constraint preferences, the choice of $M = (\mathbb{N}, +, 0, \geq)$ as the underlying PVS seems rather obvious, given that the weighting $\vec{w} = [2, 1, 1]$ is consistent with the intuition that $c_1$ should be weighted higher than $c_2$ and $c_3$. However, interpreting $\vec{w}$ as a function $w : C_{\mathrm{s}} \to \mathbb{N}$, we see that $w$ clearly is only a monotone (not isomorphic) function. It totalizes $\mathsf{U}$ by making the incomparable elements $c_2$ and $c_3$ *equal.* Alternatively, $\vec{w} = [3, 1, 1]$ would be consistent, as would be $\vec{w} = [3, 2, 1]$, although our intuition tells us that making $c_2$ *more important* than $c_3$ certainly is a bad idea. We could try another PVS, say $M' = (2^{C_{\mathrm{s}}}, \cup, \emptyset, \supseteq)$ to denote solution degrees as sets of violated soft constraints that are combined by union. Then, however, $c_1$, $c_2$, and $c_3$ would each contribute equally to a solution's quality, contrary to our model marking $c_2$ and $c_3$ as less urgent than $c_1$. Certainly, we want any mapping $\varphi$ into a PVS $N$ to preserve the given order: $\varphi(p) \leq_N \varphi(q)$ whenever $p \leq_{\mathsf{U}} q$; otherwise we invert ordering decisions.

Our point is, there is an infinite number of PVS that represent $\mathsf{U}$ to a certain degree – but the essential question is what are the *minimum requirements* in terms of comparability *any* PVS have to fulfill? Which PVS is, in this sense, the *best*, i.e., most general, one?

To find an answer, we consider the more general question of how to "convert" any partial order $P$ into a partial valuation structure. As first presented in [Knapp et al., 2014] and proved in Section 6.3, we can indeed lift any $P$ to $PVS\langle P\rangle$ – i.e., construct a suitable combination operation and neutral element: We take as elements $\mathcal{M}_{\mathrm{fin}}(P)$, the set of finite multisets composed from elements in $P$. For instance, $\wr\int$, $\wr c_1, c_2, c_2\int, \ldots \in \mathcal{M}_{\mathrm{fin}}(\mathsf{U})$. Two multisets are combined using multiset-union with $\wr\int$ being the neutral element. Finally, a compatible ordering (with $\wr\int$ being top) is found inductively by applying the Smyth-ordering on sets (see Figure 3.2) to multisets (then written as $\preceq_P$) [1]:

**Definition 6.1 – Smyth-ordering over Multisets**
The Smyth-ordering on $\mathcal{M}_{\mathrm{fin}}(P)$ is the binary relation $\preceq_P \subseteq \mathcal{M}_{\mathrm{fin}}(P) \times \mathcal{M}_{\mathrm{fin}}(P)$, given by the reflexive-transitive closure of

$$p <_P q \Rightarrow T \uplus \wr p\int \prec_P T \uplus \wr q\int$$
$$T \supset U \Rightarrow T \prec_P U$$

Intuitively, when we compare two multisets according to $\preceq_P$, we have to match every element $q$ on the right side with a dominated element $p = h(q)$ on the left side such that $p \leq_P q$ and $h$ is injective (see Lemma 6.1 and Figure 6.4). There may be additional elements on the left. For any elements $p, p'$ in a partial order $P$, we have $p \leq_P p' \Leftrightarrow \wr p\int \preceq_P \wr p'\int$. Note the monotonicity of the Smyth-ordering with respect to multiset union; if $T \preceq_P U$, then $T \uplus V \preceq_P U \uplus V$, since this holds for both defining clauses of the ordering. Antisymmetry is

[1] This relation is, in its set version, used to express non-determinism of programs in denotational semantics (set-valued to "collect different program results"), i.e., so-called power domains [Amadio and Curien, 1998, Ch. 9].

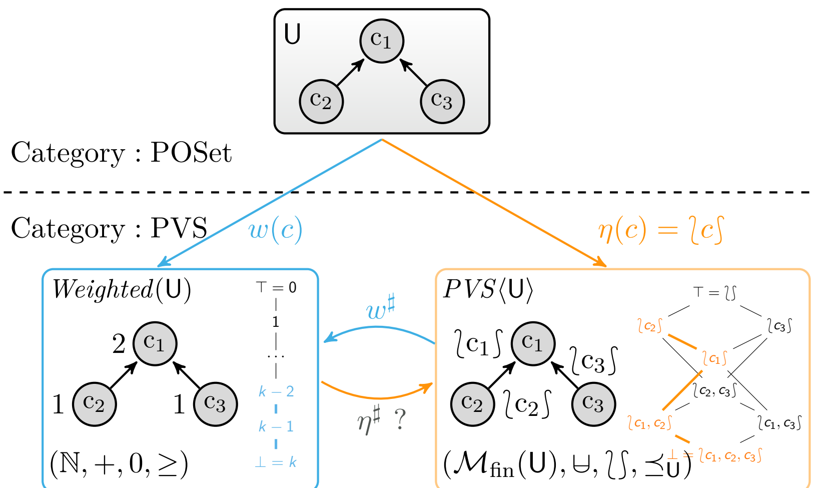



Figure 6.1: Encoding preferences given by the partial order U as two different PVS: *Weighted*(U) and *PVS*⟨U⟩. Highlighted paths show possible improvement steps during optimization. There can be no mapping $\eta^\sharp$ since distinct elements $c_2$ and $c_3$ are unified to 1 in *Weighted*(U) and would need to be represented by $\lbag c_2 \rbag$ and $\lbag c_3 \rbag$ in *PVS*⟨U⟩, respectively.

proved in Section 6.3. As an example, we have $\lbag c_1, c_1, c_2 \rbag \prec_{\mathsf{U}} \lbag c_1, c_2 \rbag$ or $\lbag c_1, c_3 \rbag \prec_{\mathsf{U}} \lbag c_2, c_3 \rbag$, if we read $c_2 \to c_1$ as $c_1 <_{\mathsf{U}} c_2$. In conclusion, $PVS\langle P \rangle = (\mathcal{M}_{\text{fin}}(P), \uplus, \lbag\rbag, \preceq_P)$.

Since $PVS\langle P \rangle$ can be the codomain of any *SCSP*, soft constraints $\mu_i$ can arbitrarily map to $\mathcal{M}_{\text{fin}}(P)$, e.g., $\mu_i(\theta) = \lbag c_1, c_1, c_2 \rbag$. We derive a particularly interesting instance instead (constraint preferences), if we convert a boolean soft constraint $c_i$ into $\mu_i(\theta)$ which maps to $\lbag c_i \rbag$ if $\theta$ satisfies $c_i$ and $\lbag\rbag$ otherwise. In the context of constraint preferences, the Smyth-ordering is called *single-predecessor-dominance* in Section 5.1.1 since – everything else being equal – a single predecessor can be dominated by a more important constraint due to the first clause of the ordering.

Figure 6.1 displays how we can encode a partial order $P$ as either a weighted PVS or using $PVS\langle P \rangle$ by, e.g., representing $c_1$ as $w(c_1) = 1$ or as $\eta(c_1) = \lbag c_1 \rbag$, respectively. Notice that a weighting $w : P \to \mathbb{N}$ can be "emulated" from $\mathcal{M}_{\text{fin}}(P)$ by defining its "lifted" version $w^\sharp : \mathcal{M}_{\text{fin}}(P) \to \mathbb{N}$ on the level of multisets: $w^\sharp(\lbag p_1, \ldots, p_n \rbag) = \sum_{i=1}^{n} w(p_i)$ (arrow from right to left in the diagram). The converse, however, does not work: Once, e.g., $c_2$ and $c_3$ are both mapped to 1, we cannot "extract" information back about the origin to design a mapping $\eta^\sharp$ that can map from $\mathbb{N}$ into $\mathcal{M}_{\text{fin}}(\mathsf{U})$ since $\eta^\sharp(1)$ would have to simultaneously be equal to $\lbag c_2 \rbag$ and $\lbag c_3 \rbag$. This tells us (not surprisingly) that $c_2$ and $c_3$ *do not necessarily have to be treated as equal elements*, i.e., there exist other, more general, partial valuation structures encoding U that keep them as distinct elements.

It was not a coincidence that we found a lifted mapping $w^\sharp$ from $PVS\langle P \rangle$ to *Weighted*$(P)$ that is equal to applying $w$ directly from $P$. Instead, $PVS\langle P \rangle$ has the *universal mapping property* [Knapp et al., 2014]: *Any* order-preserving function $\varphi$ from $P$ into the underlying partial order of a PVS can be decomposed into the form $\varphi^\sharp \circ \eta$ in a *unique* way. Thus, $PVS\langle P \rangle$ is also called the "free PVS over $P$". Practically, this means that we can always safely convert $P$ into the free PVS before mapping to another PVS (e.g., if we need an integer objective for our implementation, see Section 4.4) without losing any information. Conversely, *Weighted*$(P)$

is not free as we cannot return to $PVS\langle P\rangle$ once $P$ is mapped to $\mathit{Weighted}(P)$. Since free objects are unique up to isomorphism [Sannella and Tarlecki, 2012, p. 147], $PVS\langle P\rangle$ can be seen as *the most general* PVS over a partial order. We prove this fact in Lemma 6.2 in Section 6.3.

Our original question, "how to formulate an ordering over constraints as a PVS with the least overhead", thus boils down to the search for a *free construction*. Similar instances are the free monoid or the free group over a set. We can capture this task formally using the language of *category theory* (hinted in Figure 6.1) which studies, inter alia, algebraic structures along with their structure-preserving mappings. This perspective further enables us to treat the transformation from a partial order into a PVS and that from a PVS into a c-semiring *uniformly*. The subsequent sections hence draw on basic knowledge of category theory when they offer the derivation of the free PVS and the free c-semiring, respectively. In Section 6.2, we introduce categorical concepts relevant to free constructions with the well-known free monoid over a set. Readers familiar with basic category theory may safely skip Section 6.2 and readers familiar with term algebras may check the categorical presentation. As category theory has not been used extensively in constraint programming (except for [Diaconescu, 1994]), the interested reader is referred to excellent introductory material, e.g., [Pierce, 1991; Barr and Wells, 1990; Awodey, 2010].

Mathematical categories (written as $\mathcal{C}$) are composed of *objects* (e.g,. algebraic structures) and *morphisms* (e.g., structure-preserving mappings) between them. Morphisms generalize set-valued functions. Each $\mathcal{C}$-morphism $f$ admits a domain $A$ and codomain $B$, both being $\mathcal{C}$-objects, and is written as $f : A \to B$. For all morphisms $f : A \to B$ and $g : B \to C$ there has to be a composite arrow $(g \circ f) : A \to C$. Morphism composition $\circ$ needs to be associative and for each object $A$, there has to be an identity morphism $\mathrm{id}_A : A \to A$ acting as "neutral element" with respect to composition, i.e., $\mathrm{id}_B \circ f = f \circ \mathrm{id}_A = f$. For instance, the category Set has conventional sets as objects and functions as morphisms, whereas the category Mon has monoids as objects and monoid-homomorphisms as morphisms. Another example is given by PO, the category of partially-ordered sets, that has partial orders as objects and partial order homomorphisms (i.e., monotone functions) as morphisms. Note that this definition is proper since monotone functions are closed under function composition, i.e., if $\varphi : |P| \to |Q|$ and $\psi : |Q| \to |R|$ are monotone functions, so is $\psi \circ \varphi$. All categories relevant to the discussion of partial orders, partial valuation structures, and c-semirings are examples of so-called *concrete categories* with objects being sets with additional algebraic or ordered structure and morphisms being set-theoretic (structure-preserving) functions – in general, this need not be the case.

A *functor* $F$ between categories $\mathcal{C}$ and $\mathcal{D}$ is a mapping that sends every $\mathcal{C}$-object $A$ to a $\mathcal{D}$-object $F(A)$ and every $\mathcal{C}$-morphism $\varphi : A \to B$ to a $\mathcal{D}$-morphism $F(\varphi) : F(A) \to F(B)$. For example, for every set $A$ there is an associated monoid $(A^*, ::, \varepsilon)$ with words over $A$ and concatenation. We can use this to define a functor $F : \mathrm{Set} \to \mathrm{Mon}$ by $F(A) = (A^*, ::, \varepsilon)$ and $F(f : A \to B) : A^* \to B^*$ with $F(f)([a_1, \ldots, a_n]) = [f(a_1), \ldots, f(a_n)]$, such that, in particular, $F(f)(w_1 :: w_2) = F(f)(w_1) :: F(f)(w_2)$. Conversely, there is the *underlying* functor $|{-}| : \mathrm{Mon} \to \mathrm{Set}$ with $|(A, \cdot, \varepsilon)| = A$ and $|\varphi : (A_1, \cdot_1, \varepsilon_1) \to (A_2, \cdot_2, \varepsilon_2)| = \varphi : A_1 \to A_2$ yielding the *underlying set* of a monoid.

This operator $|{-}|$ is a convention present in category-theoretical arguments. It allows to distinguish structures and sets and must not be confused with set cardinality. We follow this convention in the remainder of the dissertation and, e.g., will write a partial order as $P = (|P|, \leq_P)$.

Using the above notions, we can now formally state what a free object is:

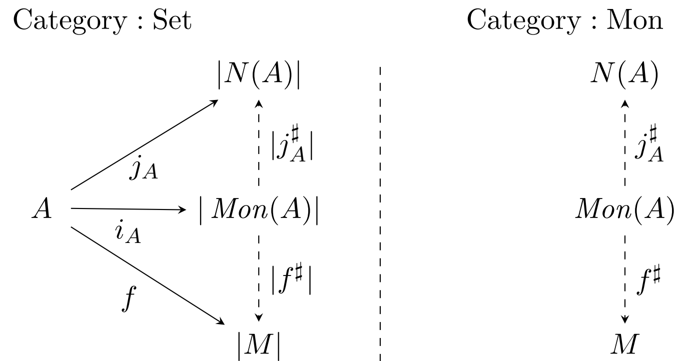



Figure 6.2: A diagram of the free monoid over a set from Example 6.1. $Mon(A)$ and $N(A)$ refer to $(A^*, ::, \epsilon)$ and $(2^A, \cup, \emptyset)$, respectively and $M$ just refers to *any* monoid. The embeddings $j_A(a) = \{a\}$ and $i_A(a) = [a]$ are defined analogously for any set $A$. A dashed arrow indicates that, e.g., there is a *unique* monoid homomorphism $f^\sharp$ that makes the diagram commute, i.e., $f = |f^\sharp| \circ i_A$.

**Definition 6.2 – Free object [Sannella and Tarlecki, 2012, p. 144]**
Given two categories $\mathcal{A}$ and $\mathcal{B}$ and a functor $G : \mathcal{B} \to \mathcal{A}$, the *free object* $F(A)$ in $\mathcal{B}$ over an object $A$ of $\mathcal{A}$ is characterized by a *unit* morphism $\eta_A : A \to G(F(A))$ in $\mathcal{A}$ such that for every $\mathcal{A}$-morphism $\varphi : A \to G(B)$ with $B$ an object of $\mathcal{B}$, there is a **unique** *lifting* $\mathcal{B}$-morphism $\varphi^\sharp : F(A) \to B$ satisfying $G(\varphi^\sharp) \circ \eta_A = \varphi$.

A free object $F(A)$ is unique up to isomorphism and the composition of two free constructions yields another free construction [Sannella and Tarlecki, 2012, Ch. 3]. Incidentally, the monoid $(A^*, \cdot, \varepsilon)$ is the free monoid over a set $A$ (see Section 6.2). A free object does not have to exist. We need to prove a particular free construction (e.g., free monoid or free PVS) by choosing the appropriate categories, functors, and, of course, the free object itself.

## 6.2 Free Objects in Category Theory: The Free Monoid over a Set

For our purposes, the true strength of category theory unveils when we consider transformations between different algebraic structures, e.g. between partial orders and PVS or between PVS and c-semirings. All of these will involve functors. We have already seen an example, $|P|$ that returns the underlying set of a partial order on objects and the underlying function of a monotone function (here, just itself) on morphisms. Despite our interest in the free *objects*, the behavior of the involved *morphisms* helps us to properly characterize the free objects. To see a more interesting example of functors that will provide intuition for Sections 6.3 and 6.4, consider the task of constructing a plain monoid (a set and one associative binary operation $\cdot$ with the neutral element $\varepsilon$) composed of elements taken from a set $A$. Our presentation closely follows [Awodey, 2010, p. 20].

**Example 6.1 – A monoid over a set**
Let $A = \{a_1, a_2, \dots\}$ be any set, called generators. We want to build a monoid $Mon(A) = (X, \cdot, \varepsilon)$ composed of the elements in $A$. $A$ is an object in the category Set, $Mon(A)$ is an object in the category Mon. Assume that a function $i_A : A \to X$ maps every $a \in A$ to a different "new" element in our new underlying set $X$. For simplicity, we represent every $a \in A$ by itself. Next, we add a dedicated neutral element $\epsilon$ and define $\epsilon \cdot x = x \cdot \epsilon = x$ for every $x \in X$. Now, for every pair of generators $a$ and $b$, we add a fresh element (denote it as $a \cdot b$ which is distinct from any other element $a' \in A$) and do so recursively for products of products etc. We only have to make sure to *equate the elements that have to be equal* by associativity, e.g., $(a \cdot b) \cdot c = a \cdot (b \cdot c)$. This can be easily achieved if we represent every element "without parentheses", leading to $X$ being the set of words over $A$ (i.e., $A^*$ with $\epsilon$ denoting the empty word) and $\cdot$ being the concatenation (written as ::). Now a functor $Mon : \text{Set} \to \text{Mon}$ takes every set $A$ to $(A^*, ::, \epsilon)$ and every function (morphism in the category Set) $f : A \to B$ to a monoid homomorphism $Mon(f) : Mon(A) \to Mon(B)$ which is defined as follows: $Mon(f)(\epsilon) = \epsilon$, $Mon(f)([a_1, \dots, a_n]) = [f(a_1), \dots, f(a_n)]$. Elements of $A$ are represented in $A^*$ by $i_A(a) = [a]$. Note that for any singleton list $[a] \in A^*$ iff $a \in A$.

With respect to Example 6.1, for constructing a monoid from a set, we could have also tried another functor $N(A)$ that maps $A$ to $(2^A, \cup, \emptyset)$ and represent $a \in A$ as $j_A(a) = \{a\}$. Clearly, $N(A)$ also satisfies the monoid axioms of associativity and $\emptyset$ being neutral. However, it is *too specific*: it assumes commutativity since $j_A(a) \cdot j_A(b) = \{a\} \cup \{b\} = \{b\} \cup \{a\} = j_A(b) \cdot j_A(a)$. But we have already seen another monoid, $Mon(A)$, where $i_A(a) \cdot i_A(b) = [ab] \neq [ba] = i_A(b) \cdot i_A(a)$. Hence, commutativity is not required for a monoid. Mapping $A$ to $N(A)$ would consequently unify elements that need not be equal. Once that mapping is done, it should be impossible to map "back" to a more general structure where the unified elements are distinct. Put differently, there do exist functions $f$ from $A$ to a monoid $M'$ that we cannot factorize as $f^\sharp \circ j_A = f$ for some $f^\sharp$.

Indeed, this is the case. Assume for a particular set $\mathsf{A} = \{\mathsf{a}, \mathsf{b}\}$ that we have some function $f$ into $|\,Mon(\mathsf{A})|$, for instance $f(\mathsf{a}) = [\mathsf{aba}]$ and $f(\mathsf{b}) = [\mathsf{bab}]$. Now assume that we mapped $\mathsf{A}$ to $N(\mathsf{A})$ via $j_\mathsf{A}$, having $\mathsf{a}$ and $\mathsf{b}$ now represented as $\{\mathsf{a}\}$ and $\{\mathsf{b}\}$, respectively. Is there a way we can "still" reconstruct the function $f$, starting from $N(A)$ and calling it $f^\sharp$? To fulfill $f = f^\sharp \circ j_\mathsf{A}$, we know that $f^\sharp(\{\mathsf{a}\}) = [\mathsf{aba}]$ and $f^\sharp(\{\mathsf{b}\}) = [\mathsf{bab}]$ must hold. But what about $f^\sharp(\{\mathsf{a}, \mathsf{b}\})$? To satisfy monoid homomorphism laws, $f^\sharp(\{\mathsf{a}, \mathsf{b}\})$ must equal $f^\sharp(\{\mathsf{a}\}) :: f^\sharp(\{\mathsf{b}\}) = [\mathsf{ababab}]$. But since $\{\mathsf{a}, \mathsf{b}\} = \{\mathsf{b}, \mathsf{a}\}$, it must also hold that $f^\sharp(\{\mathsf{a}, \mathsf{b}\}) = f^\sharp(\{\mathsf{b}, \mathsf{a}\}) = f^\sharp(\{\mathsf{b}\}) :: f^\sharp(\{\mathsf{a}\}) = [\mathsf{bababa}]$. Thus, no such function $f^\sharp$ can exist – $N(\mathsf{A})$ is too specific.

Exchanging the rôles of $N(\mathsf{A})$ and $Mon(\mathsf{A})$ does not lead to the same problem. For *any* function $f$ from a set $A$ to the underlying set of any other monoid $M$, there indeed exists *precisely one* monoid homomorphism $f^\sharp$ that emulates $f$ such that $f = f^\sharp \circ i_A$, i.e., $\forall a \in A : f(a) = f^\sharp(i_A(a))$ (see Figure 6.2 or [Awodey, 2010, p. 21] for a proof). This fact characterizes that $Mon(A)$ is called the *free monoid* over $A$, being the most general monoid a set can be mapped to. Note that the existence of $f^\sharp$ corresponds to a "no confusion" argument since no elements are equated that should not be whereas the uniqueness of $f^\sharp$ relates to a "no junk" argument: If, for instance, we used $Mon'(A) = ((A \cup \{w\})^*, ::, \epsilon)$ with $w \notin A$, then we are free to chose the value of $f^\sharp([w])$ (a "junk element") as it is not constrained by the requirement $f = f^\sharp \circ i_A$ – in contrast to all elements in $A$. Generalizing from this example, category theory

allows to state this relationship between algebraic structures formally (see Definition 6.2).

## 6.3 The Free Partial Valuation Structure over a Partial Order

Motivated by the goal of finding the most general PVS to encode constraint preferences, the search for the free PVS over a partial order $P$ answers a more fundamental problem:

*Which ordering decisions always have to hold if we extend any partial order with a combination operation (multiplication) and neutral top element?*

More formally, this is the case if we have several soft constraints $\mu_1, \dots, \mu_n$ that each grade an assignment $\theta$ in the same partial order $P$ and we take a product $\mu_1(\theta) \cdot \dots \cdot \mu_n(\theta)$. Which $\preceq$-relations *must* certainly hold if we compare $\mu_1(\theta) \cdot \dots \cdot \mu_n(\theta)$ with $\mu_1(\theta') \cdot \dots \cdot \mu_n(\theta')$? How shall we even represent these products?

A seemingly obvious choice would be to collect all soft constraints' valuations as a set, i.e., $\{p_1, \dots, p_n\}$. Each $p \in P$ could then individually be represented by the unit morphism $\eta(p) = \{p\}$ and then combined using set union. Since $\emptyset$ should be top in a PVS, we aim to order the sets by size and according to $P$. That means, we want $X \preceq \emptyset$ for any set $X$ and $\eta(p_1) = \{p_1\} \preceq \{p_2\} = \eta(p_2)$ if $p_1 \leq_P p_2$. Both cases are covered by the Smyth-ordering over sets (cf. Section 6.1). However, that approach does not yield a proper PVS if we consider that we can multiply elements $\{p_1\}$ with themselves: Assuming $p_1 \leq_P p_2$, also $\{p_1\} \preceq \{p_2\}$ holds. Combining with $\{p_1\}$ on both sides yields $\{p_1\} \preceq \{p_1, p_2\}$, by monotonicity. But, by the definition of the Smyth-ordering, we also have $\{p_1, p_2\} \preceq \{p_1\}$ and thus antisymmetry is violated.

It turns out that the idempotency of set union is the culprit, in particular the fact that $\eta(p_1) \cup \eta(p_1) = \eta(p_1)$. This fact is *not* required by PVS axioms. Instead, commutativity and associativity provide a hint about the underlying set of the free PVS: The free monoid over a set $A$ uses $A^*$, finite lists over $A$, embedded by $\eta'(a) = [a]$ and combined with concatenation :: since we only need associativity: $\eta'(a) :: (\eta'(b) :: \eta'(c)) = (\eta'(a) :: \eta'(b)) :: \eta'(c) = [a, b, c]$ (see Section 6.2). For the free PVS, we additionally need to equate $\eta(a) \cup \eta(b)$ with $\eta(b) \cup \eta(a)$, but again, not necessarily $\eta(a) \cup \eta(a)$ with $\eta(a)$. This is precisely what we achieve with $\mathcal{M}_{\text{fin}}(P)$, finite multisets over $P$ and $\eta(a) = \wr a \int$. Taking plain sets over $P$ would additionally assume idempotency and is thus too specific.[2]

### 6.3.1 The Free PVS as Single-Predecessors-Lifting

Figure 6.3 instantiates Definition 6.2 for the task of proving that $PVS\langle P \rangle$ is indeed the free PVS over a partial order $P$. We start in the category PO of partial orders as objects and monotone functions as morphisms and map to PVS, the category of partial valuation structures as objects and PVS-homomorphisms as morphisms. To switch between partial orders and partial valuation structures we need appropriate functors. First, the (free) functor $PVS\langle P \rangle$:

$$PVS\langle P \rangle = (\mathcal{M}_{\text{fin}}(P), \uplus, \wr\int, \preceq_P) \ ,$$
$$PVS\langle \varphi : P \to Q \rangle = \lambda \wr p_1, \dots, p_n \int . \wr \varphi(p_1), \dots, \varphi(p_n) \int \ .$$

[2]Interestingly enough, the fact that partial valuation structures need not be idempotent in general (e.g., weighted constraints) disallows a straightforward extension of local consistency to soft constraints [Cooper and Schiex, 2004].

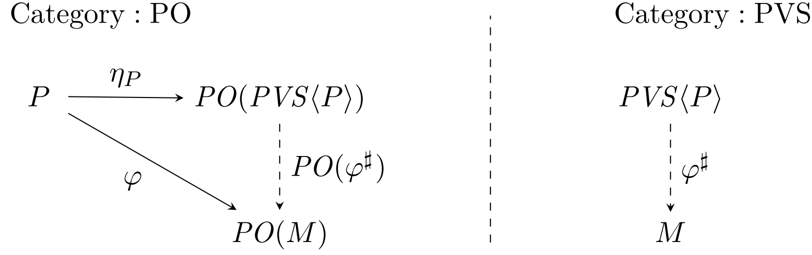



Figure 6.3: Diagram of the free PVS over a partial order. For an arbitrary PVS $M$ that we map into from a partial order $P$ using $\varphi$, we can lift this mapping to $\varphi^\sharp : PVS\langle P\rangle \to M$ such that $\varphi = PO(\varphi^\sharp) \circ \eta_P$. Consequently, $PVS\langle P\rangle$ only identifies and orders elements as absolutely required by PVS axioms – it is most general.

In the other direction, the (forgetful) functor $PO : \text{PVS} \to \text{PO}$ is defined by

$$PO(M) = (|M|, \leq_M) \ ,$$
$$PO(\varphi : M \to N) = \varphi \ .$$

Starting from a partial order $P$, commutativity and associativity motivate the underlying set $\mathcal{M}_{\text{fin}}(P)$. We can also justify each rule of the Smyth-ordering over multisets by applying Definition 6.2. First, as each $p \in |P|$ is found in $\mathcal{M}_{\text{fin}}(P)$ by $\eta_P(p) = \wr p \int$ and $\eta_P$ is a monotone function, we have that $p_1 \leq_P p_2 \Rightarrow \wr p_1 \int \preceq_P \wr p_2 \int$. This ensures that $P$ is preserved over their embedded counterparts. The other rule $T \supseteq U \Rightarrow T \preceq_P U$ stems from the fact that the neutral element is the top of the ordering in a PVS – which is the most prevalent choice in soft constraints [Meseguer et al., 2006]. This implies $m \cdot_M n \leq_M m$ since $n \leq_M \varepsilon_M \Rightarrow m \cdot n \leq_M m$, by monotonicity. Consequently, for the free PVS, $\wr m, n \int \preceq_P \wr m \int$ needs to hold, as does $T \preceq_P \wr\int$, both of which are represented by the above rule. Dually, we would have $T \subseteq U \Rightarrow T \preceq_P U$, had we defined the neutral element to be bottom of the ordering.

Next, we have to confirm that $PVS\langle P\rangle = \langle \mathcal{M}_{\text{fin}}(P), \uplus, \wr\int, \preceq_P\rangle$ is a partial valuation structure, to begin with. Associativity and commutativity of $\uplus$ and neutrality of $\wr\int$ with respect to $\mathcal{M}_{\text{fin}}(P)$ are obvious, we have already discussed reflexivity and transitivity of $\preceq_P$ as well as monotonicity of $\uplus$ with respect to $\preceq_P$ in Section 6.1. To show antisymmetry of $\preceq_P$, we prove a result (visualized in Figure 6.4) that also turns out to be useful later on when we implement the Smyth-ordering as a MiniZinc predicate to be used in search. To do so, we introduce a bit of notation to "unfold" a multiset $T$ into a set representation $\mathcal{S}(T)$, e.g., $\mathcal{S}(\wr x, x, y \int) = \{(1, x), (2, x), (1, y)\}$. Formally, for a multiset $T = \wr l_1 x_1, \ldots, l_n x_n \int \in \mathcal{M}_{\text{fin}}(X)$ with $l_1, \ldots, l_n > 0$ and $x_i \neq x_j$ if $i \neq j$, let $\mathcal{S}(T) = \bigcup_{1 \leq i \leq n} \{(j, x_i) \mid 1 \leq j \leq l_i\}$.

**Lemma 6.1** (Witness for $\preceq_P$)**.** *$T \preceq_P U$ if, and only if, there is an injective map $h : \mathcal{S}(U) \to \mathcal{S}(T)$ (called a* witness *function) with $p \leq_P q$ if $h(j, q) = (k, p)$ for all $(j, q) \in \mathcal{S}(U)$.*

*Proof.* Let first $T \preceq_P U$ hold. We restrict our attention w.l.o.g. to the case $T \neq U$ as otherwise the claim trivially holds. Then there is a sequence of multisets $T_1, \ldots, T_n \in \mathcal{M}_{\text{fin}}(P)$ with $n > 1$ such that $T_1 = T$, $T_n = U$, and for each $1 \leq i < n$, either $T_i \supseteq T_{i+1}$ or $T_i = T_i' \uplus \wr p \int$ and $T_{i+1} = T_i' \uplus \wr q \int$ with $p \leq_P q$. As required in the claim, for each $1 \leq i < n$ there is a witness $h_i : \mathcal{S}(T_{i+1}) \to \mathcal{S}(T_i)$ as follows: If $T_i \supseteq T_{i+1}$, then we choose $h_i = \text{id}_{\mathcal{S}(T_i)}$. If $T_i = T_i' \uplus \wr p \int$ and

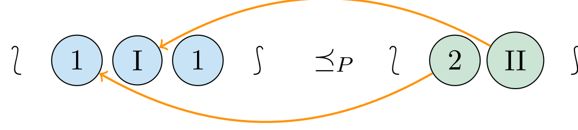



Figure 6.4: An exemplary witness function for the Smyth-ordering that assigns every item on the right side a unique lower (w.r.t. $\leq_P$) partner on the left side. More elements on the left side are acceptable, i.e., the witness needs not be surjective. Reprinted from Figure 4.3.

$T_{i+1} = T'_i \uplus \wr q \int$ with $p \leq_P q$, then we choose $h_i = \mathrm{id}_{\mathcal{S}(T'_i)} \cup \{(j,p) \mapsto (k,q)\}$ where $j = \max\{l \mid (l,p) \in \mathcal{S}(T'_i)\} + 1$ and $k = \max\{l \mid (l,q) \in \mathcal{S}(T'_i)\} + 1$. Then $h_1 \circ \ldots \circ h_{n-1} : \mathcal{S}(U) \to \mathcal{S}(T)$ is a witness function.

For the converse, we prove that if $h : \mathcal{S}(U) \to \mathcal{S}(T)$ is a witness function, then $T \preceq_P U$ by induction on the cardinality of $\mathcal{S}(U)$. Let $h : \mathcal{S}(U) \to \mathcal{S}(T)$ be given. If $|\mathcal{S}(U)| = 0$, then $T \preceq_P \wr\int = U$. Now let $|\mathcal{S}(U)| > 0$ and let $(j,q) \in \mathcal{S}(U)$ such that $j$ is maximal. Then $h(j,q) = (k,p)$ with $p \leq_P q$. Let $T', U' \in \mathcal{M}_{\mathrm{fin}}(P)$ be defined by $T = T' \uplus \wr p \int$ and $U = U' \uplus \wr q \int$. To construct a witness function between $T'$ and $U'$ and apply the induction hypothesis, we define $g : \mathcal{S}(T) \to \mathcal{S}(T')$ by $g(l,r) = (l,r)$ if $r \neq p$ or $l < k$, and $g(l,p) = (l-1,p)$ if $l > k$. Essentially, $g$ closes possible “gaps” in the image of $h$. Then $\mathcal{S}(U') = \mathcal{S}(U) \setminus \{(j,q)\}$ and $h' : \mathcal{S}(U') \to \mathcal{S}(T')$ defined as $h' = g \circ h$ is a witness function between $T'$ and $U'$. By induction hypothesis, hence, $T' \preceq_P U'$ and thus, by the monotonicity of $\preceq_P$ (see Section 6.1), $T = T' \uplus \wr p \int \preceq_P U' \uplus \wr p \int \preceq_P U' \uplus \wr q \int = U$. □

The witness function can be interpreted as assigning an “inferior” to every element on the right-hand side. To see the antisymmetry of the Smyth-ordering[3], assume for a contradiction that there are $T$ and $U$ with both $T \preceq_P U$ and $U \preceq_P T$, but $T \neq U$ and choose one $T$ with minimal cardinality satisfying this property. Then $T$ has to be non-empty. Let $f : \mathcal{S}(U) \to \mathcal{S}(T)$ and $g : \mathcal{S}(T) \to \mathcal{S}(U)$ be witnessing maps for $T \preceq_P U$ and $U \preceq_P T$, respectively. Choose an element $(j,q) \in \mathcal{S}(U)$ such that $q$ is minimal w.r.t. $\leq_P$ in $U$. Then there is an inferior $(k,p) = f(j,q)$ in $\mathcal{S}(T)$ with $p \leq_P q$. If $p \neq q$, as $U \preceq_P T$ holds as well, there would be yet another inferior $g(k,p) = (j',q') \in \mathcal{S}(U)$ such that $q' \leq_P p$ and thus $q' \leq_P p <_P q$, contradicting the minimality of $q$ in $U$; thus $f(j,q) = (k,q)$. Assume, without loss of generality, that $j$ and $k$ are maximal. Remove the occurrence of $p$ from $T$ and $U$, obtaining $T'$ and $U'$, respectively. Then $T' \preceq_P U'$ and $U' \preceq_P T'$ hold as well, since the reduced-domain functions $f' : \mathcal{S}(T') \to \mathcal{S}(U')$ with $f'(l,p) = f(l,p)$ and, similarly, $g' : \mathcal{S}(U') \to \mathcal{S}(T')$, are witnessing maps. This contradicts the assumed minimality of $T$. Thus, $PVS\langle P \rangle$ fulfills all axioms of a partial valuation structure and we are ready to show that it is indeed free.

**Lemma 6.2** (Free PVS)**.** *$PVS\langle P \rangle$ is the free PVS over the partial order $P$.*

*Proof.* Let $P$ be a partial order $(|P|, \leq_P)$ and $\varphi : P \to PO(M)$ be a PVS-homomorphism into the underlying partial order of any PVS $M$. To show the existence of a lifted variant of $\varphi$, we define $\varphi^\sharp : PVS\langle P \rangle \to M$ as a PVS-morphism by

$$\varphi^\sharp(\wr p_1, \ldots, p_n \int) = \varphi(p_1) \cdot_M \ldots \cdot_M \varphi(p_n)$$

[3]Curiously enough, the Smyth-ordering on mere sets is *not* antisymmetric. It is just a quasi-ordering.

for all $\wr p_1, \ldots, p_n \int \in \mathcal{M}_{\text{fin}}(P)$, where, if $n = 0$, $\varphi^\sharp(\wr\int) = \varepsilon_M$. This is well-defined, i.e., $\varphi^\sharp$ is indeed a PVS-homomorphism, since $\varphi^\sharp(\wr p_1, \ldots, p_m \int \uplus \wr q_1, \ldots, q_n \int) = \varphi(p_1) \cdot_M \ldots \cdot_M \varphi(p_m) \cdot_M \varphi(q_1) \cdot_M \ldots \cdot_M \varphi(q_n) = \varphi^\sharp(\wr p_1, \ldots, p_m \int) \cdot_M \varphi^\sharp(\wr q_1, \ldots, q_n \int)$, $\varphi^\sharp(\wr\int) = \varepsilon_M$, and, if $T \leq_{PVS\langle P\rangle} U$, then $\varphi^\sharp(T) \leq_M \varphi^\sharp(U)$: We consider the generating cases for one step of the Smyth-ordering as it is straightforward to consider the extension to sequences $T_1, \ldots, T_n$ as done in the proof of the witness function. Assume $T \leq_{PVS\langle P\rangle} U$. Either $T = \wr p \int \uplus T'$ and $U = \wr q \int \uplus T'$ with $p \leq_P q$. Then $\varphi^\sharp(T) = \varphi(p) \cdot_M \varphi^\sharp(T')$ and $\varphi^\sharp(U) = \varphi(q) \cdot_M \varphi^\sharp(T')$. And since $\varphi(p) \leq_M \varphi(q)$ due to $\varphi$ being a PO-morphism, we have $\varphi^\sharp(T) \leq_M \varphi^\sharp(U)$, by monotonicity of $\cdot_M$. Or, it is the case that $T \supseteq U$. Then $T = U \uplus T'$ ($T'$ may be empty) and thus $\varphi^\sharp(T) = \varphi^\sharp(T') \cdot_M \varphi^\sharp(U) \leq_M \varphi^\sharp(U)$, by the PVS axiom $m \cdot n \leq_M m$. Consequently $\varphi^\sharp$ is a PVS-homomorphism.

Moreover, $\varphi = PO(\varphi^\sharp) \circ \eta_P$ with $\eta_P(p) = \wr p \int$ for all $p \in |P|$, since $PO(\varphi^\sharp)(\eta_P(p)) = \varphi^\sharp(\wr p \int) = \varphi(p)$; hence the diagram in Figure 6.3 commutes.

Finally, $\varphi^\sharp$ is unique with this property: Assume there would be another PVS-homomorphism $\psi : PVS\langle P\rangle \to M$ that satisfies $PO(\psi) \circ \eta_P = \varphi$. Due to this requirement, we have $\psi(\wr p \int) = \varphi(p)$ for every $p \in |P|$. Thus, for $\psi$, we have $\psi(\wr\int) = \varepsilon_M = \varphi^\sharp(\wr\int)$ and

$$\begin{aligned}\psi(\wr p_1, \ldots, p_n \int) &= \psi(\wr p_1 \int) \cdot_M \ldots \cdot_M \psi(\wr p_n \int) \\ &= \varphi(p_1) \cdot_M \ldots \cdot_M \varphi(p_n) = \varphi^\sharp(\wr p_1 \int) \cdot_M \ldots \cdot_M \varphi^\sharp(\wr p_n \int) \\ &= \varphi^\sharp(\wr p_1, \ldots, p_n \int)\end{aligned}$$

since $\psi$ is a PVS-homomorphism and by the previous remark. Hence $\varphi^\sharp = \psi$, as claimed, and $PVS\langle P\rangle$ is indeed the free partial valuation structure over a partial order. □

This concludes our theoretical considerations of the free partial valuation structure. An example of an *SCSP* employing the free PVS is depicted in Figure 6.7 in Section 6.5. It actually uses soft constraints that directly map to the free PVS (i.e., $\mathcal{M}_{\text{fin}}(P)$ rather than $P$). For constraint preferences, we only distinguish between $\wr c_1 \int$ and $\wr\int$. Both the free PVS and a specialized type for constraint preferences are available in MiniBrass (cf. Section 4.3.2).

### 6.3.2 The Transitive-Predecessors-Lifting for Constraint Preferences

But of course, this encoding using the free PVS results in single-predecessor-dominance, which is only one of the possible dominance properties in constraint preferences (see Section 5.1.1). We have not considered a PVS-encoding for transitive-predecessors, yet. Put differently, the proofs in this section were only valid for SPD and we cannot simply switch the ordering. However, in Section 5.1.1, we described transitive-predecessor-dominance (TPD) as an alternative lifting of constraint preferences to violation sets. Yet, we have not presented an adequate PVS that offers TPD to modelers. Our construction of an appropriate TPD-PVS encompasses an intermediate ordering $\sqsubseteq_P$ that will also turn out to be helpful for our encoding of constraint hierarchies as PVS in Section 7.1.

**Definition 6.3 – Sym-Diff-Hoare-ordering over Multisets**
Let $P$ be any partial order. The Sym-Diff-Hoare-ordering on $\mathcal{M}_{\text{fin}}(P)$ is the binary relation $\sqsubseteq_P \subseteq \mathcal{M}_{\text{fin}}(P) \times \mathcal{M}_{\text{fin}}(P)$, defined by

$$W_1 = T \uplus V \sqsubseteq_P U \uplus V = W_2 \iff \forall p \in T \,.\, \exists q \in U \,.\, p <_P q$$

where $T$ and $U$ have no common elements, i.e., $V$ is equal to the intersection of $W_1$ and $W_2$.

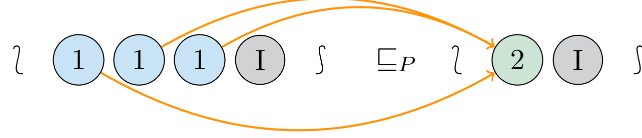



Figure 6.5: An example of the Sym-Diff-Hoare-ordering. Assume $|P| = \{\mathrm{I}, \mathrm{II}, 1, 2\}$ and their usual ordering (Roman and Arabic numerals are incomparable). To visualize $T \uplus V \sqsubseteq_P U \uplus V$, gray indicates the shared part $V$, $T$ is marked blue, and $U$ is green.

Figure 6.5 visualizes an instance of this relation. Clearly, if $W_1 \sqsubseteq_P W_2$, for every $w \in W_1$ there is at least one $v \in W_2$ with $w \leq_P v$. Also, $\sqsubseteq_P$ is monotonic w.r.t. $\uplus$. We proceed to show that $(\mathcal{M}_{\text{fin}}(P), \sqsubseteq_P)$ is indeed a partial order. By definition, $\sqsubseteq_P$ is obviously reflexive ($T$ and $U$ are empty) and antisymmetric: Assume that both $T \uplus V \sqsubseteq_P U \uplus V$ and $U \uplus V \sqsubseteq_P T \uplus V$ hold. If $T \uplus V = U \uplus V$, we are done since $T$ and $U$ are empty. Otherwise let $t \in \text{Max}^{\leq_P}(T)$ then there is a $u \in U$ with $t <_P u$. But conversely, there must also be a $t' \in T$ with $t <_P u <_P t'$, contradicting the maximality of $t$. In order to show transitivity, assume multisets $W_1$, $W_2$, and $W_3$ such that $W_1 \sqsubseteq_P W_2$ and $W_2 \sqsubseteq_P W_3$ hold. Then there are unique partitionings $T_{ij}, U_{ij}, V_{ij}$ such that $T_{ij}$ and $U_{ij}$ have no common elements and

$$
\begin{aligned}
W_1 &= T_{12} \uplus V_{12} = T_{13} \uplus V_{13} \ , \\
W_2 &= U_{12} \uplus V_{12} = T_{23} \uplus V_{23} \ , \\
W_3 &= U_{13} \uplus V_{13} = U_{23} \uplus V_{23} \ .
\end{aligned}
$$

Without loss of generality, we assume that there is no common element in all three of $W_1$, $W_2$, and $W_3$: From the definition of $\sqsubseteq_P$ we can see that $W_1 \sqsubseteq_P W_2 \Leftrightarrow W_1 \uplus \wr p \int \sqsubseteq_P W_2 \uplus \wr p \int$ since $W_1 \setminus W_2 = (W_1 \uplus \wr p \int) \setminus (W_2 \uplus \wr p \int)$ and $W_2 \setminus W_1 = (W_2 \uplus \wr p \int) \setminus (W_1 \uplus \wr p \int)$. Hence, if there were an element $g$ in each $W_i$, we would remove it to obtain $W_i' = W_i \setminus \wr g \int$ and prove that $W_1' \sqsubseteq_P W_3'$ follows from $W_1' \sqsubseteq_P W_2'$ and $W_2' \sqsubseteq_P W_3'$, and which is equivalent to $W_1 \sqsubseteq_P W_3$. As a consequence of this assumption, since $W_1' \sqsubseteq_P W_2'$ and $W_2' \sqsubseteq_P W_3'$, it must be that for every $p \in W_1'$ there is a larger $q \in W_3'$ since it cannot be that $p$ “remains equal” in $W_1$, $W_2$, and $W_3$ – we would then have a common element of all three of them.

We now have to prove that for every $p \in T_{13}$ there is a $q \in U_{13}$ such that $p <_P q$ since that includes every element in $T_{13}$. Let thus $p_{13} \in T_{13}$. Our proof proceeds by two case distinctions with regards to partition locations. First, we distinguish on what partition of $W_1 = T_{12} \uplus V_{12}$ the element $p_{13}$ resides in, either $p_{13} \in T_{12}$ or $p_{13} \in V_{12}$ (or in both):

1. $p_{13} \in T_{12}$: Then there is a $q_{12} \in U_{12}$ with $p_{13} <_P q_{12}$. We have again to distinguish where $q_{12} \in W_2$ is w.r.t. $T_{23} \uplus V_{23}$.
   - 1.1. $q_{12} \in T_{23}$: Then there is a $q_{23} \in U_{23}$ with $p_{13} <_P q_{12} <_P q_{23}$. With regards to $T_{13}$ and $U_{13}$, if $q_{23} \in U_{13}$, we are done. Otherwise, we know that $q_{23} \in V_{13}$.
   - 1.2. $q_{12} \in V_{23}$: If $q_{12} \in U_{13}$, then we are done, or $q_{12} \in V_{13}$ cannot be since otherwise $q_{12}$ would be in both $V_{12}$ and $V_{23}$ and thus in $W_1$, $W_2$, and $W_3$.
2. $p_{13} \in V_{12}$: Then $p_{13} \in T_{23}$ or $p_{13} \in V_{23}$ (or both).
   - 2.1. $p_{13} \in T_{23}$: Then there is a $q_{23} \in U_{23}$ with with $p_{13} <_P q_{23}$. If $q_{23}$ is also in $U_{13}$ we are done. Otherwise $q_{23} \in V_{13}$.

2.2. $p_{13} \in V_{23}$: This case is impossible since $p_{13} \in V_{12}$ implies that $p_{13} \in W_1$ and $p_{13} \in W_2$ whereas $p_{13} \in V_{23}$ implies $p_{13} \in W_3$. Hence, we would have a common element of $W_1$, $W_2$, and $W_3$, contradicting our assumption.

In the remaining unsettled cases of (1.1) and (2.1), we always find some $q \in V_{13}$ with $p_{13} <_P q$. But since $q \in W_1$, and $W_1 \sqsubseteq_P W_2 \sqsubseteq_P W_3$, we can apply the same case distinctions to $q$ that we did for $p_{13}$ and find a $q' \in W_3$ with $q <_P q'$ that is either in $U_{13}$ or, again, in $V_{13}$ ($q' = q$ is impossible since then $q$ would be a common element of all $W_1$, $W_2$, and $W_3$). Hence $q <_P q'$ holds and $q$ is either in $U_{13}$ or in $V_{13}$. If $q \in U_{13}$, the proof is complete. Finally, if otherwise $q' \in V_{13}$ we could find yet another greater item $q''$ and so on. This process cannot go on infinitely often since $W_3$ and thus $V_{13}$ are finite and we thus cannot build an infinitely strictly ascending chain in $V_{13}$. Hence, eventually we must find that $q' \in U_{13}$.
If $P$ is total, then $\sqsubseteq_P$ is total, too. Again, we may form the upper counterpart of $\sqsubseteq_P$ by defining $T \sqsubseteq^P U$ if, and only if, $U \sqsubseteq_{P^{-1}} T$.
With this ordering, we are ready to formalize the TPD-lifting for PVS – equally based on finite multisets. Since users typically only use a directed acyclic graph (DAG) to specify their constraint preferences (see Section 4.3.2), we also start with DAGs. Let thus $G$ be a DAG and consider the *transitive-predecessors lifting* $\prec^G_{\mathrm{TPD}} \subseteq (\mathcal{M}_{\mathrm{fin}}\,|G|) \times (\mathcal{M}_{\mathrm{fin}}\,|G|)$ of $G$ to the finite multisets $\mathcal{M}_{\mathrm{fin}}\,|G|$ over the elements of $G$ that we previously defined for sets in (TPD):

$$T \prec^G_{\mathrm{TPD}} T \uplus \wr g \int \;,$$
$$g_1, \ldots, g_n \to^+_G h \text{ implies } T \uplus \wr g_1, \ldots, g_n \int \prec^G_{\mathrm{TPD}} T \uplus \wr h \int \;.$$

Then $G^{\mathrm{TPD}} = (\mathcal{M}_{\mathrm{fin}}\,|G|, \prec^G_{\mathrm{TPD}})$ is also a DAG; we write $\leq^{\mathrm{TPD}}_G$ for $\geq_{PO\langle G^{\mathrm{TPD}}\rangle} = ((\prec^G_{\mathrm{TPD}})^*)^{-1}$, i.e., we again take the reflexive-transitive closure of $\prec^G_{\mathrm{TPD}}$ and invert it. Furthermore, $\leq^{\mathrm{TPD}}_G$ is monotonic w.r.t. multiset union and thus $(\mathcal{M}_{\mathrm{fin}}\,|G|, \uplus, \wr\int, \leq^{\mathrm{TPD}}_G)$ is a PVS if we can show that $\leq^{\mathrm{TPD}}_G$ is indeed a partial ordering. This can be done by referring to the Sym-Diff-Hoare-ordering.

**Lemma 6.3.** *For each DAG $G$, $\sqsubseteq^{PO\langle G\rangle^{-1}} = \leq^{\mathrm{TPD}}_G$.*

*Proof.* To first show $\leq^{\mathrm{TPD}}_G \subseteq \sqsubseteq^{PO\langle G\rangle^{-1}}$, we only need to consider each of the defining clauses. Hence, assume first that $T \uplus \wr g \int \leq^{\mathrm{TPD}}_G T$, with $g \in |G|$. Then $T \sqsubseteq_{PO\langle G\rangle} T \uplus \wr g \int$ holds as well since $T \setminus T \uplus \wr g \int = \wr\int$ and thus $T \uplus \wr g \int \sqsubseteq^{PO\langle G\rangle^{-1}} T$. Similarly, if $T \uplus \wr h \int \leq^{\mathrm{TPD}}_G T \uplus \wr g_1, \ldots, g_n \int$ with $g_i \to^+_G h$ holds, we have $T \uplus \wr g_1, \ldots, g_n \int \sqsubseteq_{PO\langle G\rangle} T \uplus \wr h \int$ since for every $g_i$ we have $g_i \to^+_G h$ and thus $g_i <_{PO\langle G\rangle} h$. Consequently, $T \uplus \wr h \int \sqsubseteq^{PO\langle G\rangle^{-1}} T \uplus \wr g_1, \ldots, g_n \int$.

For proving that $\sqsubseteq^{PO\langle G\rangle^{-1}} \subseteq \leq^{\mathrm{TPD}}_G$, let $T \uplus V \sqsubseteq^{PO\langle G\rangle^{-1}} U \uplus V$, i.e., $U \uplus V \sqsubseteq_{PO\langle G\rangle} T \uplus V$, hold, such that $T$ and $U$ have no common elements; we have to show that $U \uplus V\ (\prec^G_{\mathrm{TPD}})^*\ T \uplus V$ holds as well, i.e., $T \uplus V \leq^{\mathrm{TPD}}_G U \uplus V$. We proceed by induction on the size of $U$. If $U = \wr\int$, then either $T$ is empty and we have equality, or the claim holds by repeated application of the first defining clause of $\prec^G_{\mathrm{TPD}}$, i.e., $V (\prec^G_{\mathrm{TPD}})^* V \uplus T$.

Now let $U \neq \wr\int$ and let $g$ be a maximal element in $U$ w.r.t. $\leq_{PO\langle G\rangle}$. Then there is an $h$ in $T$ with $g <_{PO\langle G\rangle} h$ by definition of $\sqsubseteq_{PO\langle G\rangle}$. Now split $U$ as $U = \wr g_1, \ldots, g_n \int \uplus U'$, where $g_1, \ldots, g_n$ are *all* elements in $U$ that are dominated by $h$ w.r.t. $<_{PO\langle G\rangle}$; in particular, $g = g_i$ for some $i$. Then we see that $\wr g_1, \ldots, g_n \int \prec^G_{\mathrm{TPD}} \wr h \int$ by the second defining clause of $\prec^G_{\mathrm{TPD}}$. Let also $T = \wr h \int \uplus T'$, i.e., we remove one occurrence of $h$ to get $T'$. Then $U' \uplus V \sqsubseteq_{PO\langle G\rangle} T' \uplus V$ holds since every element in $U'$ not dominated by $h$ still has a dominator in $T'$ because we

know $U \uplus V \sqsubseteq_{PO\langle G\rangle} T \uplus V$. Hence, $T' \uplus V \sqsubseteq^{PO\langle G\rangle^{-1}} U' \uplus V$ is true, and, by the induction hypothesis, we obtain $T' \uplus V \leq_G^{\text{TPD}} U' \uplus V$, or, $U' \uplus V\ (\prec_{\text{TPD}}^G)^* \ T' \uplus V$. Combining these facts leads to:

$$\begin{aligned} U \uplus V = \wr g_1, \ldots, g_n \wr \uplus (U' \uplus V)\ (\prec_{\text{TPD}}^G)^* \wr g_1, \ldots, g_n \wr \uplus (T' \uplus V) & \quad \text{by ind. hyp.} \\ \prec_{\text{TPD}}^G \wr h \wr \uplus (T' \uplus V) = T \uplus V & \quad \text{by (TPD)} \end{aligned}$$

Hence, finally we have $T \uplus V \leq_G^{\text{TPD}} U \uplus V$. □

## 6.4 The Free C-Semiring over a Partial Valuation Structure

As mentioned before, (partial) valuation structures are not the only abstract algebraic framework for soft constraints in the literature. C-semirings constitute a particularly popular choice. They are purely algebraic by requiring a second "additive" operation instead of a partial ordering to form an (upper semi-)lattice. This idempotent, commutative, and associative operation is then used to induce a partial ordering. Moreover, any c-semiring is bounded above and below by two designated constants. We will proceed to show that every c-semiring gives rise to a bounded PVS, and, conversely, every PVS can be extended to a c-semiring by means of another free construction – although not every PVS *is* a c-semiring since the additive operation, in fact, returns a supremum which need not exist in a PVS.

This section, therefore, extends previous work that examined the similarities between c-semirings and (totally ordered bounded) valuation structures [Bistarelli et al., 1999]. The authors identified a valuation structure with every *totally ordered* c-semiring only. For branch-and-bound and similar search algorithms, a partial ordering indeed suffices (see Section 4.4.2 or [Junker, 2009; Meseguer et al., 2006]). The main algorithmic advantage of having a second algebraic operation instead of the partial ordering lies in the thereby guaranteed existence of a supremum. This least upper bound can be used for non-serial dynamic programming, i.e., variable elimination. These algorithms may, however, return an unreachable optimal solution degree (e.g., the supremum of *all* reachable optima). From a practical perspective, this free construction of a c-semiring from a PVS alleviates the need to model in c-semirings in the first place. If a fruitful algorithmic technique for c-semirings (relying on the addition) is discovered, it can also be applied to a PVS when raised to the free c-semiring. We sketch such an application in Section 6.5 but first actually derive the free c-semiring over a PVS.

Formally, a *c-semiring* [Bistarelli et al., 1997] $A = (|A|, \oplus_A, \otimes_A, \mathbf{0}_A, \mathbf{1}_A)$ is given by an (underlying) set $|A|$, two binary operations $\oplus_A, \otimes_A : |A| \times |A| \to |A|$, and two constants $\mathbf{0}_A, \mathbf{1}_A \in |A|$ such that the following axioms are satisfied:

– $\oplus_A$ is associative and commutative and has $\mathbf{1}_A$ as annihilator and $\mathbf{0}_A$ as neutral element

– $\otimes_A$ is associative and commutative, has $\mathbf{0}_A$ as annihilator and $\mathbf{1}_A$ as neutral element

– $\otimes_A$ distributes over $\oplus_A$

To preserve this structure, a *c-semiring homomorphism* $\varphi : A \to B$ from a c-semiring $A$ to a c-semiring $B$ is given by a map $\varphi : |A| \to |B|$ such that for all $a_1, a_2 \in |A|$:

1. $\varphi(a_1 \oplus_A a_2) = \varphi(a_1) \oplus_B \varphi(a_2), \qquad \varphi(a_1 \otimes_A a_2) = \varphi(a_1) \otimes_B \varphi(a_2)$
2. $\varphi(\mathbf{0}_A) = \mathbf{0}_B, \qquad \varphi(\mathbf{1}_A) = \mathbf{1}_B$

Consequently, the category cSRng of c-semirings has the c-semirings as objects and the c-semiring homomorphisms as morphisms. Note that in a c-semiring $A$ the operation $\oplus_A$ is

idempotent:

$$a \oplus_A a = (a \otimes_A \mathbf{1}_A) \oplus_A (a \otimes_A \mathbf{1}_A) = a \otimes_A (\mathbf{1}_A \oplus_A \mathbf{1}_A) = a \otimes_A \mathbf{1}_A = a \ .$$

Hence, $\oplus_A$ can be used to induce a partial ordering $\leq_A$ by interpreting it as the least upper bound: $a \leq_A b \Leftrightarrow a \oplus_A b = b$. Clearly, $\leq_A$ is reflexive due to the idempotency, transitive due to associativity, and antisymmetric due to commutativity of $\oplus_A$. With this definition, for all $a, b, c \in |A|$ it holds that

1. $\mathbf{0}_A \leq_A a \leq_A \mathbf{1}_A$;
2. $a \leq_A a \oplus_A b$ and $b \leq_A a \oplus_A b$;
3. if $a \leq_A c$ and $b \leq_A c$, then $a \oplus_A b \leq_A c$.

In particular, $a \oplus_A b$ is the supremum of $a$ and $b$ with respect to $\leq_A$. Also $\oplus_A$ is monotone w.r.t. $\leq_A$ in both arguments, i.e., $a \leq_A a'$ and $b \leq_A b'$ implies $a \oplus_A b \leq_A a' \oplus_A b'$. Additionally, the *combination* operation $\otimes_A$ is monotone w.r.t. the induced ordering $\leq_A$, since if $a \leq_A a'$ (i.e., $a \oplus_A a' = a'$) then $(a \otimes_A b) \oplus_A (a' \otimes_A b) = (a \oplus_A a') \otimes_A b = a' \otimes_A b$, i.e., $a \otimes_A b \leq_A a' \otimes b$, from which it follows that $a \leq_A a'$ and $b \leq_A b'$ implies $a \otimes_A b \leq_A a' \otimes b'$. Furthermore, for all $a, b \in |A|$, it holds that $a \otimes_A b \leq_A a$ and $a \otimes_A b \leq_A b$, since $(a \otimes_A b) \oplus_A a = (a \otimes_A b) \oplus_A (a \otimes_A \mathbf{1}_A) = a \otimes_A (b \oplus_A \mathbf{1}_A) = a \otimes_A \mathbf{1}_A = a$.

As a consequence, we can easily convert any c-semiring into a PVS by defining the functor $\mathit{PVS} : \mathrm{cSRng} \to \mathrm{PVS}$:

$$\begin{aligned} &\mathit{PVS}(A) = (|A|, \otimes_A, \mathbf{1}_A, \leq_A) \ , \\ &\mathit{PVS}(\varphi : A \to B) = \varphi \ . \end{aligned}$$

Note that $\mathit{PVS}(A)$ is a bounded PVS with $\bot_{\mathit{PVS}(A)} = \mathbf{0}_A$. This leaves us with the first part of a free construction between categories PVS and cSRng (cf. Definition 6.2). The opposite direction, constructing a c-semiring starting from a PVS, is not as obvious since the partial order of a PVS need not show suprema that are required to exist for the $\oplus$ operator (they clearly exist in total orders, making the conversion from totally ordered c-semirings to valuation structures more straightforward [Bistarelli et al., 1999]). For instance, in Figure 3.2, we saw that both $\lbag c_1 \rbag$ and $\lbag c_2, c_3 \rbag$ are upper bounds of $\lbag c_1, c_2 \rbag$ and $\lbag c_1, c_3 \rbag$ but they are incomparable.

When allowing partiality, we can always find an "artificial" supremum by collecting all (incomparable) valuations in a set and ordering these sets appropriately. Consider an arbitrary PVS $M = (|M|, \cdot_M, \varepsilon_M, \leq_M)$. We write $\mathcal{I}_{\mathrm{fin}}(M)$ to denote the set of finite sets composed of *incomparable* elements from $|M|$ (i.e., if $X \in \mathcal{I}_{\mathrm{fin}}(M)$ then for any $x \neq y \in X$ we have $x \parallel_M y$) and $\mathrm{Max}^{\leq_M}(X)$ to denote the maximal elements of $X$ with respect to $\leq_M$. For instance, if $|M| = \{1, 2, \mathrm{III}, \mathrm{IV}\}$ and $1 <_M 2$, $\mathrm{III} <_M \mathrm{IV}$, the sets $\{2, \mathrm{IV}\}$ or $\{1, \mathrm{III}\}$ are in $\mathcal{I}_{\mathrm{fin}}(M)$ but $\{1, \mathrm{III}, \mathrm{IV}\}$ is not and $\mathrm{Max}^{\leq_M}(|M|) = \{2, \mathrm{IV}\}$. We define the binary operations $\tilde{\cup}_M$ and $\tilde{\cdot}_M$ over $\mathcal{I}_{\mathrm{fin}}(M)$ by

$$\begin{aligned} &I \mathbin{\tilde{\cup}_M} J = \mathrm{Max}^{\leq_M}(I \cup J) \ , \\ &I \mathbin{\tilde{\cdot}_M} J = \mathrm{Max}^{\leq_M}\{m \cdot_M n \mid m \in I,\ n \in J\} \ . \end{aligned}$$

Clearly, $\tilde{\cup}_M$ inherits commutativity from $\cup$, and is idempotent since $\mathrm{Max}^{\leq_M}(I) = I$ for any set $I$ consisting of already incomparable elements. It is easy to check that it is also associative. Further, $\{\varepsilon_M\}$ is an annihilator for $\tilde{\cup}_M$ since $\varepsilon_M$ is the greatest element of $|M|$ with respect to $\leq_M$, and $\emptyset$ is its neutral element.

Similarly, $\tilde{\cdot}_M$ is obviously commutative since $\cdot_M$ is commutative. Dually to $\tilde{\cup}_M$, it has $\{\varepsilon_M\}$ as neutral element (since $\varepsilon_M$ is neutral in $M$) and $\emptyset$ as annihilator. For the associativity of $\tilde{\cdot}_M$, we have

$$\begin{aligned}
&I \mathbin{\tilde{\cdot}_M} (J \mathbin{\tilde{\cdot}_M} K) = \\
&\quad \mathrm{Max}^{\leq_M}\{m_I \cdot_M m_{JK} \mid m_I \in I,\ m_{JK} \in \mathrm{Max}^{\leq_M}\{m_J \cdot_M m_K \mid m_J \in J,\ m_K \in K\}\} = \\
&\quad \mathrm{Max}^{\leq_M}\{m_I \cdot_M m_J \cdot_M m_K \mid m_I \in I,\ m_J \in J,\ m_K \in K\}\} = \\
&\quad \mathrm{Max}^{\leq_M}\{m_{IJ} \cdot_M m_K \mid m_{IJ} \in \mathrm{Max}^{\leq_M}\{m_I \cdot_M m_J \mid m_I \in I,\ m_J \in J\},\ m_K \in K\} = \\
&\quad (I \mathbin{\tilde{\cdot}_M} J) \mathbin{\tilde{\cdot}_M} K\ ,
\end{aligned}$$

since $\mathrm{Max}^{\leq_M}\{m \cdot_M n \mid m \in I,\ n \in \mathrm{Max}^{\leq_M}(X)\} = \mathrm{Max}^{\leq_M}\{m \cdot_M n \mid m \in I,\ n \in X\}$ for all finite sets $X \subseteq |M|$. Finally, $\tilde{\cdot}_M$ distributes over $\tilde{\cup}_M$:

$$\begin{aligned}
&I \mathbin{\tilde{\cdot}_M} (J \mathbin{\tilde{\cup}_M} K) = \\
&\quad \mathrm{Max}^{\leq_M}\{m_I \cdot_M m_{JK} \mid m_I \in I,\ m_{JK} \in \mathrm{Max}^{\leq_M}(J \cup K)\} = \\
&\quad \mathrm{Max}^{\leq_M}\{m_I \cdot_M m_{JK} \mid m_I \in I,\ m_{JK} \in J \cup K\} = \\
&\quad \mathrm{Max}^{\leq_M}(\{m_I \cdot_M m_J \mid m_I \in I,\ m_J \in J\} \cup \{m_I \cdot_M m_K \mid m_I \in I,\ m_K \in K\}) = \\
&\quad \mathrm{Max}^{\leq_M}(\mathrm{Max}^{\leq_M}\{m_I \cdot_M m_J \mid m_I \in I,\ m_J \in J\} \cup \\
&\qquad\qquad \mathrm{Max}^{\leq_M}\{m_I \cdot_M m_K \mid m_I \in I,\ m_K \in K\}) = \\
&\quad (I \mathbin{\tilde{\cdot}_M} J) \mathbin{\tilde{\cup}_M} (I \mathbin{\tilde{\cdot}_M} K)\ ,
\end{aligned}$$

since $\mathrm{Max}^{\leq_M}(I \cup \mathrm{Max}^{\leq_M}(X)) = \mathrm{Max}^{\leq_M}(I \cup X)$ for all finite $X \subseteq |M|$. Thus, we conclude

**Lemma 6.4.** *$(\mathcal{I}_{\mathrm{fin}}(M), \tilde{\cup}_M, \tilde{\cdot}_M, \emptyset, \{\varepsilon_M\})$ is a c-semiring.* □

This structure will serve to define the object part of a free functor from PVS to cSRng.
At this point, it is worth noting that a similar construction of c-semiring addition and multiplication operations has been introduced by Rollón [2008], although starting from a given c-semiring instead of a PVS. She proves that when $A$ is a c-semiring, its so-called *frontier algebra* $\mathcal{A} = (\mathcal{I}(A) \setminus \{\emptyset\}, \tilde{\oplus}_A, \tilde{\otimes}_A, \{\mathbf{0}_A\}, \{\mathbf{1}_A\})$ again is a c-semiring, where $\mathcal{I}(A)$ are (possibly *infinite*) subsets of $|A|$ containing only pairwise incomparable elements w.r.t. $\leq_A$, and,

$$\begin{aligned}
&I \mathbin{\tilde{\oplus}_A} J = \mathrm{Max}^{\leq_A}(I \cup J)\ , \\
&I \mathbin{\tilde{\otimes}_A} J = \mathrm{Max}^{\leq_A}\{i \otimes_A j \mid i \in I,\ j \in J\}
\end{aligned}$$

for all $I, J \in \mathcal{I}(A) \setminus \{\emptyset\}$. The underlying set of the frontier algebra thus contains sets of arbitrary cardinality, not only finite sets as in our approach of the free construction. In fact, such infinite sets would correspond to "junk elements" (cf. Section 6.2), i.e., they would be unnecessary to have in the carrier set of the free c-semiring since we only have the finitary combination and supremum operation.

In [Rollón, 2008], the condition that only *non-empty* sets have to be considered is missing. The empty set has to be excluded, however, since otherwise $\emptyset \mathbin{\tilde{\otimes}_A} \{\mathbf{0}_A\} = \emptyset$, although $\{\mathbf{0}_A\}$ has to be the annihilator for $\tilde{\otimes}_A$, and $\emptyset \mathbin{\tilde{\oplus}_A} \{\mathbf{0}_A\} = \{\mathbf{0}_A\}$, i.e., $\emptyset \leq_{\mathcal{A}} \{\mathbf{0}_A\}$ contradicting that $\{\mathbf{0}_A\}$ has to be the smallest element w.r.t. $\leq_{\mathcal{A}}$. By contrast, in our approach, we have to consider $\emptyset$ as well in order to obtain a "fresh" bottom element of the free c-semiring over an arbitrary

PVS. If we only applied the construction of a free c-semiring to the sub-category of bounded PVS, we also could exclude $\emptyset$ and would obtain $\{\bot_M\}$ as bottom element of the free c-semiring over the bounded PVS $M$. However, the free PVS over a partial order – our original mission – clearly is not bounded.

To verify that we can design the morphism part of a free functor, it is useful to convince ourselves that the application of the maximum operator in a target structure subsumes the maximum operator in a source structure.

**Lemma 6.5** (Subsumption of Maximum). *Let $\varphi : M \to N$ be a PVS homomorphism. For finite sets $X \subseteq |M|$, we have* $\mathrm{Max}^{\leq_N}(\varphi(\mathrm{Max}^{\leq_M}(X))) = \mathrm{Max}^{\leq_N}(\varphi(X))$.

*Proof.* First, $\mathrm{Max}^{\leq_N}(\varphi(\mathrm{Max}^{\leq_M}(X))) \subseteq \mathrm{Max}^{\leq_N}(\varphi(X))$, since $\mathrm{Max}^{\leq_M}(X) \subseteq X$ holds which in turn implies $\varphi(\mathrm{Max}^{\leq_M}(X)) \subseteq \varphi(X)$.
To conversely show $\mathrm{Max}^{\leq_N}(\varphi(X)) \subseteq \mathrm{Max}^{\leq_N}(\varphi(\mathrm{Max}^{\leq_M}(X)))$, it suffices to show that for each $n \in \varphi(X)$ there is a (weakly dominating) $n' \in \varphi(\mathrm{Max}^{\leq_M}(X))$ such that $n \leq_N n'$: If $n \in \varphi(X)$ then $n = \varphi(m)$ for some $m \in X$. Either $m$ is maximal, in which case $n$ is obviously in $\varphi(\mathrm{Max}^{\leq_M}(X))$ as well. Otherwise, there is an $m' \in \mathrm{Max}^{\leq_M}(X)$ with $m \leq_M m'$, hence $n = \varphi(m) \leq_N \varphi(m')$, and $\varphi(m') \in \varphi(\mathrm{Max}^{\leq_M}(X))$. □

Finally, we define the functor $\mathit{cSRng}\langle - \rangle : \mathrm{PVS} \to \mathrm{cSRng}$ as

$$\mathit{cSRng}\langle M \rangle = (\mathcal{I}_{\mathrm{fin}}(M), \tilde{\cup}_M, \tilde{\cdot}_M, \emptyset, \{\varepsilon_M\}) ,$$
$$\mathit{cSRng}\langle \varphi : M \to N \rangle = \lambda \{m_1, \dots, m_k\} \in \mathcal{I}_{\mathrm{fin}}(M) \,.\, \mathrm{Max}^{\leq_N}\{\varphi(m_1), \dots, \varphi(m_k)\} .$$

We need to check (using Lemma 6.5) that $\mathit{cSRng}\langle \varphi : M \to N \rangle$ is indeed a c-semiring homomorphism from $\mathit{cSRng}\langle M \rangle$ to $\mathit{cSRng}\langle N \rangle$ for the functor to be well-defined:

$$\begin{aligned}
&\mathit{cSRng}\langle \varphi \rangle(\emptyset) = \emptyset , \quad \mathit{cSRng}\langle \varphi \rangle(\{\varepsilon_M\}) = \{\varphi(\varepsilon_M)\} = \{\varepsilon_N\} , \\
&\mathit{cSRng}\langle \varphi \rangle(I_1 \mathbin{\tilde{\cup}_M} I_2) = \mathit{cSRng}\langle \varphi \rangle(\mathrm{Max}^{\leq_M}(I_1 \cup I_2)) = \\
&\quad \mathrm{Max}^{\leq_N}(\varphi(\mathrm{Max}^{\leq_M}(I_1 \cup I_2))) = \mathrm{Max}^{\leq_N}(\varphi(I_1 \cup I_2)) = \mathrm{Max}^{\leq_N}(\varphi(I_1) \cup \varphi(I_2)) = \\
&\quad \mathit{cSRng}\langle \varphi \rangle(I_1) \mathbin{\tilde{\cup}_N} \mathit{cSRng}\langle \varphi \rangle(I_2) , \\
&\mathit{cSRng}\langle \varphi \rangle(I_1 \mathbin{\tilde{\cdot}_M} I_2) = \mathit{cSRng}\langle \varphi \rangle(\mathrm{Max}^{\leq_M}\{m_1 \cdot_M m_2 \mid m_1 \in I_1,\ m_2 \in I_2\})) = \\
&\quad \mathrm{Max}^{\leq_N}(\varphi(\mathrm{Max}^{\leq_M}\{m_1 \cdot_M m_2 \mid m_1 \in I_1,\ m_2 \in I_2\})) = \\
&\quad \mathrm{Max}^{\leq_N}\{\varphi(m_1 \cdot_M m_2) \mid m_1 \in I_1,\ m_2 \in I_2\} = \\
&\quad \mathrm{Max}^{\leq_N}\{\varphi(m_1) \cdot_N \varphi(m_2) \mid m_1 \in I_1,\ m_2 \in I_2\} = \\
&\quad \mathrm{Max}^{\leq_N}\{n_1 \cdot_N n_2 \mid n_1 \in \varphi(I_1),\ n_2 \in \varphi(I_2)\} = \\
&\quad \mathit{cSRng}\langle \varphi \rangle(I_1) \mathbin{\tilde{\cdot}_N} \mathit{cSRng}\langle \varphi \rangle(I_2) .
\end{aligned}$$

With these functors from PVS to cSRng and vice versa defined, we are ready to apply Definition 6.2 to the problem of finding the free c-semiring over a PVS, as depicted in Figure 6.6. As unit morphism, we define $\eta_M : M \to \mathit{PVS}(\mathit{cSRng}\langle M \rangle)$ for every PVS $M$ by $\eta_M(m) = \{m\}$. Now let $M$ be some PVS, $A$ a c-semiring, and $\varphi : M \to \mathit{PVS}(A)$ be a PVS-homomorphism. Again, we search a lifting $\varphi^\sharp$ that "emulates" (and extends) the PVS-homomorphism $\varphi$ at the c-semiring level, i.e., makes the diagram in Figure 6.6 commute by asserting that $\mathit{PVS}(\varphi^\sharp) \circ \eta_M = \varphi$. We define $\varphi^\sharp : \mathit{cSRng}\langle M \rangle \to A$ as a function $\varphi^\sharp : \mathcal{I}_{\mathrm{fin}}(M) \to |A|$ and need to show that it is a c-semiring homomorphism:

$$\varphi^\sharp(\{m_1, \dots, m_n\}) = \varphi(m_1) \oplus_A \cdots \oplus_A \varphi(m_n)$$

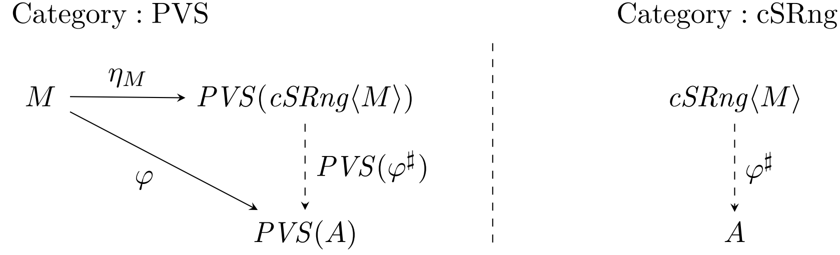



Figure 6.6: Diagram of the free c-semiring over a PVS. As with previous free constructions, $cSRng\langle M\rangle$ only identifies and orders elements as absolutely required by c-semiring axioms – it is again most general.

for all $\{m_1, \ldots, m_n\} \in \mathcal{I}_{\text{fin}}(M)$, where, if $n = 0$, $\emptyset$ is mapped to $\mathbf{0}_A$; $\varphi^\sharp$ is indeed a c-semiring homomorphism, since for the constants, $\varphi^\sharp(\mathbf{0}_{cSRng\langle M\rangle}) = \varphi^\sharp(\emptyset) = \mathbf{0}_A$ and $\varphi^\sharp(\mathbf{1}_{cSRng\langle M\rangle}) = \varphi^\sharp(\{\varepsilon_M\}) = \varphi(\varepsilon_M) = \varepsilon_{PVS(A)} = \mathbf{1}_A$. To show that $\varphi^\sharp$ preserves the operations $\tilde{\cdot}_M$ and $\tilde{\cup}_M$, we first note that for each finite set $\{m_1, \ldots, m_n\} \subseteq |M|$ (not necessarily composed of incomparable elements) it holds that $\varphi^\sharp(\text{Max}^{\leq_M}\{m_1, \ldots, m_n\}) = \varphi(m_1) \oplus_A \ldots \oplus_A \varphi(m_n)$: if some dominating $m_i \leq_M m_j$ exists, then $\varphi(m_i) \leq_{PVS(A)} \varphi(m_j)$ (since $\varphi$ is a PVS-homomorphism), hence, $\varphi(m_i) \oplus_A \varphi(m_j) = \varphi(m_j)$. We can thus "remove" each occurrence of the dominated $m_i$ in $\varphi(m_1) \oplus_A \ldots \oplus_A \varphi(m_n)$ since its dominator $m_j$ is included in that term. Therefore,

$$\begin{aligned}
&\varphi^\sharp(\{m_1, \ldots, m_k\} \mathbin{\tilde{\cup}_M} \{m_{k+1}, \ldots, m_n\}) = \varphi^\sharp(\text{Max}^{\leq_M}\{m_1, \ldots, m_n\}) = \\
&\quad \varphi(m_1) \oplus_A \ldots \oplus_A \varphi(m_n) = \\
&\quad (\varphi(m_1) \oplus_A \ldots \oplus_A \varphi(m_k)) \oplus_A (\varphi(m_{k+1}) \oplus_A \ldots \oplus_A \varphi(m_n)) = \\
&\quad \varphi^\sharp(\{m_1, \ldots, m_k\}) \oplus_A \varphi^\sharp(\{m_{k+1}, \ldots, m_n\}) \ .
\end{aligned}$$

Similarly, for two sets $I, J \in \mathcal{I}_{\text{fin}}(M)$

$$\begin{aligned}
&\varphi^\sharp(I \mathbin{\tilde{\cdot}_M} J) = \varphi^\sharp(\text{Max}^{\leq_M}\{m_1 \cdot_M m_2 \mid m_1 \in I,\ m_2 \in J\}) = \\
&\quad \textstyle\bigoplus_A\{\varphi(m_1 \cdot_M m_2) \mid m_1 \in I,\ m_2 \in J\} \stackrel{\text{PVS hom.}}{=} \\
&\quad \textstyle\bigoplus_A\{\varphi(m_1) \cdot_{PVS(A)} \varphi(m_2) \mid m_1 \in I,\ m_2 \in J\} = \\
&\quad \textstyle\bigoplus_A\{\varphi(m_1) \otimes_A \varphi(m_2) \mid m_1 \in I,\ m_2 \in J\} \stackrel{\text{distr.}}{=} \\
&\quad \textstyle\bigoplus_A\{\varphi(m_1) \mid m_1 \in I\} \otimes_A \bigoplus_A\{\varphi(m_2) \mid m_2 \in J\} = \varphi^\sharp(I) \otimes_A \varphi^\sharp(J) \ .
\end{aligned}$$

Thus, $\varphi^\sharp$ is a c-semiring homomorphism and additionally, $PVS(\varphi^\sharp)(\eta_M(m)) = \varphi(m)$, i.e., the diagram in Figure 6.6 commutes, and $\varphi^\sharp$ is unique with this property (the proof is analogous to that of Lemma 6.2). We may thus conclude:

**Lemma 6.6.** *$cSRng\langle M\rangle$ is the free c-semiring over the partial valuation structure $M$.* □

From the fact that the composition of two free constructions is a free construction itself [Sannella and Tarlecki, 2012, Ch. 3], we further know:

**Corollary 6.7.** *$cSRng\langle PVS\langle P\rangle\rangle$ is the free c-semiring over the partial order $P$.* □

Therefore, we abbreviate $cSRng\langle PVS\langle P\rangle\rangle$ as $cSRng\langle P\rangle$ and obtain a *generic* way to embed *any* partial order $P$ into a c-semiring in a canonical way. More explicitly, this c-semiring has finite sets of incomparable (w.r.t. the Smyth-ordering) multisets composed of elements from $|P|$ as its elements. If, e.g., $|P| = (\{\mathrm{I}, \mathrm{II}, 1, 2\})$, then $\{\wr \mathrm{I}, 2 \int\}$ or $\{\wr \mathrm{I}, \mathrm{I} \int, \wr 1, \mathrm{I} \int\}$ are in $\mathcal{I}_{\text{fin}}(\mathcal{M}_{\text{fin}}(P))$ but $\{\wr \mathrm{I}, \mathrm{I} \int, \wr \mathrm{II} \int\}$ is not since $\wr \mathrm{I}, \mathrm{I} \int \preceq_P \wr \mathrm{II} \int$ (cf. Figure 6.7 for a similar ordering). In the following section, we revisit this free c-semiring to illustrate the application of $\oplus$ and the required distributivity with respect to possible solving algorithms.

## 6.5 Adequacy of Algebraic Structures for Soft Constraints

The original goal of algebraic abstractions of specific soft constraint formalisms was to provide a common theoretical ground for questions of computational complexity and, perhaps more intensely studied, efficient solving algorithms. The latter include search strategies, dynamic programming techniques, and constraint propagation.

In terms of model expressiveness, we seek a fairly general structure that captures a broad variety of formalisms. In terms of algorithmic efficiency, however, we are inclined to sacrifice generality for some additional structure that makes search and propagation more effective. Most algorithmic efforts can roughly be divided into:

- Classical search algorithms such as branch-and-bound, limited discrepancy search, or large neighborhood search [Shaw, 1998] with accompanying search heuristics and efficient bounding techniques such as Russian doll search or mini bucket elimination [Meseguer et al., 2006].
- Soft local consistency and soft global constraints to enhance a search scheme [Cooper and Schiex, 2004; Cooper et al., 2010]
- Dynamic programming algorithms (variable elimination, bucket elimination, cluster tree elimination) [Bertele and Brioschi, 1973; Bistarelli, 2004; Dechter, 1999]

Originally, valued constraints and c-semiring-based soft constraints generalized weighted constraints and fuzzy constraints, respectively. While c-semirings additionally allowed for partiality to better represent incomparable decisions, valued constraints put a total ordering first instead of an operator for the supremum.[4] Totality is beneficial for solving as it reduces the search to better-known scalar optimization tasks with a unique optimal solution degree and allows for more efficient pruning. Soft local consistency techniques exploit the monotonicity of valuation structures. If the combination operator is not idempotent, these algorithms further require so-called “fair” valuation structures that admit a difference operator $a \ominus b$ – which is not mandatory for a PVS.

Similarly, the supremum $\oplus$ presupposed by a c-semiring is put to use in non-serial dynamic programming such as bucket elimination whenever we perform a “projection” operation. Projection means finding the best extension (with a greater variable scope) of a given assignment. If we are dealing with PVS without a supremum (such as the free PVS), these algorithms are not directly applicable. However, as a remedy, we can still use this family of algorithms if we put in place the free c-semiring instead. Example 6.2 demonstrates this procedure for bucket elimination. This algorithm proceeds by picking a variable elimination order, leading to “buckets” for each variable $x$ which collect all soft constraints $\mu$ that have $x$ as next (not

[4] Obviously, in a total ordering, the supremum is just min/max.

yet eliminated) variable in their scope. Intermediate soft constraints $\nu$ are generated by taking the union of all variables in a bucket, then calculating the intermediate results (i.e., the combination over all soft constraint valuations in the bucket) for each assignment in the Cartesian product of the domains, and projecting out $x$ (see [Dechter, 1999]). One can check that each elimination step is an application of the distributivity law (see Lemma 6.4). All known limitations regarding time and space which prohibit widespread usage in practice, of course, remain [Meseguer et al., 2006].

**Example 6.2 – Rating System on the free c-semiring**
Consider a decision that is made based on some abstract "rating system" $\mathsf{R}$ (as can be seen in Figure 6.7) that is inspired by, e.g., two executives that make an independent choice, denoted by $\{1, 2\}$ and $\{\mathrm{I}, \mathrm{II}\}$, respectively, where a higher number means a *better* evaluation. Any "two", however, is better than any "one". There is an explicit top element $\top$ representing maximal satisfaction. There is no unique least upper bound for 1 and I, though. We assume that soft constraints are specified by a map from variable assignments to elements of $|\mathsf{R}|$, as presented in the figure. To consider combinations of individual soft constraint valuations, i.e., to have a proper *SCSP*, we use the free partial valuation structure $PVS\langle\mathsf{R}\rangle$ to obtain a multiplication. We represent every element $r$ other than $\top$ as $\eta_{\mathsf{R}}^{\mathrm{PVS}}(r) = \wr r \int$ and let $\top$ map to the neutral element $\wr\int$. Note how the resulting partial order $PO(PVS\langle\mathsf{R}\rangle)$ over $\mathcal{M}_{\mathrm{fin}}(|\mathsf{R}|)$ is not suprema-closed (center in Figure 6.7). To still be able to apply bucket elimination, we embed $PVS\langle\mathsf{R}\rangle$ into its associated free c-semiring $cSRng\langle PVS\langle\mathsf{R}\rangle\rangle = cSRng\langle\mathsf{R}\rangle$. Consequently, we embed any soft constraint $\mu$ mapping to $|\mathsf{R}|$ into $cSRng\langle\mathsf{R}\rangle$ as follows:

$$\mu^{\mathsf{e}}(\theta) = \eta_{PVS\langle\mathsf{R}\rangle}^{\mathrm{cSRng}}(\eta_{\mathsf{R}}^{\mathrm{PVS}}(\mu(\theta)) = \begin{cases} \{\wr\int\} & \text{if } \mu(\theta) = \top \\ \{\wr\mu(\theta)\int\} & \text{if } \mu(\theta) \neq \top \end{cases} \tag{6.1}$$

For instance, $\mu_{\mathrm{x}}^{\mathsf{e}}(\{\mathrm{x} \mapsto 1\}) = \{\wr 2\int\}$ and $\mu_{\mathrm{xy}}^{\mathsf{e}}(\{\mathrm{x} \mapsto 0, \mathrm{y} \mapsto 0\}) = \{\wr\int\}$. Finally, we invoke bucket elimination to obtain the optimal solution degree (see [Dechter, 1999] for a similar illustration). The algorithm terminates with $\{\wr 1, 1, \mathrm{II}, \mathrm{II}\int, \wr \mathrm{I}, 2, 2, \mathrm{II}, \mathrm{II}\int, \wr 1, 2, 2, 2, \mathrm{II}\int\}$ that is clearly not reachable by any individual assignment. However, each of the three components (i.e., multiset over $|\mathsf{R}|$) corresponds to one assignment. By appropriate bookkeeping during the elimination process, we find that $\theta_1 = \{\mathrm{x} \mapsto 0, \mathrm{y} \mapsto 0, \mathrm{z} \mapsto 0\}$, $\theta_2 = \{\mathrm{x} \mapsto 0, \mathrm{y} \mapsto 1, \mathrm{z} \mapsto 0\}$, and $\theta_3 = \{\mathrm{x} \mapsto 1, \mathrm{y} \mapsto 1, \mathrm{z} \mapsto 0\}$ map to the respective optimal solution degrees and are thus optimal solutions. The free c-semiring provides enough information for said bookkeeping – another c-semiring returning a supremum of all solution degrees need not do this, in general.

Note that in fact, we get a set of all PVS-optima as the unique optimal solution degree in the free c-semiring. Clearly, however, enforcing totality or a supremum for the only sake of better algorithms might counteract a modeler's intentions. Some (in reality incomparable) solutions are dominated by others. If we do not rely on explicit soft constraint operations but rather formulate it as a conventional constraint optimization problem that is solved by search and propagation (as in branch-and-bound or large neighborhood search), *the structure a PVS offers suffices* – which makes them the appropriate data structure for designing MiniBrass.

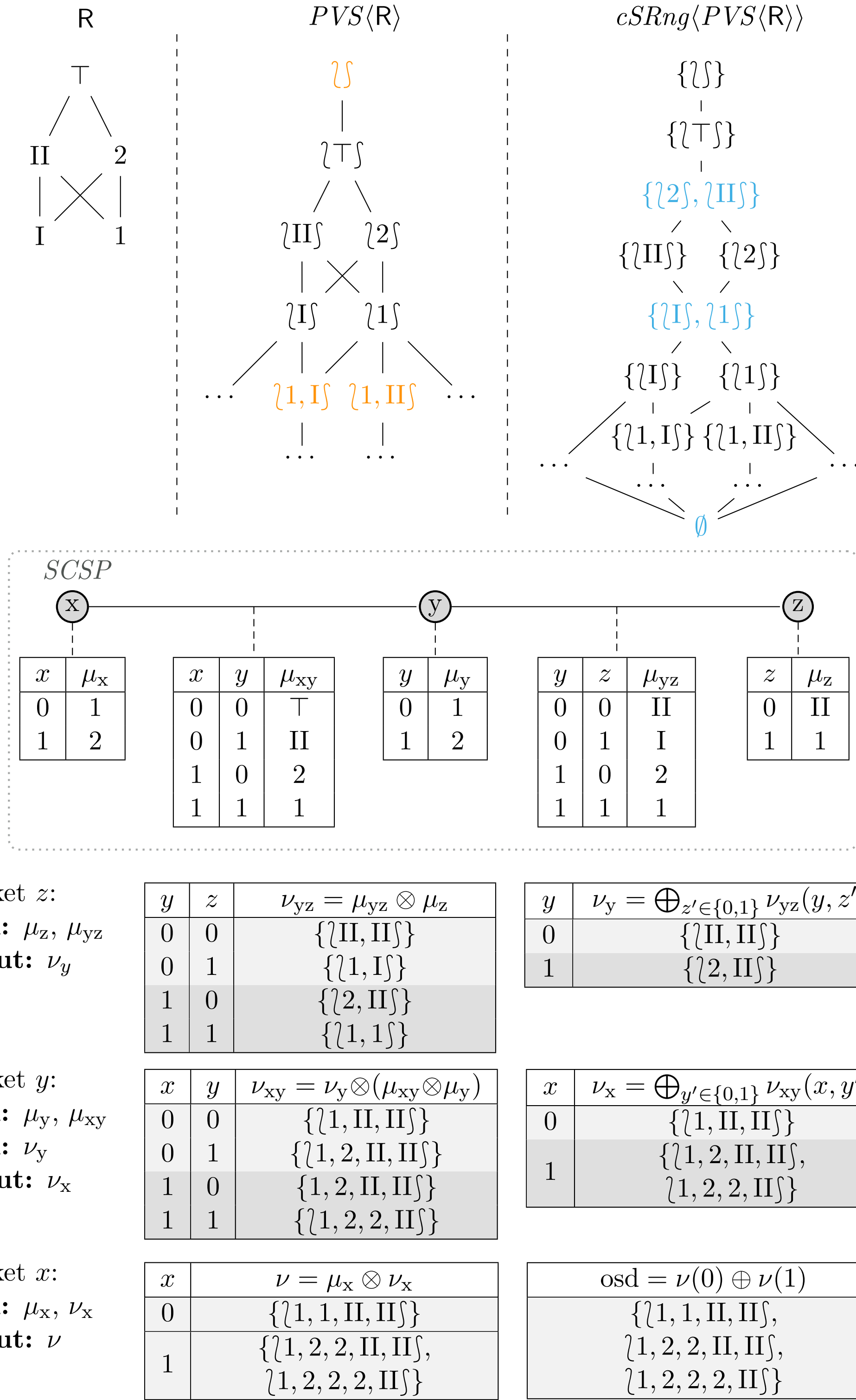



Figure 6.7: Upper: The rating system R, its free PVS, and its free c-semiring. Highlighted elements are introduced by the respective axioms. Center: A *SCSP* mapping to |R|. Lower: Finding the optimal solution degrees by bucket elimination on *cSRng*⟨R⟩.

**Chapter Summary and Outlook**

In this chapter, we discussed algebraic structures for soft constraints from first principles, i.e., without restricting ourselves to the existing literature. Partial valuation structures emerged as the least common denominator for a variety of formalisms, including the free PVS over a partial order that can be used to encode constraint preferences as PVS. Moreover, we also presented a free construction of a c-semiring which is based on sets of PVS-degrees. Chapter 7 offers foundations for hierarchical (i.e., lexicographic) combinations of PVS that are also present in MiniBrass.

Chapter **7**

# Hierarchically Layered Soft Constraints

**Summary.** Among all conceivable operations over ordering relations and PVS, lexicographic combinations (where one ordering is strictly more important than the other) play a key rôle – from both a theoretical and practical viewpoint. This chapter explores and completes Hosobe's previously undertaken approaches to formalizing constraint hierarchies as algebraic structures by providing a general treatment of lexicographic products of PVS. This encompasses the negative result that certain classes of hierarchical layers (including a "worst-case-better" semantics) originally proposed by Borning and left out by Hosobe lead to PVS prohibitive for lexicographic combinations. We propose the notion of optima-simulation to replace such a PVS by another one which generally leads to modified and refined soft constraint problems having a subset of optimal solutions. We show two concrete instances, based on $p$-norms and real-valued multisets, respectively.

**Publication.** Some of the concepts and results outlined in this chapter have been previously published in [Schiendorfer et al., 2015c; Knapp and Schiendorfer, 2014].

Following the previous chapters' discussions, it is straightforward to show that many specific formalisms in the literature are instances of partial valuation structures – from weighted to fuzzy constraints. However, there is one "stubborn" formalism, *constraint hierarchies*, that turned out to be reluctant to an encoding in algebraic structures. Even though Meseguer et al. [2006] note that the combinations of error functions on a single layer (e.g., sum of errors, sum of squares of errors, or maximum error) can be cast as valued constraint problems, "the general definition of combining functions does not forbid the use of functions that would definitely violate fundamental semiring or valued constraint axioms (such as monotonicity)." Even if we can encode a single layer as a valuation structure, the essence of constraint hierarchies lies in the lexicographic solution ordering according to the layers' satisfaction values. Hosobe [2009] was the first to provide a partial answer by encoding what he calls "rational constraint hierarchies", including weighted-sum and sum-of-squares comparators as c-semirings. In particular, "rational c-semirings exclude the worst-case-better (WCB) comparator" which – as we shall see – still has many practical applications.

Therefore, this chapter follows the important theoretical and practical question:

*When is the result of a lexicographic combination of several PVS itself a PVS?*

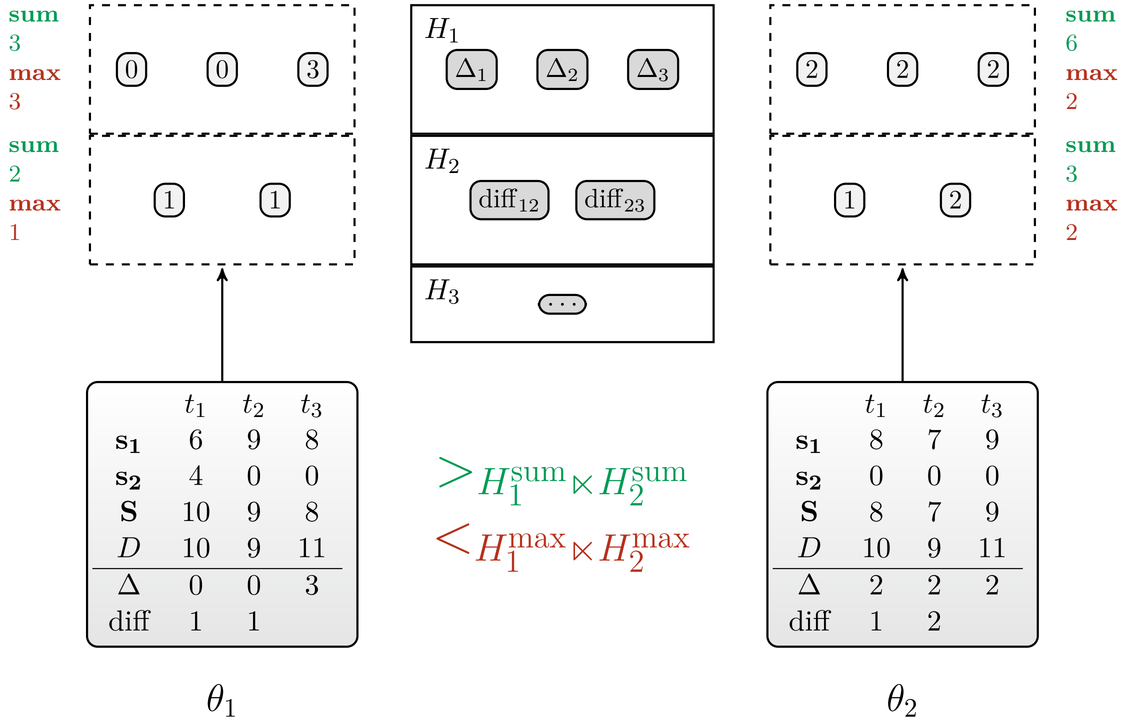



Figure 7.1: A hierarchically specified soft constraint problem with two candidate solutions $\theta_1$, $\theta_2$ and their valuations in PVS, as described in Example 7.1. $H_i^{\text{sum}}$ refers to the PVS $(\mathbb{N}_{\geq 0}, +, 0, \geq)$ and $H_i^{\text{max}}$ refers to $(\mathbb{N}_{\geq 0}, \max, 0, \geq)$ where $i \in \{1, 2\}$ indicates the layer from which soft constraints (e.g., $\Delta_1$) map into the PVS. Recall that we read, e.g., $x <_{H_1^{\text{sum}}} y$ as "$x$ is worse than $y$ in $H_1^{\text{sum}}$" which holds if $x > y$ since *violation* weights are accumulated.

More technically, we can apply the lexicographic operator introduced by Gadducci et al. [2013] (written as $\ltimes$) that takes two PVS and returns a PVS. For the example of constraint hierarchies, individual layers could then be mapped to, e.g., weighted PVS and the layers' PVS be combined by said operator. But such an operator has far more application scenarios than expressing constraint hierarchies: Combining fuzzy CSP with weighted, constraint preferences with probabilistic, etc. In fact, the lexicographic combination is one of the most mathematically sound [Andréka et al., 2002] and widely used aggregation strategies [Rentmeesters et al., 1996]. We have already introduced the lexicographic ordering for products of PVS in Section 3.2.3 and made "intuitive" use of it in MiniBrass in Section 4.4.1. Hence, we can simply attempt to apply this ordering to a product construction of two PVS $M$ and $N$, i.e., based on the carrier set $|M| \times |N|$. We briefly revisit the definition of the ordering:

$$(m, n) \leq_{M \ltimes N} (m', n') \leftrightarrow (m <_M m') \lor (m = m' \land n \leq_N n')$$

To demonstrate its application in analogy to Example 1.1, we propose a simple example that Figure 7.1 visualizes:

**Example 7.1 – Small Smart Grid**
Consider a trivially small soft constraint model inspired by our smart grid case study: Assume

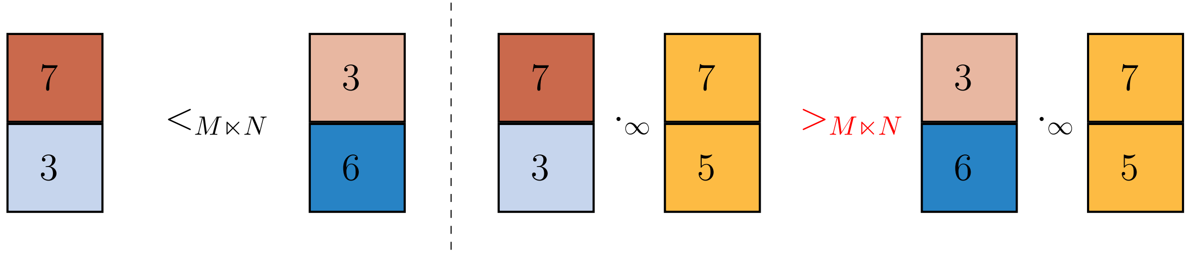



Figure 7.2: When considering two layers $M$ and $N$ in a lexicographic ordering, collapsing elements are invalidating the multiplication of partial valuation structures. Here, $M = N = \mathbb{N}^{\max} = (\mathbb{N}_{\geq 0}, \max, 0, \geq)$, $<_M$ refers to $>$ in the natural ordering, and we write $\cdot_\infty$ as a shorthand for "max". While thus $(7,3) <_{M \ltimes N} (3,6)$, on the right side $(7,5) >_{M \ltimes N} (7,6)$ upon multiplication with the same element $(7,5)$.

two power plants that can regulate their power supply ($s_1$ and $s_2$) between $\underline{S} = 0$ and $\overline{S} = 10$ (perhaps kWh) and assume that a certain demand $D$ is given for three time steps (say, hours). The most important *organizational* goal could be to schedule the supply such as to minimize its deviation to the demand. We model this as a weighted PVS $H_1$ with three soft constraints $\Delta_t$ for every time step that maps to the deviation at said time step. As a second layer that is again mapped to a weighted PVS $H_2$, assume that we prefer that the power plants do not change their output heavily – thus we have two soft constraints ("diff") for the transition from time step $t_1$ to $t_2$ and one for $t_2$ to $t_3$. Both PVS combine the soft constraints' valuations using either the maximum or sum operator as the binary multiplication. Figure 7.1 provides an example with two sample assignments $\theta_1$ and $\theta_2$. Depending on whether maximum or sum is used, either $\theta_1$ or $\theta_2$ is preferred.

So far, this seems to work out just fine, i.e., from a given assignment $\theta$ and PVS $M$ and $N$, we can have sets of soft constraints $C_s^M$ and $C_s^N$ map to $m \in |M|$ and $n \in |N|$, respectively, build the pairs $(m, n)$ and order the assignments lexicographically by $\leq_{M \ltimes N}$. But this is obviously an optimization problem different from our "canonical" soft constraint problem that maps to a single PVS and makes use of its combination operator. To be able to treat lexicographic combinations within the unifying framework of PVS, we need to make sure that $M \ltimes N$ itself forms a PVS – respecting all required axioms.

But what happens if we simply take pairs of $|M|$ and $|N|$ as elements and try to combine them using component-wise application of $\cdot_M$ and $\cdot_N$? An essential property of any PVS $M$ is that $\cdot_M$ must be monotonic w.r.t. $\leq_M$. Consequently the same must be true for $\cdot_{M \ltimes N}$ and $\leq_{M \ltimes N}$. If we have $(m, n) \leq_{M \ltimes N} (m', n')$ then multiplying both sides by the same object $(o_1, o_2)$ must not change the ordering. Unfortunately, Figure 7.2 provides a counterexample that invalidates this immediate approach. The problem is evident in the definition of the lexicographic ordering and the (weak) monotonicity requirement of PVS: If $m < m'$ then for any $n$ and $n'$, $(m, n) <_{M \ltimes N} (m', n')$. But on the other hand monotonicity only ensures that $m \cdot_M o \leq_M m' \cdot_M o$ for some $o \in |M|$, in particular $m \cdot_M o = m' \cdot_M o$ is possible and acceptable. Then the ordering entirely depends on the ordering in the second component, i.e., if $n \cdot_N o_2 >_N n' \cdot_N o_2$ holds in our previous example we would get $(m \cdot_M o_1, n \cdot_N o_2) >_{M \ltimes N}$

Table 7.1: Approximating the collapsing PVS $\mathbb{N}^{\max}$ by other non-collapsing PVS using $p$-norms. The table shows three assignments $\theta_i$ that are graded by soft constraints $\mu_j$. By $\cdot_\infty$, $\theta_1$ and $\theta_3$ are equal and strictly better than $\theta_2$. For $p = 2$, the ordering between $\theta_1$ and $\theta_2$ after evaluating $\cdot_2$ is wrong (indicated in red), whereas for $p > 2$, it is correct (indicated in green). Also, the embeddings into $\mathcal{M}_{\text{fin}}(\mathbb{R}_{>0})$ (introduced in Section 7.4.4) for $\overline{\mathrm{R}}$ are given.

| | $\mu_1$ | $\mu_2$ | $\mu_3$ | max | $\prod_2$ | $\prod_3$ | $\prod_4$ | $\prod_{\overline{\mathrm{R}}}$ |
|---|---|---|---|---|---|---|---|---|
| $\theta_1$ | 0 | 0 | 3 | **3** | 3 | 3 | 3 | $\Lbag 3 \Rbag$ |
| $\theta_2$ | 2 | 2 | 2 | **2** | 3.46 | 2.88 | 2.63 | $\Lbag 2, 2, 2 \Rbag$ |
| $\theta_3$ | 0 | 2 | 3 | **3** | 3.6 | 3.27 | 3.13 | $\Lbag 2, 3 \Rbag$ |

$(m' \cdot_M o_1, n' \cdot_N o_2)$, contradicting the required monotonicity. One might be inclined to relax the monotonicity requirement for arbitrary PVS altogether but this leads to a paradoxical situation following the example in Figure 7.2: Consider a PVS $M$ where we might have two assignments $\theta$ and $\theta'$, each deemed equal by the product of $n$ soft constraints' valuations, i.e., $m = \mu_1(\theta) \cdot_M \ldots \cdot_M \mu_n(\theta) = \mu_1(\theta') \cdot_M \ldots \cdot_M \mu_n(\theta')$. Now a new soft constraint $\mu_{n+1}$ is added with $\mu_{n+1}(\theta) <_M \mu_{n+1}(\theta')$, i.e., $\theta$ is considered strictly *better* by $\mu_{n+1}$. However, it could be that $m \cdot_M \mu_{n+1}(\theta) >_M m \cdot_M \mu_{n+1}(\theta')$, i.e., in total, $\theta$ would now be *worse* than $\theta'$.

Gadducci et al. [2013] first identified this problem with lexicographic combinations and introduced the concept of *collapsing elements* to make it formally precise. Intuitively, elements are collapsing precisely if they equalize incomparable elements upon multiplication, as seen with the max operator in Figure 7.2. In Section 7.1.1, we see that every idempotent element (i.e., $m \cdot_M m = m$) is collapsing – this clearly includes every element and the max operator. Collapsing elements are prohibitive for lexicographic products and thus need to be removed from the carrier set.

This result of Gadducci et al. [2013] provides insight into why Hosobe [2009] failed to provide a c-semiring construction (that is even more specialized than that of a PVS) for "worst-case-better". That comparator simply is the maximum operator applied to error functions and thus has every element as collapsing. While this is primarily a negative result, it is the purpose of this chapter and [Schiendorfer et al., 2015c] to formally define and prove how we can reasonably "replace" one PVS containing collapsing elements by another one that does not. Clearly, those PVS cannot be equivalent in terms of optima. But as long as some properties of the search space ordering are preserved (in particular, the optimal solutions of the replaced problem are a subset of the originally optimal solutions), we argue that modelers would not mind using a non-collapsing replacement to remain in the scope of PVS-based soft constraint problems. We call this criterion "optima-simulation" and explain it in more detail in Section 7.4.2.

We need two vital insights to find a suitable replacement for the collapsing maximum-based PVS. First, we may note that the max operator corresponds to the $\infty$-norm which is the limit of $p$-norms, taking $p$ to infinity. Formally, for real numbers $r$ and $s$, $\lim_{p\to\infty}(r^p + s^p)^{1/p} = \max\{r, s\}$. Second, any *finite* $p$-norm, by contrast, is free of collapsing elements since $(r^p + s^p)^{1/p}$ is strictly monotonic in both arguments. Our intuition is that if we pick $p$ large enough, the $p$-norm approximates the maximum operator for the relevant part of the search space. Table 7.1 exemplifies this idea: for $p = 2$, the vector $[2, 2, 2]$ has higher costs than $[0, 0, 3]$ whereas for $p \geq 3$, the ordering relation is equivalent to what max would pick. Section 7.3 presents "actionable" advice how to reasonably select $p$. If we are not restricted to a PVS with a scalar carrier set, Section 7.4.4 offers an alternative collapse-free PVS capable of replacing the

maximum PVS based on finite multisets over reals that is more general than the purely scalar $p$-norm variant.

As a case study throughout this chapter, we consider the PVS that combines violation degrees by the maximum operator, as used in Figure 7.1. Note that once we have constructed a lexicographic product of PVS, we can always embed the resulting PVS in a c-semiring using the free construction presented in Section 6.4. It is however not possible to define a lexicographic product for c-semirings [Hölzl et al., 2009].

## 7.1 Towards Lexicographic Products of PVS

To formalize the aforementioned intuitions, we first closely follow the exposition of Gadducci et al. [2013] to treat lexicographic products of PVS in this section.

### 7.1.1 Collapsing Elements as an Obstacle

For a PVS $M = (|M|, \cdot_M, \varepsilon_M, \leq_M)$, define its set of *collapsing elements* by

$$\mathcal{C}(M) = \{m \in |M| \mid \exists m_1, m_2 \in |M| \,.\, m_1 <_M m_2 \wedge m_1 \cdot_M m = m_2 \cdot_M m\} \ .$$

We abbreviate the regular, non-collapsing elements $|M| \setminus \mathcal{C}(M)$ as $\mathcal{R}(M)$ and obviously have $|M| = \mathcal{C}(M) \cup \mathcal{R}(M)$.

**Example 7.2 – Collapsing Elements**
Let $P$ be a partial order.

(1) The set of collapsing elements of the free PVS $\mathit{PVS}\langle P\rangle = (\mathcal{M}_{\text{fin}}\, |P|, \uplus, \wr\int, \Subset^P)$ is empty: If $T \Subset^P U$, then $T \uplus \wr p\int = U \uplus \wr p\int$ for some $p \in |P|$ would imply that $T = U$.

(2) Consider the PVS of natural numbers that are combined by the maximum operation, i.e., $m \cdot_M n = \max\{m, n\}$ leads to the the PVS $\mathbb{N}^{\max} = (\mathbb{N}_{\geq 0}, \max, \geq, 0)$ where $\geq$ is the usual ordering. The set of collapsing elements of $\mathbb{N}^{\max}$ is $\mathbb{N} \setminus \{0\}$: Let $n > 0 \in \mathbb{N}$. Then $n > 0$ but $\max\{n, 0\} = n = \max\{n, n\}$. The situation is analogous in fuzzy PVS (see Section 4.3.3).

Generalizing the first example, if $M$ is a *strict* PVS, i.e., $m <_M n$ implies $m \cdot_M o <_M n \cdot_M o$ for all $m, n, o \in |M|$, then $\mathcal{C}(M) = \emptyset$. Generalizing the second example, all idempotent elements of a PVS $M$ which are different from $\varepsilon_M$ are collapsing: If $m \in |M|$ such that $m \neq \varepsilon_M$ and $m \cdot_M m = m$, then $m <_M \varepsilon_M$ but $m \cdot_M m = m = \varepsilon_M \cdot_M m$. It is well-known that strictness and idempotency are mutually exclusive in PVS if $|M| > 2$ [Schiex et al., 1995]: If $m \neq \varepsilon_M$ then $m <_M \varepsilon_M$ and, by strict monotonicity, also $m \cdot_M m <_M m$ but idempotency would require $m \cdot_M m = m$.

Collapsing elements are problematic for lexicographic products, as we have seen in Figure 7.2. On the other hand, all *non-collapsing* elements form a sub-structure of a PVS that is again a PVS. To work towards this result, we first show that the multiplication operation is closed under all non-collapsing elements. Formally:

**Lemma 7.1.** *$|M| \setminus \mathcal{C}(M)$ is closed under $\cdot_M$.*

*Proof.* Saying that $|M| \setminus \mathcal{C}(M)$ is closed under $\cdot_M$ is equivalent to stating that if, and only if, $m$ or $n$ are collapsing, their product $m \cdot_M n$ is collapsing.

If $m \in \mathcal{C}(M)$ then there are $m_1, m_2 \in |M|$ such that $m_1 <_M m_2$ but $m_1 \cdot_M m = m_2 \cdot_M m$ which clearly implies that $m \cdot_M n$ is also collapsing since $m_1 <_M m_2$ but $(m_1 \cdot_M m) \cdot_M n = (m_2 \cdot_M m) \cdot_M n$ – and analogously if $n \in \mathcal{C}(M)$.

Conversely, if $m \cdot_M n \in \mathcal{C}(M)$, then there are $m_1, m_2 \in |M|$ with $m_1 <_M m_2$, and $m_1 \cdot_M (m \cdot_M n) = m_2 \cdot_M (m \cdot_M n)$. For a contradiction, assume that neither $m$ nor $n$ are collapsing. Then $m_1 <_M m_2$ implies $m_1 \cdot_M m <_M m_2 \cdot_M m$ since $m \notin \mathcal{C}(M)$, and consequently $(m_1 \cdot_M m) \cdot_M n <_M (m_2 \cdot_M m) \cdot_M n$ since $n \notin \mathcal{C}(M)$ – contradicting our assumption of $m \cdot_M n$ being collapsing for $m_1$ and $m_2$. □

In particular, the neutral element $\varepsilon_M$ is never collapsing since for any $m_1 <_M m_2$ it must be that $m_1 \cdot_M \varepsilon_M <_M m_2 \cdot_M \varepsilon_M$. Thus, $(|M| \setminus \mathcal{C}(M), \cdot_M, \varepsilon_M, \leq_M)$ forms a PVS since all required PVS-properties are inherited from $M$.

In a bounded PVS $M$, the smallest element $\bot_M$ must also be an *absorbing* element for $\cdot_M$, i.e., it holds that $m \cdot_M \bot_M = \bot_M$ for all $m \in |M|$: Since $m \leq_M \varepsilon_M$, by monotonicity, we get $\bot_M \cdot_M m \leq_M \bot_M$ and thus, by antisymmetry of $\leq_M$, we have $\bot_M \cdot_M m = \bot_M$. Furthermore, $\bot_M \in \mathcal{C}(M)$ if $\bot_M \neq \varepsilon_M$ since then $\bot_M <_M \varepsilon_M$ but $\bot_M \cdot_M \bot_M = \varepsilon_M \cdot_M \bot_M = \bot_M$.

### 7.1.2 The Lexicographic Product excludes Collapsing Elements

Let $M$ be a PVS and let $N$ be a bounded PVS (augmented, if it was not already bounded, as mentioned in Section 3.2.3). As a consequence of the existence of collapsing elements, we handle these elements differently to obtain a meaningful carrier set for the lexicographic product. We define $L \subseteq |M| \times |N|$ as

$$L = ((|M| \setminus \mathcal{C}(M)) \times |N|) \cup (\mathcal{C}(M) \times \{\bot_N\}) ,$$

i.e., the subset of pairs $(m, n) \in |M| \times |N|$ such that if $m \in \mathcal{C}(M)$, then $n = \bot_N$. Define the binary operation $\cdot_L : L \times L \to L$ component-wise (analogously to the direct product) by

$$(m_1, n_1) \cdot_L (m_2, n_2) = (m_1 \cdot_M m_2, n_1 \cdot_N n_2) .$$

This is well-defined, i.e., for all $(m_1, n_1), (m_2, n_2) \in L$, $(m_1 \cdot_M m_2, n_1 \cdot_N n_2) \in L$: If neither $m_1$ nor $m_2$ are collapsing, neither is $m_1 \cdot_M m_2$ as a corollary to Lemma 7.1. Hence $m_1 \cdot_M m_2 \in |M| \setminus \mathcal{C}(M)$ and $n_1 \cdot_N n_2 \in |N|$, thus $(m_1 \cdot_M m_2, n_1 \cdot_N n_2) \in L$.

If either of $m_1$ and $m_2$ are collapsing, say $m_1$ is, then $m_1 \cdot_M m_2 \in \mathcal{C}(M)$ by Lemma 7.1. Since $n_1$ then must be $\bot_N$ due to the definition of $L$, $n_1 \cdot_N n_2 = \bot_N$ due to $\bot_N$ being the absorbing element for $\cdot_N$. Hence, again, $(m_1 \cdot_M m_2, n_1 \cdot_N n_2) = (m_1 \cdot_M m_2, \bot_N) \in L$. The case for $m_2 \in \mathcal{C}(M)$ is symmetric.

The binary operation $\cdot_L$ inherits associativity and commutativity from $M$ and $N$. Further, to get a neutral element for $\cdot_L$, we define $\varepsilon_L \in L$ as $\varepsilon_L = (\varepsilon_M, \varepsilon_N)$ which is indeed in $L$ since $\varepsilon_M \notin \mathcal{C}(M)$. Also $(m, n) \cdot_L \varepsilon_L = (m, n)$. As we did before, we define the *lexicographic ordering* $\leq_L \subseteq L \times L$ on $L$ by

$$(m_1, n_1) \leq_L (m_2, n_2) \iff (m_1 <_M m_2) \text{ or } (m_1 = m_2 \text{ and } n_1 \leq_N n_2) ,$$

and conclude:

**Lemma 7.2.** $(L, \cdot_L, \varepsilon_L, \leq_L)$ *is a PVS.*

Since $(L, \cdot_L, \varepsilon_L)$ is a commutative monoid by the remarks above, it only remains to show the monotonicity of $\cdot_L$ w.r.t. $\leq_L$. That is, that $(m_1, n_1) \leq_L (m_2, n_2)$ implies $(m_1, n_1) \cdot_L (m', n') \leq_L (m_2, n_2) \cdot_L (m', n')$ for all $(m', n') \in L$. The crucial insight is to note that $m'$ might be collapsing for $m_1$ and $m_2$ which leaves nothing to then force order-preservation in the second component, as Figure 7.2 illustrated.

Hence, if $m'$ were indeed collapsing, it can only be paired in $L$ with $n' = \bot_N$. Then even though $m_1 \cdot_M m' = m_2 \cdot_M m'$, we would map both $n_1 \cdot_N n'$ and $n_2 \cdot_N n'$ to $\bot_N$ and assert that $(m_1, n_1) \cdot_L (m', n') \leq_L (m_2, n_2) \cdot_L (m, n)$ by equality in both components.

*Proof.* To prove monotonicity formally, let $(m_1, n_1) \leq_L (m_2, n_2)$ and an $(m', n') \in L$ be given. Case $m_1 <_M m_2$: If $m_1 \cdot_M m' <_M m_2 \cdot_M m'$ then obviously $(m_1 \cdot_M m', n_1 \cdot_M n') <_L (m_2 \cdot_M m', n_2 \cdot_M n')$. If otherwise $m_1 \cdot_M m' = m_2 \cdot_M m'$ holds, then $m' \in \mathcal{C}(M)$ and thus $n' = \bot_N$. Hence

$$
\begin{aligned}
(m_1, n_1) \cdot_L (m', n') = (m_1 \cdot_M m', n_1 \cdot_N n') = (m_1 \cdot_M m', \bot_N) = \\
(m_2 \cdot_M m', \bot_N) = (m_2 \cdot_M m', n_2 \cdot_N n') = (m_2, n_2) \cdot_L (m', n') \ .
\end{aligned}
$$

Case $m_1 = m_2$ and $n_1 \leq_N n_2$: Then $m_1 \cdot_M m' = m_2 \cdot_M m'$ and $n_1 \cdot_N n' \leq_N n_2 \cdot_N n'$ by monotonicity which implies $(m_1, n_1) \cdot_L (m', n') \leq_L (m_2, n_2) \cdot_L (m', n')$. □

We use the operator $\ltimes$ for lexicographic products, i.e., we write $M \ltimes N$ for $(L, \cdot_L, \varepsilon_L, \leq_L)$. If $M$ is also bounded, then $M \ltimes N$ is bounded with $\bot_{M \ltimes N} = (\bot_M, \bot_N)$.

An important practical consequence of the definition of $\ltimes$ is that the first parameter PVS $M$ better shows no collapsing elements if we want a lossless encoding of a lexicographic relationship between two PVS. From a soft constraint perspective, no soft constraint is allowed to map an assignment to any collapsing element. This restriction, however, renders PVS such as, e.g., $\mathbb{N}^{\max}$ practically useless since every element other than $\varepsilon_{\mathbb{N}^{\max}} = 0$ is collapsing and would thus be removed for $\ltimes$.

Since in general, hierarchically structured soft constraint problems may form deeper hierarchies than just two layers, associativity of $\ltimes$ is a property of interest, i.e., that $M \ltimes (N \ltimes O) = (M \ltimes N) \ltimes O$. Our proof is supported by describing the collapsing and non-collapsing elements of a product $M \ltimes N$ in terms of the collapsing and non-collapsing elements of $M$ and $N$.

**Lemma 7.3.** *Let $M$ and $N$ be PVS such that $N$ is bounded. The set of collapsing elements for the lexicographic product of $M$ and $N$ is composed of all collapsing elements of $M$ paired with $\bot_N$ and every non-collapsing element of $M$ paired with any collapsing element of $N$:*

$$
\mathcal{C}(M \ltimes N) = (\mathcal{C}(M) \times \{\bot_N\}) \cup (\mathcal{R}(M) \times \mathcal{C}(N)) \ .
$$

*Proof.* Indeed, let $(m, n) \in \mathcal{C}(M \ltimes N)$. Then there are $(m_1, n_1), (m_2, n_2) \in |M \ltimes N|$ with $(m_1, n_1) <_L (m_2, n_2)$, i.e., either we have $m_1 <_M m_2$ or it is the case that $m_1 = m_2$ and $n_1 <_N n_2$ holds, but $(m_1, n_1) \cdot_{M \ltimes N} (m, n) = (m_2, n_2) \cdot_{M \ltimes N} (m, n)$ and thus $m_1 \cdot_M m = m_2 \cdot_M m$ and $n_1 \cdot_N n = n_2 \cdot_N n$.

In the case of $m_1 <_M m_2$, $m$ is collapsing in $M$ and therefore $n$ must be equal to $\bot_N$ since $(m, n) \in |M \ltimes N|$ – consequently $(m, n) \in \mathcal{C}(M) \times \{\bot_N\}$. If on the other hand $m_1 = m_2$, then $n$ must be in $\mathcal{C}(N)$ since $n_1 <_N n_2$ but $n_1 \cdot_N n = n_2 \cdot_N n$. Thus, $(m, n) \in \mathcal{R}(M) \times \mathcal{C}(N)$.

Conversely, first assume $(m, n) \in \mathcal{C}(M) \times \{\bot_N\}$. Then there are $m_1 <_M m_2 \in |M|$ with $m_1 \cdot_M m = m_2 \cdot_M m$. With these elements we can construct $(m_1, \bot_N)$ and $(m_2, \bot_N)$ where

$(m_1, \bot_N) <_{M \ltimes N} (m_2, \bot_N)$ but multiplying with $(m, n)$ yields $(m \cdot_M m_1, \bot_N) = (m \cdot_M m_2, \bot_N)$ – hence $(m, n)$ is collapsing in $M \ltimes N$. Now, for the second case, let $(m, n) \in (|M| \setminus \mathcal{C}(M)) \times \mathcal{C}(N)$. Then there are $n_1 < n_2 \in |N|$ with $n_1 \cdot_N n = n_2 \cdot_N n$. But then also $(m, n_1) <_{M \ltimes N} (m, n_2)$ and $(m, n_1) \cdot_{M \ltimes N} (m, n) = (m \cdot_M m, n_1 \cdot_N n) = (m, n_2) \cdot_{M \ltimes N} (m, n)$ which makes $(m, n)$ collapsing in $M \ltimes N$. □

Next, we characterize the non-collapsing elements of a lexicographic product:

$$\begin{aligned}
&\mathcal{R}(M \ltimes N) = |M \ltimes N| \setminus \mathcal{C}(M \ltimes N) = \\
&\quad \Big( [\,\mathcal{C}(M) \times \{\bot_N\}] \cup [\,\mathcal{R}(M) \times |N|\,] \Big) \setminus \Big( [\,\mathcal{C}(M) \times \{\bot_N\}] \cup [\,\mathcal{R}(M) \times \mathcal{C}(N)] \Big) = \\
&\quad \mathcal{R}(M) \times (|N| \setminus \mathcal{C}(N)) = \mathcal{R}(M) \times \mathcal{R}(N) \ .
\end{aligned}$$

Applying both of the decompositions for $\mathcal{C}(M \ltimes N)$ and $\mathcal{R}(M \ltimes N)$, we examine associativity for three PVS $M$, $N$, and $O$, where $N$ and $O$ are bounded:

$$\begin{aligned}
&|(M \ltimes N) \ltimes O| = \mathcal{R}((M \ltimes N) \ltimes O) \cup \mathcal{C}((M \ltimes N) \ltimes O) = \\
&\quad [\,\mathcal{R}(M \ltimes N) \times \mathcal{R}(O)] \cup [\,\mathcal{C}(M \ltimes N) \times \{\bot_O\}] \cup [\,\mathcal{R}(M \ltimes N) \times |O|\,] = \\
&\quad [\,\mathcal{R}(M \ltimes N) \times |O|\,] \cup [\,\mathcal{C}(M \ltimes N) \times \{\bot_O\}] = \\
&\quad [\,\mathcal{R}(M) \times \mathcal{R}(N) \times |O|\,] \cup [\,(\mathcal{C}(M) \times \{\bot_N\} \cup \mathcal{R}(M) \times \mathcal{C}(N)) \times \{\bot_O\}\,] = \\
&\quad [\,\mathcal{R}(M) \times \mathcal{R}(N) \times |O|\,] \cup [\,\mathcal{R}(M) \times \mathcal{C}(N) \times \{\bot_O\}\,] \cup [\,\mathcal{C}(M) \times \{(\bot_N\} \times \{\bot_O)\}\,] = \\
&\quad \mathcal{R}(M) \times [\,(\mathcal{R}(N) \times |O|) \cup (\mathcal{C}(N) \times \{\bot_O\})\,] \cup [\,\mathcal{C}(M) \times \{\bot_{N \ltimes O}\}\,] = \\
&\quad [\,\mathcal{R}(M) \times |N \ltimes O|\,] \cup [\,\mathcal{C}(M) \times \{\bot_{N \ltimes O}\}\,] = \\
&\quad |M \ltimes (N \ltimes O)| \ ,
\end{aligned}$$

from which it follows that $\ltimes$ is associative since the binary operation $\cdot_{M \ltimes N \ltimes O}$ is associative and the ordering $\leq_{M \ltimes (N \ltimes O)}$ is equivalent to $\leq_{(M \ltimes N) \ltimes O}$. Concluding, there exists an associative lexicographic product of PVS that results in a PVS itself. The only caveat is that collapsing elements must necessarily be eliminated and receive special treatment in picking the carrier set of the product. This answers this chapter's introductory question "When is the result of a lexicographic combination of several PVS itself a PVS?": Precisely if none but the last layer contain collapsing elements. Otherwise, we may get a severely truncated version of the lexicographic product we aimed for.

## 7.2 Constraint Hierarchies as Products of PVS

Equipped with a working lexicographic product of PVS, we revisit *constraint hierarchies* [Borning et al., 1992] to investigate what kind of comparators layers need to use in order to map to a PVS suitable for lexicographic products, i.e., one without collapsing elements. Consider Figure 7.1 as an example although originally constraint hierarchies considered boolean constraints that were mapped to the reals with (weighted) error functions.

Thus, an $n$-layered constraint hierarchy $C^H = (C^{(k)})_{1 \leq k \leq n}$ over variables $X$ and domains $D$, or $(X, D)$-*constraint hierarchy*, is given by a family of $n$ sets of constraints. The constraints in *level* $1 \leq k \leq n$ are considered as *strictly more important* than the constraints in level $k + 1$. A constraint hierarchy is *finite* if $\bigcup_{1 \leq k \leq n} C^{(k)}$ is finite.

To map a finite constraint hierarchy $C_H = (C^{(k)})_{1 \leq k \leq n}$ to PVS-based soft constraint problems, let $L = (H_k)_{1 \leq k \leq n}$ be a corresponding family of partial valuation structures that serve as

target structures to be mapped from assignments. Furthermore, every constraint $c \in C^{(k)}$ on layer $k$ is associated to an $H_k$-soft constraint $\eta_c$ which is in charge of interpreting $c$ with respect to $H_k$. For instance, the boolean soft constraint "$x \leq 5$" could likewise be interpreted as a boolean $H_k$-soft constraint if $|H_k| = \{\text{true}, \text{false}\}$ or it could be interpreted as $\eta_c(\theta) = |\theta(x) - 5|$ to obtain an error metric if $|H_k| \subseteq \mathbb{N}$. We call $C_{\mathrm{s}}^H = (C_{\mathrm{s}}^{(k)})_{1 \leq k \leq n}$ with $C_{\mathrm{s}}^{(k)} = \{\eta_c \mid c \in C^{(k)}\}$ for $1 \leq k \leq n$ a *soft* constraint hierarchy.

For a $\theta \in [X \to D]$ the *hierarchy solution degree* for $C_{\mathrm{s}}^H$ of $\theta$ is defined to be the vector $(C_{\mathrm{s}}^{(k)}(\theta))_{1 \leq k \leq n}$. We can use the hierarchical solution degrees to induce a binary relation $<_H \subseteq [X \to D] \times [X \to D]$ over assignments by

$$\theta <_H \theta' \iff \exists 1 \leq k \leq n \,.\, (\forall 1 \leq i \leq k-1 \,.\, C_{\mathrm{s}}^{(i)}(\theta) = C_{\mathrm{s}}^{(i)}(\theta')) \wedge C_{\mathrm{s}}^{(k)}(\theta) <_{H_k} C_{\mathrm{s}}^{(k)}(\theta') \;,$$

saying that the valuation $\theta'$ is *strictly better* than the valuation $\theta$, and denote its reflexive closure on $[X \to D]$ by $\leq_H$, which is precisely the lexicographic order on the set $\{(C_{\mathrm{s}}^{(k)}(\theta))_{1 \leq k \leq n} \mid \theta \in [X \to D]\}$. To express the same ordering with a lexicographic product of PVS, we have that in particular,

$$\theta <_H \theta' \iff (C_{\mathrm{s}}^{(k)}(\theta))_{1 \leq k \leq n} <_{H_1 \ltimes \ldots \ltimes H_n} (C_{\mathrm{s}}^{(k)}(\theta'))_{1 \leq k \leq n}$$

if, on the one hand, every $H_k$ is a bounded PVS for at least all $2 \leq k \leq n$, and, on the other hand, $C_{\mathrm{s}}^{(k)}(\theta), C_{\mathrm{s}}^{(k)}(\theta') \notin \mathcal{C}(H_k)$ for all $1 \leq k \leq n-1$, or, equivalently, if $\eta_c(\theta), \eta_c(\theta') \notin \mathcal{C}(H_k)$ for each $c \in C^{(k)}$, $1 \leq k \leq n-1$. The first requirement, that each $H_k$ is bounded, can be achieved by moving from $H_k$ to its lifted variant $(H_k)_\perp$. The other requirement, collapse-free PVS, does not give rise to a canonical solution but rather requires appropriate modeling endeavors.

### 7.2.1 Locally Predicate Better

Borning et al. [1992] referred to $<_H$ simply as the "better" predicate which is configurable by various comparator functions that are responsible for aggregating values on a single layer. Their most elementary suggestion, called *locally-predicate-better*, consists of interpreting every constraint as a boolean predicate and ordering individual layers by the strict inclusion of violation sets. An assignment is considered strictly worse if it violates a strict superset of another assignment's violation set (equivalently, if it satisfies a strict subset). On the one hand, this predicate has been shown to be expressible by constraint preferences in Section 5.3.1. On the other hand, we can use the local predicate viewpoint to exemplify the previously introduced construction of a soft constraint hierarchy $C_{\mathrm{s}}^H$ corresponding to a constraint hierarchy $C^H$.

Formally, let $C$ be a finite set of constraints. The *locally-predicate-better* (*LPB*) *level comparator* for $C$ corresponds to requiring

$$\theta <_C^{\mathrm{LPB}} \theta' \iff \{c \in C \mid \theta' \not\models c\} \subset \{c \in C \mid \theta \not\models c\} \;.$$

This can be expressed by choosing the PVS $M = (2^C, \cup, \emptyset, \supseteq)$ and the set of $M$-soft constraints $C_{\mathrm{s}}^M = \{\eta_c \mid c \in C\}$ with $\eta_c(\theta) = \{c\}$ if $\theta \not\models c$ and $\eta_c(\theta) = \emptyset$ otherwise, for each $c \in C$. However, all elements of $M$ are idempotent, and thus the collapsing elements of $M$ are $2^C \setminus \{\emptyset\}$. Hence, $M$ is not suitable for a lexicographic product.

This situation is rather obvious to solve. Choosing instead the PVS $N = (\mathcal{M}_{\mathrm{fin}}(C), \uplus, \emptyset, \supseteq)$ which has no collapsing elements and the set of $N$-soft constraints $C_{\mathrm{s}}^N = \{\nu_c \mid c \in C\}$ with

$\nu_c(\theta) = \wr c \int$ if $\theta \not\models c$ and $\nu_c(\theta) = \wr\int$ otherwise, for each $c \in C$, deviates this situation, since

$$C_s^M(\theta) \leq_M C_s^M(\theta') \iff C_s^N(\theta) \leq_N C_s^N(\theta')$$

holds for all $\theta, \theta' \in [X \to D]$. In terms of optima on a given set of assignments, $C_s^M$ and $C_s^N$ are thus equivalent (cf. Section 7.4.2).

### 7.2.2 Globally Better – Real-valued PVS

Locally predicate better turned out to be rather restrictive and indecisive (due to the generally large number of incomparable solution degrees). In addition to it, Borning et al. [1992] proposed *global* comparators that take as input numeric values obtained from an assignment for a single layer and aggregate them to one scalar (cf. Figure 7.1). Due to all proposed comparators being real-valued, we focus on *real* PVS in this section as the target PVS for the soft constraint hierarchies to construct. A real PVS $R$ has $0 \in |R| \subseteq \mathbb{R}_{\geq 0}$ as its underlying set, 0 as its neutral element, and the (inverted) usual ordering on the real numbers $\geq$ as its ordering. Choosing $\cdot_R$ introduces variety in real PVS.

The choices proposed by Borning et al. [1992] align with common error measures such as the sum of errors, the sum of squared errors (to punish stronger violations superlinearly), and the maximum of all errors. The first two metrics are instances of $p$-norms. More specifically, H. F. Bohnenblust axiomatically characterized *binary* operations that give rise to a norm [Bohnenblust, 1940] where one theorem is of particular interest for the task at hand: If homogeneity is to be satisfied, i.e., $(t \cdot r) \cdot_R (t \cdot s) = t \cdot (r \cdot_R s)$ holds in the real meet monoid $R$ for all $r, s, t \in \mathbb{R}_{\geq 0}$ (where $\cdot$ is the usual multiplication and $\cdot_R$ is the binary operation in question) then there are only two possible choices for $\cdot_R$:

a) $1 \cdot_R 1 = 1$ and $r \cdot_R s = \max\{r, s\}$ for all $r, s \in \mathbb{R}_{\geq 0}$

b) $1 \cdot_R 1 > 1$ and $r \cdot_R s = (r^p + s^p)^{1/p}$ for all $r, s \in \mathbb{R}_{\geq 0}$ for some $p > 0$.

While nothing actually forces us to chose a norm-inducing binary operation $\cdot_R$, the theorem exposed the special position that maximum and $p$-norms take and their behavior regarding idempotency which influences the existence of collapsing elements. Combining this with the well-known fact that $\lim_{p \to \infty}(r^p + s^p)^{1/p} = \max\{r, s\}$ gives priority to $p$-norms. Thus, the following are examples of real PVS:

- *Weighted sum*: $\mathrm{R}_1 = (\mathbb{R}_{\geq 0}, \cdot_1, 0, \geq)$ with $r \cdot_1 s = r + s$;
- *Sum of squares*: $\mathrm{R}_2 = (\mathbb{R}_{\geq 0}, \cdot_2, 0, \geq)$ with $r \cdot_2 s = \sqrt{r^2 + s^2}$;
- *p-norm* for $p > 0$: $\mathrm{R}_p = (\mathbb{R}_{\geq 0}, \cdot_p, 0, \geq)$ with $r \cdot_p s = (r^p + s^p)^{1/p}$;
- *Worst case*: $\mathrm{R}_\infty = (\mathbb{R}_{\geq 0}, \cdot_\infty, 0, \geq)$ with $r \cdot_\infty s = \max\{r, s\}$.

Since, contrary to the max operator, $\cdot_p$ is not closed over an arbitrary set $V \subseteq \mathbb{R}_{\geq 0}$ and $p > 0$, we let $\langle V \rangle_p$ be the smallest subset of $\mathbb{R}_{\geq 0}$ such that $0 \in \langle V \rangle_p$ and $r \cdot_p s \in \langle V \rangle_p$ if $r, s \in \langle V \rangle_p$. Then we obtain a real PVS $(\langle V \rangle_p, \cdot_p, 0, \geq)$. For a $V \subseteq \mathbb{R}_{\geq 0}$, let $V_\infty$ denote the real PVS $(V \cup \{0\}, \cdot_\infty, 0, \geq)$, and let $V_p$ denote the PVS $(\langle V \rangle_p, \cdot_p, 0, \geq)$ for $p > 0$.

As hinted by the theorem of Bohnenblust, all real PVS $R$ with $\cdot_R = \cdot_p$ for some $p > 0$ have no collapsing elements, since $r \cdot_p s = (r^p + s^p)^{1/p}$ is strictly monotonic in both arguments. For real meet PVS with $\cdot_R = \cdot_\infty$, however, $\mathcal{C}(R) = |R| \setminus \{0\}$, since $\cdot_\infty$ is idempotent. Hence our

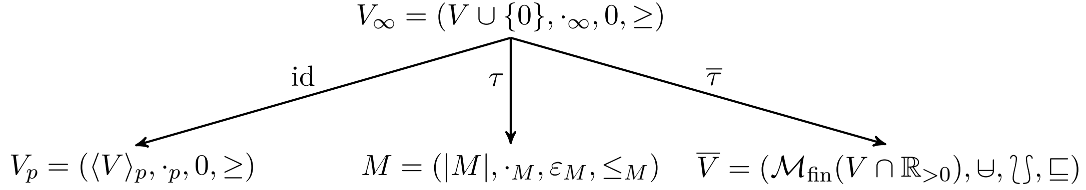



Figure 7.3: Conceptual overview of the PVS that serve as targets for homomorphisms from $V_\infty$. Here, "id" refers to the identity *function* ($\text{id}(x) = x$ for $x \in \mathbb{R}$) and not the categorical identity morphism $\text{id}_{V_\infty}$. While $M$ represents a generic PVS, $V_p$ and $\overline{\tau}$ are specific PVS based on $p$-norms and real-valued multisets, respectively.

goal is to replace $V_\infty$ by $V_p$ (with a sufficiently large $p$), similar to (yet more involved than) replacing the set-based PVS by the multiset-based PVS in Section 7.2.1.[1]

## 7.3 A Mapping from the Maximum PVS to a $p$-Norm PVS

Replacing $V_\infty$ by $V_p$ with sufficiently large $p$ seems intuitive and promising. But what does "sufficiently large" mean? When we search for a criterion to choose $p$ such that $V_p$ can order elements in lieu of $V_\infty$, it is helpful to start with specific examples. With regard to Table 7.1, let $\vec{r} = [2, 2, 2]$ and $\vec{s} = [0, 0, 3]$ be two real-valued vectors. In $V_\infty$, the single occurrence of 3 in $\vec{s}$ is more dominant than the three occurrences of 2 in $\vec{r}$, i.e., $\prod_\infty \vec{r} = 2 < 3 = \prod_\infty \vec{s}$ whereas for small values of $p$, the three occurrences of 2 can "overpower" the single 3. For instance, $\prod_2 \vec{s} = 3$ but $\prod_2 \vec{r} = \sqrt[2]{2^2 + 2^2 + 2^2} \approx 3.46$ which is greater. The larger $p$, the more dominant the maximal components become. Here, just selecting $p = 3$ already orders $\vec{r}$ and $\vec{s}$ correctly.

A key idea is that if $\max \vec{r} = \prod_\infty \vec{r}$ is strictly less than $\max \vec{s} = \prod_\infty \vec{s}$ (i.e., $\vec{s}$ is considered *worse* than $\vec{r}$), no matter how small the difference between $\max \vec{r}$ and $\max \vec{s}$ is, it must not be that even aggregating $\max \vec{r}$ for $n$ times can become worse than a single occurrence of $\vec{s}$. It turns out that this principle holds generally for any PVS that we can map to from $V_\infty$ via a PVS homomorphism. Figure 7.3 presents the PVS involved in the following derivations.

Formally, let $V \subseteq \mathbb{R}_{\geq 0}$ with $0 \in V$, $M$ any PVS, and $\tau : V_\infty \to M$ a PVS homomorphism. For a $\vec{r} = (r_i)_{1 \leq i \leq n} \in V^n$, we write $\tau(\vec{r})$ for $(\tau(r_i))_{1 \leq i \leq n}$. Our goal is to find criteria for $M$ and $\tau$ such that if we have $\max \vec{r} = \prod_\infty \vec{r} < \prod_\infty \vec{s} = \max \vec{s}$ (i.e., $\vec{s}$ is considered *worse* in $V_\infty$), we also have $\prod_M \tau(\vec{r}) >_M \prod_M \tau(\vec{s})$ in $M$ (i.e., $\tau(\vec{s})$ is also considered *worse* in $M$). For an $m \in |M|$, we write $m^{(n)_M}$ for the $n$-fold product of $m$ w.r.t. $\cdot_M$.

**Lemma 7.4.** *Let $V \subseteq \mathbb{R}_{\geq 0}$ with $0 \in V$, $M$ any PVS, and $\tau : V_\infty \to M$ a PVS homomorphism. For each $n \geq 1$ we have*

$$\textstyle\prod_\infty \vec{r} < \prod_\infty \vec{s} \quad \textit{implies} \quad \prod_M \tau(\vec{r}) >_M \prod_M \tau(\vec{s}) \quad \textit{for all } \vec{r}, \vec{s} \in V^n \qquad (*)$$

*if, and only if,*

$$r < s \quad \textit{implies} \quad \tau(r)^{(n)_M} >_M \tau(s) \quad \textit{for all } r, s \in V. \qquad (**)$$

[1] Given the recent popularity of neural networks, one might wonder if the "softmax" function could also be used as a replacement for max. However, softmax is a unary vector function and cannot be reduced to an associative binary operation.

*Proof.* Indeed, we let first $(*)$ hold and show that $(**)$ holds as well. Hence, we assume $r, s \in V \subseteq \mathbb{R}_{\geq 0}$ with $r < s$ and need to show $\tau(r)^{(n)_M} >_M \tau(s)$. Construct the vectors $\vec{r} = [r, \ldots, r]$ and $\vec{s} = [0, 0, \ldots, s]$. Since $\prod_\infty \vec{r} = r < s = \prod_\infty \vec{s}$, we get $\prod_M \tau(\vec{r}) = \tau(r)^{(n)_M} >_M \prod_M \tau(\vec{s}) = \tau(0) \cdot_M \tau(0) \cdot_M \ldots \cdot_M \tau(s) = \varepsilon_M^{(n-1)_M} \cdot_M \tau(s) = \tau(s)$ by $(*)$.

Now conversely, assume $(**)$ holds and we show $(*)$ holds. Hence, let $\prod_\infty \vec{r} < \prod_\infty \vec{s}$. We call $r_m = \prod_\infty \vec{r}$ and $s_m = \prod_\infty \vec{s}$ and define the vector $\vec{r}^* = [r_m, \ldots, r_m]$. Since $r_m < s_m$, we also have $\tau(r)^{(n)_M} = \prod_M \tau(\vec{r}^*) >_M \tau(s)$. In addition, since for every $1 \leq i \leq n$ we have $\vec{r}_i \leq r_m$ and thus $\tau(\vec{r}_i) \geq_M \tau(r_m)$ by $\tau$ being a PVS homomorphism, we also get $\prod_M \tau(\vec{r}) \geq_M \prod_M \tau(\vec{r}^*)$ by the monotonicity of $\cdot_M$. Hence, $\prod_M \tau(\vec{r}) >_M \prod_M \tau(\vec{s})$. Moreover, $\prod_M \tau(\vec{s}) = \tau(s_m) \cdot \prod_M \tau(\vec{s}_{\text{-}1})$ with $\vec{s}_{\text{-}1}$ being the remainder vector obtained from $\vec{s}$ by removing one instance of $s_m$. Moreover, since $M$ is a PVS, we have $\tau(s_m) \cdot \prod_M \tau(\vec{s}_{\text{-}1}) \leq_M \tau(s_m)$. Combining all facts leads us to the desired result: $\prod_M \tau(\vec{r}) \geq_M \prod_M \tau(\vec{r}^*) >_M \tau(s_m) \geq_M \prod_M \tau(\vec{s})$. □

As a matter of fact, $(**)$ yields some restrictions on useful choices of $V$, $M$, and $\tau$ when it comes to lexicographic products. We have $\tau(r)^{(n)_M} \leq_M \tau(r)$ since $M$ is a PVS. If $\tau(r) = \tau(r)^{(n)_M}$, then $\tau(r)$ is idempotent, since $\tau(r) = \tau(r)^{(n)_M} \leq_M \tau(r)^{(n-1)_M} \leq_M \ldots \leq_M \tau(r) \cdot_M \tau(r) \leq_M \tau(r)$. If $\tau(r)^{(n)_M} <_M \tau(r)$, then there must be no $r' \in V$ with $r < r'$ that $\tau$ "sandwiches" between $\tau(r)^{(n)_M}$ and $\tau(r)$, i.e., $\tau(r)^{(n)_M} \leq_M \tau(r') \leq_M \tau(r)$, since otherwise, by choosing this $r'$ for $s$ in $(**)$, $\tau(r') <_M \tau(r)^{(n)_M} \leq_M \tau(r')$ would have to hold.

Moreover, plugging in the definitions of $p$-norm real PVS yields even more insight into feasible structures and homomorphisms.

**Corollary 7.5.** *As special cases of the equivalence of $(*)$ and $(**)$ for $V \subseteq \mathbb{R}_{\geq 0}$ with $0 \in V$, $p > 0$, and $\tau : V_\infty \to V_p$ with $\tau(r) = r$ we obtain for each $n \geq 1$:*

$$\textstyle\prod_\infty \vec{r} < \prod_\infty \vec{s} \quad \textit{implies} \quad \prod_p \vec{r} < \prod_p \vec{s} \quad \textit{for all } \vec{r}, \vec{s} \in V^n \tag{$*_p$}$$

*if, and only if,*

$$r < s \quad \textit{implies} \quad n^{\frac{1}{p}} \cdot r < s \quad \textit{for all } r, s \in V. \tag{$**_p$}$$

The term $n^{\frac{1}{p}}$ results from the $n$-fold application of $\cdot_p$ on $r$. Our ultimate goal is to find $p$ such that $(*_p)$ holds for a given *SCSP* where we assume $n$, the number of soft constraints to be given and constant. But $(*_p)$ is certainly not true for any $V \subseteq \mathbb{R}_{\geq 0}$. To find useful restrictions, $(**_p)$ provides guidance: We have seen this equation in action in our introductory example with $\vec{r} = [2, 2, 2]$ and $\vec{s} = [0, 0, 3]$. Here, even $n$ occurrences of 2 must not exceed a single value of 3 in the respective $p$-product.

Since $(**_p)$ has to hold no matter how small the difference between $r$ and $s$ is, it makes sense to consider the most extreme case first since then a chosen $p$ will be high enough for larger differences. If $r = 0$, $(**_p)$ holds for any $s > 0$. Hence, assume that we can express the smallest quotient $\frac{s}{r}$ for $0 \neq r < s$ between any values in $V$ by $\delta$, e.g., here $\delta = \frac{3}{2} = 1.5$. Instantiating $(**_p)$ yields that $r < \delta r$ has to imply $n^{\frac{1}{p}} \cdot r < \delta r$. But since $n^{\frac{1}{p}}$ decreases with increasing $p$ and $\delta$ is constant, we can surely find some $p$ satisfying this. More precisely, for some $r > 0$, we get

$$n^{\frac{1}{p}} \cdot r < \delta r \iff n^{\frac{1}{p}} < \delta \iff \frac{1}{p} \ln(n) < \ln(\delta) \iff \frac{\ln(n)}{\ln(\delta)} < p\ .$$

For instance, with $\delta = 1.5$ and $n = 3$ the desired property holds for all $p > \frac{\ln(3)}{\ln(1.5)} \approx 2.71$ which confirms our intuition gained from Table 7.1. Therefore, sets $V$ that admit a smallest quotient $\delta$ are amenable to PVS based on $p$-norms. We call such sets $\delta$-separated. Formally,

**Definition 7.1 – $\delta$-Separation**
A set $V \subseteq \mathbb{R}_{\geq 0}$ with $0 \in V$ is *$\delta$-separated* for some $\delta > 1$ if $\frac{s}{r} \geq \delta$ for all $0 \neq r < s \in V$.

Conversely, if this is not the case, i.e., if for each $\delta > 1$ there are $r < s \in V$ with $\frac{s}{r} < \delta$, then $(**_p)$ is violated for each $p > 0$: Let any $p > 0$ be given and pick $r < s \in V$ such that $r \neq 0$ and $\frac{s}{r} < n^{\frac{1}{p}}$. Then $n^{\frac{1}{p}} \cdot r > s$. In particular, this prohibits sets such as $\mathbb{R}_{\geq 0}$ itself or $[0, r]$ for some $r \in \mathbb{R}_{\geq 0}$. Still, there are several practical cases that involve $\delta$-separated sets.

**Example 7.3 – Examples for $\delta$-separation**
(1) The set $V = \{0, 2.5, \pi, 5, 8, 9\}$ is $\delta$-separated with $\delta = \frac{9}{8} = 1.125$.

(2) More generally, let $V \subseteq \mathbb{R}_{\geq 0}$ with $0 \in V$ be finite. Then there is an $\varepsilon > 0$ such that $|r_1 - r_2| \geq \varepsilon$ for all $r_1 \neq r_2 \in V$. Let $0 \neq r < s \in V$. Then $\frac{s}{r} \geq \frac{r+\varepsilon}{r} = 1 + \frac{\varepsilon}{r} \geq 1 + \frac{\varepsilon}{\max V}$. Thus $V$ is $(1 + \frac{\varepsilon}{\max V})$-separated. For the special but frequently occurring case of $V = \{0, \ldots, k\} \subseteq \mathbb{N}$ for some $k \in \mathbb{N}$, i.e., we have $\varepsilon = 1$ and $\max V = k$ which results in $\frac{k+1}{k}$-separation. For this case, we get a tighter $\delta$ bound since $V$ is also $\frac{k}{k-1}$-separated.

(3) For powers of a given basis $c \in \mathbb{R}$ with $c > 1$, we let $V^c = \{c^n \mid n \in \mathbb{N}\} \cup \{0\}$. If $0 \neq r < s \in V^c$, then there are $m < n$ with $r = c^m$ and $s = c^n$; then $\frac{c^n}{c^m} = c^{m-n} \geq c$. Thus $V^c$ is $c$-separated and unbounded.

(4) Alternatively, let $d \in \mathbb{R}$ with $d > 1$ and let $V^{\frac{1}{d}} = \{d^{-n} \mid n \in \mathbb{N}\} \cup \{0\}$. If $0 \neq r < s \in V^{\frac{1}{d}}$, then there are $m$ and $n$ with $r = d^{-n}$ and $s = d^{-m}$ and thus $m < n$. Then $\frac{d^{-m}}{d^{-n}} = d^{n-m} \geq d$ holds. In addition, $0 < d^{-n} \leq d$ for all $n \in \mathbb{N}$. Hence $V^{\frac{1}{d}}$ is $d$-separated and also bounded.

Consequently, if some set $V$ is $\delta$-separated and $V_\infty$ was the original target of a PVS-based soft constraint problem with $n$ soft constraints that map to $V$ and are combined by $\cdot_\infty$, we can find a $p$ that is large enough to "replace" $V_\infty$. Whenever an assignment $\theta$ is deemed better in $V_\infty$ than another one $\theta'$, e.g., the maximal violation of $\theta$ is smaller than that of $\theta'$, the same decision would be made in $V_p$. In the next section, we formalize this principle to obtain a more general notion of "replacing PVS" in optimization problems.

## 7.4 Optima-Simulation for PVS-based Constraint Problems

When designing a soft constraint problem, modelers arguably care most about an induced ordering over *solutions* to a constraint problem rather than what particular set and ordering are used in the algebraic structures – or even only about the set of optimal solutions. As a trivial example for this, consider that in any *COP* with objective function $f : [X \to D] \to \mathbb{R}$ and the natural ordering over reals, a solution maximizing $f$ would also maximize any monotonic function composed to $f$ such as e.g., $2f$ or $f^3$. In practice, this is often exploited, e.g., by taking the logarithm of an objective to achieve better numerical stability. In a similar spirit, we aim to replace a PVS, say $V_\infty$, by another one, say $V_p$, given that certain criteria hold. Yet,

we argue that the relationship between both partial orders requires more care and restriction than just using any PVS homomorphism.

A similar effort was made by Bistarelli, Codognet, and Rossi, who discuss abstractions of c-semiring-based soft constraint problems by means of Galois connections [Bistarelli et al., 2002] which was later shown to amount to a semiring isomorphism by Li and Ying [2008] – hence being too restrictive for true abstraction. The problem can also be seen in the context of viewpoints in model reformulation [Smith, 2006] in the sense that we seek an alternative PVS that reflects the same underlying user preferences.

Before diving into our proposal of formalizing replacement PVS, we state more precisely what classes of soft constraint problems are affected by it.

### 7.4.1 Admissible Soft Constraint Problems

Even though most (soft) constraint problems considered in practice are finite in terms of the number of soft constraints as well as the search space, for the development of our mathematical model this restriction not necessarily applies. For the optimization task to be meaningful though, we seek a sufficiently precise criterion that still allows for infinite domains in the problem specifications. Intuitively, we consider optimization useful if there *is* an optimal value to achieve – as opposed to, e.g., an unreachable upper bound in a continuous problem. We call this criterion *admissibility* and now develop it more formally.

Recall that, given a constraint problem $CSP = (X, D, C)$ and a PVS $M$, an *$M$-soft constraint* $\mu$ over $(X, D)$ is given by a map $\mu : [X \to D] \to |M|$. The *maximum solution degrees* of $\mu$ are given by

$$\mu^* = \mathrm{Max}^{\leq_M} \{\mu(\theta) \mid \theta \in [X \to D]\} \ ,$$

and similarly the *optimal solutions* by

$$\widehat{\mu}([X \to D]) = \{\theta \in [X \to D] \mid \mu(\theta) \in \mu^*\} \ .$$

An $M$-soft constraint $\mu$ is *admissible* if for each $\theta \in [X \to D]$ there is an $m \in \mu^*$ such that $\mu(\theta) \leq_M m$. A finite set $C_{\mathrm{s}}^M$ of $M$-soft constraints can be viewed as the $M$-soft constraint $C_{\mathrm{s}}^M(\theta) = \prod_M \wr \mu(\theta) \mid \mu \in C_{\mathrm{s}}^M \wr$ for $\theta \in [X \to D]$. Such a set $C_{\mathrm{s}}^M$ is *admissible* if $C_{\mathrm{s}}^M$ is finite and every $M$-soft constraint in $C_{\mathrm{s}}^M$ is admissible.

**Example 7.4 – Admissibility**
Let $\mu$ be an $M$-soft constraint over $(X, D)$ for a PVS $M$.

(1) If $(X, D)$ is finite, then $\mu$ is admissible.

(2) If $<_M$ has no infinite strictly ascending chains, then $\mu$ is admissible.

(3) To see a non-admissible soft constraint, let $X = \{x\}$, $D_x = [0, 1]$, i.e., the continuous closed real-valued interval from 0 to 1, and $(|M|, \leq_M) = ([0, 1], \geq)$. Put more conventionally, we seek to *minimize* the objective value in $|M|$. Assume $\mu : [X \to D] \to |M|$ to be defined by $\mu(\{x \mapsto r\}) = r$ if $r > 0$, and $\mu(\{x \mapsto 0\}) = 1$. Then $\mu^* = \emptyset$, since the set of solution degrees is the open interval $]0, 1]$ with 0 the optimum, which, however, cannot be reached by any assignment. Thus $\mu$ is not admissible.

Admissibility asserts that a search and propagation algorithm exploring the search space of a given *CSP* may at least be guaranteed to find an optimal solution in finite time. The concept is related to proving unboundedness in linear programming which avoids continuing the solving process. Admissibility can be seen as a necessary criterion for any optimization procedure although, in principle, so-called anytime algorithms could optimize indefinitely, terminating only upon reaching time-outs.

### 7.4.2 Substituting PVS for Optimization: Optima-Simulation and Optima-Equivalence

Resuming the mapping presented in Section 7.3, we saw that applying a PVS based on $p$-norms to $n$ distinct real values can result in the same judgment as if the maximum operator were used. It thus constitutes a promising candidate to replace $V_\infty$ since it is furthermore free of collapsing elements. For example, $\prod_\infty[2,2,2] = 2 < 3 = \prod_\infty[0,0,3]$ and also $\prod_3[2,2,2] \approx 2.88 < 3 = \prod_3[0,0,3]$ which is a result that we derived from $\delta$-separation. But what if we compared $[3,3,3]$ to $[0,0,3]$? Then $\prod_\infty[3,3,3] = 3 = 3 = \prod_\infty[0,0,3]$ but, e.g., $\prod_3[3,3,3] \approx 4.33 > 3 = \prod_3[0,0,3]$. More generally, for any $p > 1$, $\prod_p[3,3,3] > 3 = \prod_p[0,0,3]$ since $\cdot_p$ is strictly monotonic in both arguments. Should this be acceptable? Arguably, many solvers would only present either of the two equal solutions in a strict optimization algorithm – most likely only the first. This "non-determinism" would be replaced by the induced ordering but the modeler is not supposed to distinguish between both assignments, otherwise they should have been evaluated differently, in the first place. On the other hand, often users might prefer to inspect a whole set of equally valued assignments in which case the induced distinction is more harmful.

This section discusses whether this induced ordering between elements that were originally mapped to the *same* value is still acceptable. Aiming for a rational justification, we introduce an asymmetric notation called *optima simulation* that regulates the relationship between, e.g., $V_\infty$ and $V_p$ but is also more widely applicable for homomorphisms involving any PVS.

Formally, for a *CSP* given by $(X, D, C)$, consider two PVS $M$ and $N$. Additionally, let $C_\mathrm{s}^M$ and $C_\mathrm{s}^N$ be finite sets of $M$ and $N$-soft constraints, respectively. Each of them induces a quasi-ordering over assignments (antisymmetry does not necessarily hold since unequal assignments may of course map to the same element in $|M|$ or $|N|$). Hence both sets give rise to different optimization problems.

Most obviously, we say that $C_\mathrm{s}^M$ and $C_\mathrm{s}^N$ are *optima equivalent*, written as $C_\mathrm{s}^M \approx C_\mathrm{s}^N$, if assignments from $[X \to D]$ are optimal w.r.t. $C_\mathrm{s}^M$ if and only if they are optimal w.r.t. $C_\mathrm{s}^N$. A sufficient but stronger notion for optima equivalence is ordering equivalence, i.e., that $C_\mathrm{s}^M(\theta') \leq_M C_\mathrm{s}^M(\theta)$ if, and only if, $C_\mathrm{s}^N(\theta') \leq_N C_\mathrm{s}^N(\theta)$ for all $\theta, \theta' \in [X \to D]$. We have seen an instance of optima equivalence in Section 7.2.1 where we replaced a set-based PVS by a multiset-based variant to avoid collapsing elements.

However, optima equivalence may sometimes be too restrictive for practical purposes. Therefore, a weaker requirement is to say that we can use PVS $N$ with $C_\mathrm{s}^N$ *instead* of PVS $M$ with $C_\mathrm{s}^M$ – as long as every assignment deemed optimal with $N$ is also considered optimal with $M$ (we are not "making suboptimal assignments optimal") and some (but not all) optimal assignments are still optimal with $N$ (we are losing a "tolerable amount of optima"). Figure 7.4 visualizes this idea which is formally presented in the following definition.

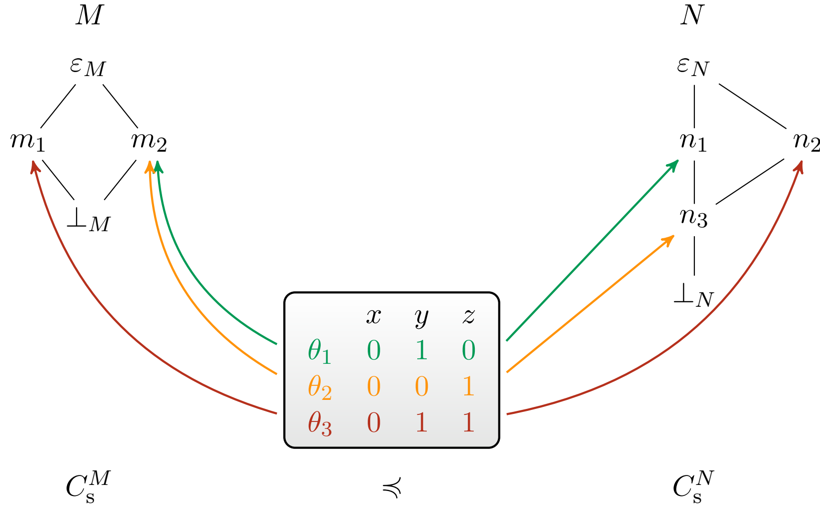



Figure 7.4: An example to illustrate optima simulation. A problem specified using a set of soft constraints $C_s^M$ mapping to a PVS $M$ can be simulated by the set $C_s^N$ mapping to $N$ if for every reachable optimum w.r.t. $M$ there is at least one solution that is still optimal w.r.t. $N$ (here $\theta_1$ is still optimal in $C_s^N$ but $\theta_2$ is not any more). Conversely, any assignment deemed optimal w.r.t. $C_s^N$ must also be optimal w.r.t. $C_s^M$. It may only happen that equally optimal solutions are ordered (i.e., distinguished) in $N$ by the mapping of $C_s^N$.

**Definition 7.2 – Optima simulation**
$C_s^N$ *optima simulates* $C_s^M$, written as $C_s^M \preccurlyeq C_s^N$, if for each $\theta_M \in \widehat{C_s^M}([X \to D])$ there is a $\theta_N \in \widehat{C_s^N}([X \to D])$ with $C_s^M(\theta_M) = C_s^M(\theta_N)$, and, vice versa, if for each $\theta_N \in \widehat{C_s^N}([X \to D])$ there is a $\theta_M \in \widehat{C_s^M}([X \to D])$ with $C_s^M(\theta_M) = C_s^M(\theta_N)$.

Obviously, $C_s^M \approx C_s^N$ if, and only if, $C_s^M \preccurlyeq C_s^N$ and $C_s^N \preccurlyeq C_s^M$. Intuitively, our definition of optima simulation allows that assignments in the same equivalence class w.r.t. $C_s^M$ are *further* distinguished in $C_s^N$ as long as each equivalence class in the $M$-optimal assignments is represented in $N$. This implies that the set of optimal assignments according to $C_s^N$ is a subset of the optimal assignments for $C_s^M$, i.e,. $\widehat{C_s^N}([X \to D]) \subseteq \widehat{C_s^M}([X \to D])$. We can furthermore give sufficient criteria for the relations of assignments evaluated in $C_s^M$ and $C_s^N$ to check if $C_s^N \preccurlyeq C_s^M$ holds, provided that both $C_s^M$ and $C_s^N$ are admissible:

**Lemma 7.6.** *Let $M$ and $N$ be PVS and let $C_s^M$ be an admissible set of $M$-soft constraints and $C_s^N$ an admissible set of $N$-soft constraints such that*

$$C_s^M(\theta) <_M C_s^M(\theta') \quad \textit{implies} \quad C_s^N(\theta) <_N C_s^N(\theta')$$
$$C_s^M(\theta) \parallel_M C_s^M(\theta') \quad \textit{implies} \quad C_s^N(\theta) \parallel_N C_s^N(\theta')$$

*for all $\theta, \theta' \in [X \to D]$. Then $C_s^M \preccurlyeq C_s^N$.*

*Proof.* To show $C_s^M \preccurlyeq C_s^N$, let first $\theta_M \in \widehat{C_s^M}([X \to D])$ be an optimal assignment according to $C_s^M$ that is simultaneously not optimal according to $C_s^N$, i.e., $\theta_M \notin \widehat{C_s^N}([X \to D])$. Then, since

$C_s^N$ is admissible, there exists a $\theta_N \in \widehat{C_s^N}([X \to D])$ with $C_s^N(\theta_M) <_N C_s^N(\theta_N)$. Moreover, there is a $\theta'_M \in \widehat{C_s^M}([X \to D])$ with $C_s^M(\theta_N) \leq_M C_s^M(\theta'_M)$, since $C_s^M$ is also admissible. But it cannot be that $C_s^M(\theta_N) <_M C_s^M(\theta'_M)$ since then we would also have $C_s^N(\theta_N) <_N C_s^N(\theta'_M)$ by the assumed premise, and thus $\theta_N$ would not be optimal, i.e., contradicting $\theta_N \in \widehat{C_s^N}([X \to D])$. Hence, $C_s^M(\theta_N) = C_s^M(\theta'_M)$.

Moreover, either $C_s^M(\theta_M) \parallel_M C_s^M(\theta'_M)$ or $C_s^M(\theta_M) = C_s^M(\theta'_M)$ since both $\theta_M$ and $\theta'_M$ are elements of $\widehat{C_s^M}([X \to D])$. But $C_s^M(\theta_M) \parallel_M C_s^M(\theta'_M)$ is impossible since then $C_s^M(\theta_M) \parallel_M C_s^M(\theta'_M) = C_s^M(\theta_N)$ and consequently also $C_s^N(\theta_M) \parallel_N C_s^N(\theta_N)$ by the assumed premise. But we already established $C_s^N(\theta_M) <_N C_s^N(\theta_N)$, hence $C_s^M(\theta_M) = C_s^M(\theta'_M) = C_s^M(\theta_N)$. Thus it must be that $\theta_M \in \widehat{C_s^N}([X \to D])$, contrary to our assumption.

Now let conversely be $\theta_N \in \widehat{C_s^N}([X \to D])$. If $\theta_N$ were not optimal w.r.t. $C_s^M$, i.e., $\theta_N \notin \widehat{C_s^M}([X \to D])$, there would be a $\theta_M$ with $C_s^M(\theta_N) <_M C_s^M(\theta_M)$ since $C_s^M$ is admissible. But then also $C_s^N(\theta_N) <_N C_s^N(\theta_M)$ which contradicts the optimality of $\theta_N$ w.r.t. $C_s^N$. Thus, any assignment optimal w.r.t. $C_s^N$ must also be optimal w.r.t. $C_s^M$. □

Instead of checking for incomparability, we can also state an alternative variant of the lemma. From the conditions of the lemma it follows that $C_s^N(\theta) \leq_N C_s^N(\theta')$ implies $C_s^M(\theta) \leq_M C_s^M(\theta')$ for all $\theta, \theta' \in [X \to D]$: To show this, let the conditions of the lemma hold and for a contradiction, assume $C_s^N(\theta) \leq_N C_s^N(\theta')$ but $C_s^M(\theta) \not\leq_M C_s^M(\theta')$. Then either $C_s^M(\theta') <_M C_s^M(\theta)$ or $C_s^M(\theta) \parallel_M C_s^M(\theta')$, implying $C_s^N(\theta') <_N C_s^N(\theta)$ or $C_s^N(\theta) \parallel_N C_s^N(\theta')$ which both contradict $C_s^N(\theta) \leq_N C_s^N(\theta')$.

In addition, from the requirement that $C_s^N(\theta) \leq_N C_s^N(\theta')$ implies $C_s^M(\theta) \leq_M C_s^M(\theta')$ for all $\theta, \theta' \in [X \to D]$ it follows that $C_s^M(\theta) \parallel_M C_s^M(\theta')$ implies $C_s^N(\theta) \parallel_N C_s^N(\theta')$ for all $\theta, \theta' \in [X \to D]$: If we had $C_s^M(\theta) \parallel_M C_s^M(\theta')$ but $C_s^N(\theta) \not\parallel_N C_s^N(\theta')$ then for either $C_s^N(\theta) \diamond_N C_s^N(\theta')$ with $\diamond \in \{<, >, =\}$, we would also get $C_s^M(\theta) \diamond_M C_s^M(\theta')$, contradicting the incomparability with $C_s^M$. Thus, the conditions of the lemma can be equivalently replaced by

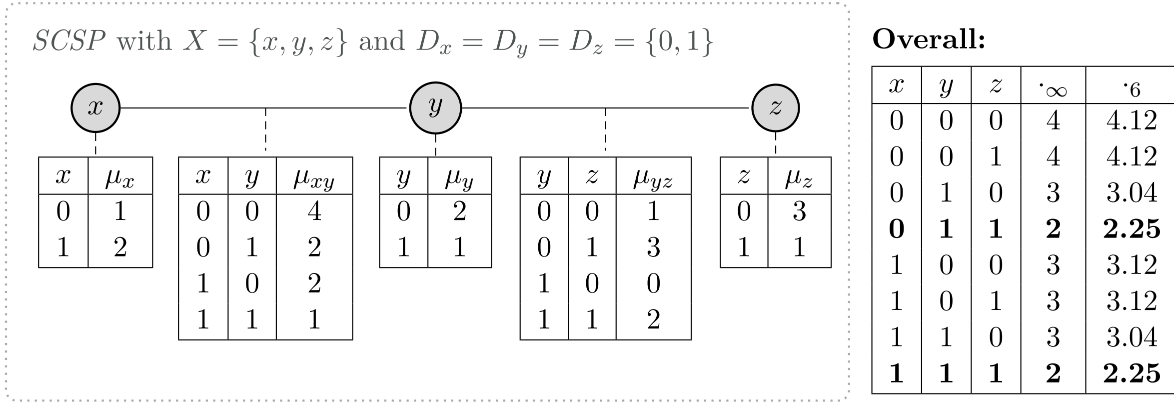
$$C_s^M(\theta) <_M C_s^M(\theta') \quad \text{implies} \quad C_s^N(\theta) <_N C_s^N(\theta')$$
$$C_s^N(\theta) \leq_N C_s^N(\theta') \quad \text{implies} \quad C_s^M(\theta) \leq_M C_s^M(\theta')$$

for all $\theta, \theta' \in [X \to D]$.

Intuitively, these conditions state that true domination in $C_s^M$ must remain true domination when moving to $C_s^N$ whereas equality in $C_s^M$ can lead to refined ordering in $C_s^N$. Moreover, in $C_s^N$, we cannot invert rank assignments made differently in $C_s^M$. The term optima-simulation for the relation $\preccurlyeq$ between sets of soft constraints $C_s^M$ and $C_s^N$ is loosely inspired by simulation relations for transition systems: An ordering decision made by $C_s^M$ can adequately be simulated by $C_s^N$ although it may truly refine the coarser ordering relation induced by $C_s^M$. In the next section, we put the definition of optima simulation into action by formalizing the relation between $V_\infty$ and a $V_p$ for corresponding sets of soft constraints.

### 7.4.3 Optima-Simulating Max by Large $p$-Norm PVS

**Lemma 7.7.** *Let* $(X, D, C)$ *be a CSP,* $V \subseteq \mathbb{R}_{\geq 0}$ *with* $0 \in V$ *be* $\delta$*-separated,* $C_s^{V_\infty}$ *a finite admissible set of* $V_\infty$*-soft constraints with* $|C_s^{V_\infty}| = n$*, and* $p > \frac{\ln(n)}{\ln(\delta)}$*. Define* $\tau_p : |V_\infty| \to |V_p|$ *by* $\tau_p(r) = r$ *and the finite set of* $V_p$*-soft constraints* $C_s^{V_p}$ *by* $C_s^{V_p} = \{\tau_p \circ \mu \mid \mu \in C_s^{V_\infty}\}$*.*
*If* $C_s^{V_p}$ *is admissible, then* $C_s^{V_\infty} \preccurlyeq C_s^{V_p}$*.*




| $x$ | $\mu_x$ |
|---|---|
| 0 | 1 |
| 1 | 2 |

| $x$ | $y$ | $\mu_{xy}$ |
|---|---|---|
| 0 | 0 | 4 |
| 0 | 1 | 2 |
| 1 | 0 | 2 |
| 1 | 1 | 1 |

| $y$ | $\mu_y$ |
|---|---|
| 0 | 2 |
| 1 | 1 |

| $y$ | $z$ | $\mu_{yz}$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 3 |
| 1 | 0 | 0 |
| 1 | 1 | 2 |

| $z$ | $\mu_z$ |
|---|---|
| 0 | 3 |
| 1 | 1 |

**Overall:**

| $x$ | $y$ | $z$ | $\cdot_\infty$ | $\cdot_6$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 4 | 4.12 |
| 0 | 0 | 1 | 4 | 4.12 |
| 0 | 1 | 0 | 3 | 3.04 |
| **0** | **1** | **1** | **2** | **2.25** |
| 1 | 0 | 0 | 3 | 3.12 |
| 1 | 0 | 1 | 3 | 3.12 |
| 1 | 1 | 0 | 3 | 3.04 |
| **1** | **1** | **1** | **2** | **2.25** |

Figure 7.5: An exemplary *SCSP* that has $V_6$ optima-simulate $V_\infty$. The $n = 5$ soft constraints are mapped to $\langle V \rangle_6$ instead of $V = \{0, 1, 2, 3, 4\}$ and for combination, $\cdot_6$ is in charge instead of $\cdot_\infty$ (the solution degree is likely to be only in $\langle V \rangle_6$ instead of $V$). Since $V$ is $\frac{4}{3}$-separable, choosing $p > \frac{\ln(n)}{\ln(\delta)} = \frac{\ln(5)}{\ln(4/3)} \approx 5.59$ is sufficient for optima simulation and we are guaranteed that at least one of the $V_\infty$ optimal solutions is also optimal in $V_6$ (here, even both).

*Proof.* To show $C_s^{V_\infty} \preccurlyeq C_s^{V_p}$, we may check the conditions developed in Lemma 7.6. Since incomparability is not relevant in the totally ordered real PVS $V_\infty$, we only need to consider the first condition which holds as a consequence of $(**_p)$ and the choice of $p$. □

Figure 7.5 wraps up the concepts that led up to this use case of optima simulation. Since $V_p$ is a PVS without collapsing elements and optima simulates $V_\infty$, we can safely use it instead for lexicographic products. The guarantees we get are those specified by optima simulation, i.e., at least one optimal solution from an equivalence class of optimal solution degrees will remain optimal and no $V_p$-optimal solution was $V_\infty$-suboptimal.

With respect to constraint hierarchies, this means that the class of problems that Hosobe [2009] excluded from his solution (remember that "worst-case-better" was not part of his "rational c-semirings") is now at least partially expressible in lexicographic products of PVS. We cannot expect to do much better than optima-simulation for $V_\infty$ since the max operator leads to a meaningless PVS in lexicographic products due to its idempotency.

Still, for this construction to work, $V$ needs to be $\delta$-separable and $n$ has to be fixed, but a scalar PVS type $\langle V \rangle_p \subseteq \mathbb{R}$ is obtained. Also, the required $p$ increases with increasing $n$ and decreasing $\delta$, fairly quickly for realistic problem sizes (see Section 7.5). If, on the other hand, structured non-scalar data types are acceptable, it turns out that the $p$-norm-based solution is not the only non-collapsing PVS that can optima-simulate $V_\infty$.

### 7.4.4 Optima-Simulating Max with Finite Multisets

Instead of using a scalar real-valued data type as the carrier set of an optima-simulating PVS, we can equivalently collect *all* soft constraints' valuations in a multiset (since gradings can occur more than once) and compare the resulting multisets of assignments by means of a suitable multiset ordering which is in fact a variant of the lexicographic ordering. Since we again assume that any *SCSP* only maps an assignment to finitely many soft constraint gradings, the set of *finite* multisets over reals (written as $\mathcal{M}_{\text{fin}}(\mathbb{R}_{>0})$) is sufficient. In this

multiset representation, any *positive* real value is collected in a multiset whereas zero values are not included.

**Definition 7.3 – Non-Collapsing Max-ordering over multisets**
The Non-Collapsing Max-ordering on $\mathcal{M}_{\text{fin}}(\mathbb{R}_{>0})$ is the binary relation $\sqsubseteq\ \subseteq \mathcal{M}_{\text{fin}}(\mathbb{R}_{>0}) \times \mathcal{M}_{\text{fin}}(\mathbb{R}_{>0})$, defined by $T \sqsubseteq U$ if, and only if, there is a $q > 0$ such that $\prod_p T \geq \prod_p U$ for all $p > q$, where $\prod_p \wr\int = 0$ and $\prod_p(T \uplus \wr r\int) = r \cdot_p \prod_p T$.

For instance, $\prod_p \wr 3\int > \prod_p \wr 2,2,2\int$ for all $p \geq 3$. Then $\sqsubseteq$ is a partial order, where reflexivity and transitivity are obvious, and we only have to demonstrate antisymmetry: Let $T, U \in \mathcal{M}_{\text{fin}}(\mathbb{R}_{>0})$ with $T \sqsubseteq U$ and $U \sqsubseteq T$. Then there is a $q_{TU} > 0$ such that $\prod_p T \geq \prod_p U$ for all $p > q_{TU}$, and equivalently a $q_{UT} > 0$ such that $\prod_p U \geq \prod_p T$ for all $p > q_{UT}$. Hence $\prod_p T = \prod_p U$ for all $p > \max\{q_{TU}, q_{UT}\}$ (which does not guarantee $T = U$, yet). Since $\lim_{p\to\infty} \prod_p T = \prod_\infty T$, we either have that $T = \wr\int = U$ or that there is an $r \in \mathbb{R}_{>0}$ with $\max T = r = \max U$. In the latter case, with $T = T' \uplus \wr r\int$, $U = U' \uplus \wr r\int$, we have $r \cdot_p \prod_p T' = r \cdot_p \prod_p U'$ for all $p > \max\{q_{TU}, q_{UT}\}$ and thus $\prod_p T' = \prod_p U'$ for all $p > \max\{q_{TU}, q_{UT}\}$ since $\cdot_p$ is strictly monotonic in both arguments. Thus, $T = U$ follows by induction on the size of $T$.

For any $V \subseteq \mathbb{R}_{\geq 0}$, we consider the structure $\overline{V} = (\mathcal{M}_{\text{fin}}(V \cap \mathbb{R}_{>0}), \uplus, \wr\int, \sqsubseteq)$. If $T \sqsubset U$, then $T \uplus V \sqsubset U \uplus V$: Let $\prod_p T > \prod_p U$ for all $p > q$ for some $q > 0$. Then $\prod_p(T \uplus V) = (\prod_p T) \cdot_p (\prod_p V) > (\prod_p U) \cdot_p (\prod_p V) = \prod_p(U \uplus V)$ for all $p > q$, since $\cdot_p$ is strictly monotonic in both arguments. This proves strict monotonicity for $\overline{V}$. We thus have that $\overline{V} = (\mathcal{M}_{\text{fin}}(V), \uplus, \wr\int, \sqsubseteq)$ is a PVS that has no collapsing elements.

In fact, checking whether $T \sqsubseteq U$ does not have to involve evaluating any $p$-norms other than the $\infty$-norm: Let $T, U \in \mathcal{M}_{\text{fin}}(\mathbb{R}_{>0})$ be given. If $U = \wr\int$, then $T \sqsubseteq U$, and if $T = \wr\int$, then $T \sqsupseteq U$. Thus we are left with the case that $T \neq \wr\int$ and $U \neq \wr\int$; in particular, $\max T > 0$ and $\max U > 0$. If $\max T > \max U$, then $T \sqsubset U$, since then $\lim_{p\to\infty} \prod_p T = \max T > \max U = \lim_{p\to\infty} \prod_p U$; conversely, if $\max T < \max U$, then $T \sqsupset U$. Hence, we are now left with the case that $\max T = \max U$. But then we proceed recursively for $T'$ and $U'$ with $T = \wr\max T\int \uplus T'$ and $U = \wr\max U\int \uplus U'$. This procedure gives us a simple reduction to checking lexicographic orderings on ordered vectors of reals: For multisets $T$ and $U$, we obtain descendingly ordered vectors (e.g., $\wr 2,1,4,2\int \mapsto [4,2,2,1]$) and proceed to lexicographically order those vectors starting from the first component until there is a strict difference or one of the vectors has become empty.

Thus, $\sqsubseteq$ is a variant of lexicographically ordering a multiset. Intriguingly, the following relation with the Sym-Diff-Hoare-ordering of Definition 6.3 holds which also provided a multiset lifting for the TPD-semantics of constraint preferences.

**Lemma 7.8.** $\sqsubseteq\ =\ \sqsubseteq^{\mathrm{R}}$ *for* $\mathrm{R} = (\mathbb{R}_{>0}, \geq)$.

*Proof.* Let first $T \sqsubseteq^{\mathrm{R}} U$ hold, i.e., $U \sqsubseteq_{\mathrm{R}^{-1}} T$ and we can split $T = T' \uplus V$ and $U = U' \uplus V$, such that $T'$ and $U'$ have no common elements, and for all $r \in U'$ there is an $s \in T'$ such that $r < s$. If $U' = \wr\int$, then $T \supseteq U$ and thus for any $p > 0$: $\prod_p T = \prod_p T' \cdot_p \prod_p V \geq \prod_p V = \prod_p U$, confirming $T \sqsubseteq U$. Otherwise $U' \neq \wr\int$ and we get an $r$ as the maximum of $U'$. Let $s$ be the maximum in $T'$. By the definition of $\sqsubseteq_{\mathrm{R}^{-1}}$, we have $r < s$. Additionally, $\prod_p T' \geq s > r$ and $\prod_p R = |U'|^{1/p} r \geq \prod_p U'$ for any $p > 0$ where $R$ is a multiset containing $|U'|$ copies of $r$. Since

$s > r$ and $|U'|^{1/p}$ decreases with rising $p$, we can choose a $q \in \mathbb{R}_{>0}$ such that $s \geq |U'|^{1/q} r$. Then $\prod_p T' \geq s \geq \prod_p R = |U'|^{1/p} r \geq \prod_p U'$ for all $p > q$.

Conversely, now assume $\prod_p T \geq \prod_p U$ for all $p > q$ for some $q \in \mathbb{R}_{>0}$. Split again $T = T' \uplus V$ and $U = U' \uplus V$, such that $T'$ and $U'$ have no common elements. To prove $T \sqsubseteq^{\mathrm{R}} U$, i.e., $U \sqsubseteq_{\mathrm{R}^{-1}} T$, we have to show that for all $r \in U'$ there is an $s \in T'$ such that $r < s$. If $U'$ is empty this claim obviously holds. Thus, we consider the case where $U'$ is non-empty. Since $\prod_p T \geq \prod_p U$, also $\prod_p T' \geq \prod_p U'$ for all $p > q$. Let $r$ be the maximum of $U'$. Then $T'$ must be non-empty, since $\prod_p T' \geq \prod_p U' > 0$ must hold since $U'$ is non-empty and this would be impossible if $\prod_p T' = \prod_p \wr\int = 0$. Hence, let $s$ be the maximum of $T'$. Since $s = \lim_{p \to \infty} \prod_p T'$ and $r = \lim_{p \to \infty} \prod_p U'$, we deduce $s \geq r$ from $\prod_p T' \geq \prod_p U'$ for all $p > q$; and $s > r$, since otherwise $r$ would be a common element of $T'$ and $U'$ – but those are collected in $V$. □

Mapping soft constraint problems to $\overline{V}$ instead of $V_\infty$ (for some $V \subseteq \mathbb{R}_{\geq 0}$ with $0 \in V$) is straightforward, as Table 7.1 shows. As depicted in Figure 7.3, define $\overline{\tau} : \mathbb{R}_{\geq 0} \to |\overline{V}|$ by $\overline{\tau}(r) = \wr r \int$ for $r \neq 0$ and $\overline{\tau}(0) = \wr\int$; then $\overline{\tau} : V_\infty \to \overline{V}$ is a PVS homomorphism and we can convert any set of $V_\infty$-soft constraints $C_{\mathrm{s}}^{V_\infty}$ into a set of $\overline{V}$-soft constraints $C_{\mathrm{s}}^{\overline{V}}$ by defining $C_{\mathrm{s}}^{\overline{V}} = \{\overline{\tau} \circ \mu_\infty \mid \mu_\infty \in C_{\mathrm{s}}^{V_\infty}\}$. As a preparatory step for optima simulation, we instantiate Lemma 7.4 for this situation. Let $r, s \in \mathbb{R}_{\geq 0}$ with $r < s$, and let $n \geq 1$. If $r = 0$, then $\overline{\tau}(s) = \wr s \int \sqsubset \wr\int = \overline{\tau}(r)^{(n)_{\overline{\mathrm{R}}}}$. If $r \neq 0$, then there is a $q > 0$ such that $s > n^{\frac{1}{p}} \cdot r$ for all $p > q$, and hence $\overline{\tau}(s) = \wr s \int \sqsubset \wr n \cdot r \int = \overline{\tau}(r)^{(n)_{\overline{V}}}$. Thus (∗∗) holds for all $n \geq 1$.

However, a soft constraint problem that is admissible when working in $V_\infty$ is not necessarily admissible when transferring the problem to $\overline{V}$. Let $X = \{x\}$, $D = [0, 1]$ with two soft constraints $\mu$ and $\nu$ mapping to $V = [0, 1]$. We define $\mu : [X \to D] \to \mathbb{R}_{\geq 0}$ with $\mu(\{x \mapsto r\}) = r$ if $r \neq 0$, and $\mu(\{x \mapsto 0\}) = 1$; and $\nu : [X \to D] \to \mathbb{R}_{\geq 0}$ with $\nu(\{x \mapsto r\}) = 1$, i.e., in total, every assignment $\{x \mapsto r\}$ is mapped to $\max\{1, r\} = 1$ in aggregation. Then $C_{\mathrm{s}}^{V_\infty} = \{\mu, \nu\}$ is a finite set of $V_\infty$-soft constraints, which is also admissible, since its set of optimal solution degrees $(C_{\mathrm{s}}^{V_\infty})^* = \{1\}$ and $C_{\mathrm{s}}^{V_\infty}(\theta) = 1$ for all $\theta \in [X \to D]$. However, for $C_{\mathrm{s}}^{\overline{V}} = \{\overline{\tau} \circ \mu, \overline{\tau} \circ \nu\}$, which is a finite set of $\overline{V}$-soft constraints, we get that $(C_{\mathrm{s}}^{\overline{V}})^* = \emptyset$, since each $\theta_r = \{x \mapsto r\}$ with $C_{\mathrm{s}}^{\overline{V}}(\theta_r) = \wr r, 1 \int$ can be improved to some $\theta_{\frac{r}{2}} = \{x \mapsto \frac{r}{2}\}$ with $C_{\mathrm{s}}^{\overline{V}}(\theta_{\frac{r}{2}}) = \wr r/2, 1 \int$ and there is no $\theta \in [X \to D]$ with $C_{\mathrm{s}}^{\overline{V}}(\theta) = \wr 1 \int$. □

Hence, to obtain an optima-simulating set of soft constraints, we need to explicitly assume admissibility. Wrapping up all the definitions for the second instance of optima-simulation, we get:

**Lemma 7.9.** *Let $X$ be a set of variables with domains $D$, $V \subseteq \mathbb{R}_{\geq 0}$ with $0 \in V$, and $C_{\mathrm{s}}^{V_\infty}$ a finite admissible set of $V_\infty$-soft constraints (from $X$ to $D$). Define the finite set of $\overline{V}$-soft constraints by $C_{\mathrm{s}}^{\overline{V}}$ by $C_{\mathrm{s}}^{\overline{V}} = \{\overline{\tau} \circ \mu \mid \mu \in C_{\mathrm{s}}^{V_\infty}\}$. If $C_{\mathrm{s}}^{\overline{V}}$ is admissible, then $C_{\mathrm{s}}^{V_\infty} \preccurlyeq C_{\mathrm{s}}^{\overline{V}}$.*

*Proof.* We proceed by the alternate sufficient criterion provided by Lemma 7.6. Hence, we have to show that $C_{\mathrm{s}}^{V_\infty}(\theta) <_{V_\infty} C_{\mathrm{s}}^{V_\infty}(\theta')$ implies $C_{\mathrm{s}}^{\overline{V}}(\theta) \sqsubset C_{\mathrm{s}}^{\overline{V}}(\theta')$ and conversely $C_{\mathrm{s}}^{\overline{V}}(\theta) \sqsubseteq C_{\mathrm{s}}^{\overline{V}}(\theta')$ implies $C_{\mathrm{s}}^{V_\infty}(\theta) \leq_{V_\infty} C_{\mathrm{s}}^{V_\infty}(\theta')$.

Therefore, let $C_{\mathrm{s}}^{V_\infty}(\theta) <_{V_\infty} C_{\mathrm{s}}^{V_\infty}(\theta')$, i.e., $r = \max\{\mu(\theta) \mid \mu \in C_{\mathrm{s}}^{V_\infty}\} > \max\{\mu(\theta') \mid \mu \in C_{\mathrm{s}}^{V_\infty}\} = s$. If $s = 0$ then $C_{\mathrm{s}}^{\overline{V}}(\theta') = \wr\int$ and $C_{\mathrm{s}}^{\overline{V}}(\theta) \sqsubset C_{\mathrm{s}}^{\overline{V}}(\theta')$ holds. Otherwise, $C_{\mathrm{s}}^{\overline{V}}(\theta) = R = \wr r_1, \ldots, r, \ldots, r_n \int$ and $C_{\mathrm{s}}^{\overline{V}}(\theta') = S = \wr s_1, \ldots, s, \ldots, s_m \int$, where $r_i \leq r$, $s_i \leq s$, and $r > s$. Then firstly $\prod_p R \geq r$ holds for every $p > 0$ due to the strict monotonicity of $\cdot_p$. Moreover, for large $p$, $r$ is greater than even $|C_{\mathrm{s}}^{V_\infty}|$ copies of $s$ since $\prod_p \wr s \mid 1 \leq i \leq |C_{\mathrm{s}}^{V_\infty}| \int = |C_{\mathrm{s}}^{V_\infty}|^{1/p} \cdot s$ and the

term $|C_s^{V_\infty}|^{1/p}$ decreases as we increase $p$. Still, the $p$-product of $|C_s^{V_\infty}|$ copies of $s$ is greater than or equal to the $p$-product of all valuations in $\{\mu(\theta') \mid \mu \in C_s^{V_\infty}\}$. Therefore, choose a $q > 0$ such that $r > |C_s^{V_\infty}|^{1/q} s$. Then $\prod_p C_s^{\overline{V}}(\theta) \geq r > |C_s^{V_\infty}|^{1/p} s \geq \prod_p C_s^{\overline{V}}(\theta')$ for all $p > q$, i.e., $C_s^{\overline{V}}(\theta) \sqsubset C_s^{\overline{V}}(\theta')$.

Now let conversely $C_s^{\overline{V}}(\theta) \sqsubseteq C_s^{\overline{V}}(\theta')$, where we have $T = \wr \mu(\theta) \mid \mu \in C_s^{V_\infty},\ \mu(\theta) \neq 0 \int$ and $T' = \wr \mu(\theta') \mid \mu \in C_s^{V_\infty},\ \mu(\theta') \neq 0 \int$. Then $\prod_p T \geq \prod_p T'$ for all $p > q$ for some $q > 0$. Since $\lim_{p\to\infty} T = \max T$ and $\lim_{p\to\infty} \prod_p T' = \max T'$, we obtain $C_s^{V_\infty}(\theta) = \max T \geq \max T' = C_s^{V_\infty}(\theta')$ and consequently $C_s^{V_\infty}(\theta) \leq_{V_\infty} C_s^{V_\infty}(\theta')$. □

In practice, $\overline{V}$ acts as a PVS collecting all soft constraints' valuations and then aggregates them in an ordering that respects the ordering induced by the maximal valuation. In a sense, the combination operation is deferred to the check for comparability of two solutions since all information is stored in the multisets.

## 7.5 Discussion: Applicability and Consequences

The driving theme throughout this chapter was to theoretically investigate the boundaries of expressing constraint hierarchies as PVS, in particular as lexicographic products of PVS representing individual layers. We have seen that any PVS directly representing maximal violation as aggregation strategy leads to collapsing elements that are prohibitive for lexicographic products. Hence, we defined and motivated optima-simulation as a criterion that formalizes approximation to a true lexicographic product for that and many other cases.

Finally, we have seen two specific optima-simulating PVS for the maximal violation case (or "worst-case-better"), one relying on $p$-norms and one building upon finite multisets of reals as the element type. These constructions give a straightforward "recipe" to apply in modeling situations involving lexicographic products (including but not limited to constraint hierarchies):

1. Define $X$, $D$ and $C$ for the (hard) constraint model.
2. Model layers of lexicographically ordered preferences using several PVS $M_1, \ldots, M_k$ with appropriate soft constraints mapping from $[X \to D]$.
3. Look for collapsing elements in $M_1, \ldots, M_k$.
4. If there are collapsing elements in PVS $M_i$, find an optima-simulating $M_i'$ along with a PVS homomorphism $\tau : M_i \to M_i'$ that can be used to "forward" soft constraints from $M_i$ to $M_i'$.
5. Solve the *SCSP* for the PVS $M_1 \ltimes \ldots \ltimes M_i' \ltimes \ldots \ltimes M_k$ and be safe to find optimal solutions that are originally optimal, i.e., if $M_i$ took place in the ordering.

These constructions are mathematically sound and we hope that, in particular, the notion of optima-simulation is useful when it comes to systematically replacing PVS from a modeler's perspective. For a purely PVS-based soft constraint framework, we argue that such constructions are essential if one wants to offer lexicographic combinations. Still, questions of practicality and actual implementations of both of the presented optima-simulating PVS remain open. The right choice obviously depends on the modeled problem in question.

Choosing the $p$-norm based $V_p$ instead of $V_\infty$ is feasible if $V$ is $\delta$-separable but leads to a scalar solution degree type and the standard ordering on reals. A caveat is that the required $p$ can increase drastically with rising $n$ and decreasing $\delta$. Evaluating large $p$-norms (on the order

Table 7.2: The magnitude of $p$ needed for $V_p$ to be optima-simulating $V_\infty$ is a function of the number of soft constraints as well as the $\delta$ that separates $V \subseteq \mathbb{R}_{\geq 0}$. Here are some numeric values to gain some intuition about the growth of $p$. We compare the influence of $n$ soft constraints and $k$ as the maximum for sets $V = \{0, \ldots, k\} \subseteq \mathbb{N}$ which are $\frac{k}{k-1}$-separated.

| | $k = 10$ | $k = 50$ | $k = 100$ | $k = 500$ | $k = 1000$ |
|---|---|---|---|---|---|
| $n = 5$ | 16 | 80 | 161 | 804 | 1609 |
| $n = 10$ | 22 | 114 | 230 | 1151 | 2302 |
| $n = 20$ | 29 | 149 | 299 | 1497 | 2995 |
| $n = 50$ | 38 | 194 | 390 | 1955 | 3911 |
| $n = 100$ | 44 | 228 | 459 | 2301 | 4603 |

of 1000s or even more, see Table 7.2) may lead to numerical issues due to possible overflows since intermediate terms $x_i^p$ need to be calculated. There are numerical algorithms dealing with well-conditioned solutions even for large $p$ [Higham, 1992; Ge et al., 2011].

Choosing the real-valued multisets $\overline{V}$ instead of $V_\infty$ on the other hand is *not* restricted to $\delta$-separable sets $V$. However, for this method to work, solvers must support more complex types than just scalars and be able to define custom orderings such that $\sqsubseteq$ can be decided. In contrast to $V_p$ that targets numerical optimizing technology, perhaps constraint-programming-based solvers are more amenable to this solution if they are able to efficiently encode multisets (perhaps similar to the way we encoded the free PVS over a partial order in MiniZinc in Section 4.3.2).

Hence, we recommend using the $p$-norm variant if a relatively loose (i.e., large) $\delta$-separation can be assumed. This is the case if the set $V$ contains small but well-separated values. If the required $p$ gets too large, we recommend the real-valued multiset construction instead. We conclude by recalling that for the MiniBrass encoding approach described in Chapter 4, no optima-simulation is yet needed as the underlying branch-and-bound algorithm only has to map complete assignments to a PVS' underlying set and use the lexicographic ordering to search for optima. It does not map assignments to tuples that are then combined using a lexicographic multiplication which is where the PVS axioms would be violated by collapsing elements. However, for a "pure" PVS-based soft constraint framework, these constructions are indispensable tools.

**Chapter Summary and Outlook**

In this chapter, we presented underlying theoretical foundations for hierarchically layered soft constraints that play a pivotal rôle in MiniBrass. Motivated by the problem of expressing Borning's classical "constraint hierarchies" as algebraic structures (in particular PVS) we showed that the "worst-case-better" semantics inevitably leads to so-called collapsing elements that were identified by Gadducci et al. [2013] and hindered Hosobe [2009] in his c-semiring construction. To still be able to use that semantics, we introduced the concept of optima-simulation to reasonably replace PVS in soft constraint problems and offered two concrete instances for "worst-case-better". We hope that optima-simulation stipulates further research for translating other classes of well-explored soft constraint formalisms into PVS.

Chapter **8**

# Aggregating Soft Constraints by Voting

**Summary.** This chapter motivates voting operators for multi-agent optimization problems based on PVS in MiniBrass. We briefly survey the known restrictions on such endeavors imposed by social choice theory, offer first preliminary voting operators for MiniBrass, and conclude with an outlook on promising future directions. Compared to previous chapters, this one is more "speculative" in the sense that it does not yet provide definite answers but highlights the relevance of the task.

Pareto and lexicographic combinations, as presented in Section 4.4.1, are an elementary tool to combine preference relations [Andréka et al., 2002]. Yet, the lexicographic product appeals to a strict form of preference aggregation that is unsuitable for "democratic" situations where several participants' preferences should be considered to a similar extent. Hitherto, the Pareto-product is the only established concept for such situations in MiniBrass. Especially with partial orders, this can become too indecisive for practical situations. For example, assume a Pareto-ordering of twenty agents' PVS deciding on some form of schedule, e.g., a shift plan or a resource allocation. An assignment $\theta$ is deemed better if and only if *all* agents agree that it is better or at least equally good. Otherwise, two assignments end up being incomparable. In reality, it is rarely the case that all agents uniformly agree on an "is-better-relation" – at least one agent could spoil things for all others. With a rising number of agents, the number of Pareto-optimal solutions is thus likely to grow uncontrollably. Being the top choice of at least one agent already makes an option a Pareto-optimal solution. Consequently, if a small number of options opposes a larger number of participating agents, most options may tend to be Pareto-optimal. Even though computing the Pareto-front, i.e., the set of Pareto-optimal solutions, is a major goal in multi-objective optimization [Ehrgott, 2005], it is arguably not decisive enough for the above-mentioned situations.

Conversely, consider the other extreme in terms of decisiveness – reducing complex multi-agent preference information to a single numeric utility function. Assume, for instance, that students rank several seminar topics according to their desirability. Unless we force every student to rank *every* topic, we are almost doomed to introduce some kind of bias: consider a student $A$ stating, say, six preferences whereas another one, $B$, only specifies three. How should we rate a violation of $A$'s top preference in comparison with one of $B$'s top preference? If we naïvely model both students' preferences as weighted PVS instances, $A$'s top choice will get a weight of six whereas $B$'s top choice will get a weight of three. Summing them

| | |
|---|---|
| 4 voters: | hiking $\succ$ biking $\succ$ relaxing |
| 3 voters: | relaxing $\succ$ biking $\succ$ hiking |
| 2 voters: | biking $\succ$ relaxing $\succ$ hiking |

Figure 8.1: A preference profile representing the orders submitted by voters. Assume that we have to pick one choice of activity, given the respective total orders over options.

up clearly leads to a bias unfairly favoring solutions that please $A$ since the solver otherwise risks higher penalties. Alternatively, we could allocate a fixed budget $q$ to every involved agent, that proportionally distributes $q$ according to its preferences, e.g., $A$ would have to split 21 points as $\langle 6, 5, 4, 3, 2, 1\rangle$ whereas $B$ could split the same 21 weight points as follows $\langle 10, 9, 2\rangle$. In that case, the solver would rather cater to $B$ since $A$ has more options that need to share the fixed budget. Either way – without more complex orderings, we introduce possibly unwanted bias. This kind of bias is, however, present in DCOPs which focus on summed scalar utilities [Fioretto et al., 2018] which calls for extensions to the generally accepted view.

Both cases (Pareto-combinations and scalar utility) highlight the need for other tools to make social decisions. We can take inspiration from the field of *social choice theory* [Pacuit, 2012; Arrow et al., 2010] which provides mathematical means to aggregate several preference relations into one representative for the group [van der Bellen, 1976].

## 8.1 Computational Social Choice

To get a sense for the flavor of problems that are treated in social choice theory, consider a situation where nine people decide on a common weekend activity (the example is strongly inspired by Brandt et al. [2013]). The available options are hiking, biking, or just relaxing in a nearby park. Figure 8.1 illustrates how the involved people voted by giving a full ordering of all activities. Which activity should be picked?

First, we might argue that hiking should be the way to go since it is the top priority of four agents and no other option has more top-level support. This rule, called plurality or majority(-tops), is by far the most common decision rule in practical situations [Shoham and Leyton-Brown, 2008], in part because we often only elicit one pick per candidate and do not ask for the full ordering (e.g., political elections). But hiking is also the *least* preferred option for a majority of (five) people. On the other hand, we may compare activities in a pairwise, exclusive fashion. Then biking wins against hiking with 5:4 votes and it wins against relaxing with 6:3 votes – hence, we should pick biking. Finally, if we remove the option with the least top-level support iteratively until only one remains, we first eliminate biking (which puts relaxing at the top of the last two voters) and then hiking which leaves relaxing as the option of choice. To sum up, three individually reasonable voting procedures yield three different outcomes – which shows that social choice is non-trivial and requires great care in implementing it for MiniBrass.

### 8.1.1 Aggregating Preferences

Formally, the field of social choice theory is concerned with aggregating preference relations. For a set of outcomes $O$, we call $Q(O)$, $P(O)$, and $T(O)$ the sets of quasi, partial, and total orders over $O$, respectively. We denote a set of agents (or voters) as $N = \{1, \ldots, n\}$. Then, a *preference profile* $[\preceq] = (\preceq_i)_{i \in N} \in P(O)^n$ (or $Q(O)^n$, etc.) is a vector containing a preference

Voter 1: $A \succ C \succ B \succ D \succ E$
Voter 2: $B \succ D \succ E \succ A \succ C$
Voter 3: $A \succ B \succ E \succ C \succ D$
Voter 4: $E \succ B \succ C \succ D \succ A$

Figure 8.2: Another exemplary preference profile representing the total orders submitted by four voters.

relation $\preceq_i$ for every agent $i \in N$ where, as usual in this dissertation, $o \preceq_i o'$ indicates that outcome $o$ is considered *worse* than $o'$. The task of voting procedures is then to map a preference profile either to a winning option, as we did in Figure 8.1, or to another full preference relation. Both outcomes should reflect the society's joint wishes as well as possible.

**Definition 8.1 – Social Choice Function [Shoham and Leyton-Brown, 2008]**
A *social choice function* $C$ maps a preference profile $[\preceq]$ of $n$ agents to a single winning output: $C : P(O)^n \to O$.

**Definition 8.2 – Social Welfare Function [Shoham and Leyton-Brown, 2008]**
A *social welfare function* $W$ maps a preference profile $[\preceq]$ of $n$ agents to a preference relation: $W : P(O)^n \to P(O)$.

We also write $o_1 \succ_{W([\succ])} o_2$ (or simply $o_1 \succ_W o_2$ if the profile is clear) if $(o_1, o_2) \in W([\succ])$, i.e., $o_1$ is collectively preferred to $o_2$. The problems are intimately related (e.g., evidenced by the famous Gibbard-Satterthwaite theorem [Reny, 2001]) since we can use a social welfare function as a social choice function by picking a top option, or conversely, repeatedly call a social choice function to obtain a full ordering as a social welfare function. For the sake of presentation, we now restrict our attention to totally ordered social welfare functions, i.e., $W : T(O)^n \to T(O)$.

To keep the chapter self-contained, we briefly survey some of the most common social welfare functions (see [Brandt et al., 2016]) with an exemplary preference profile shown in Figure 8.2. For all welfare functions, we assume that ties are broken alphabetically (otherwise we would have to sacrifice antisymmetry) or some other tie-breaking rule is imposed.

**Example 8.1 – Specific Welfare Functions**
Consider the (abstract) set of options $O = \{A, B, C, D\}$ with the voters $N = \{1, 2, 3, 4\}$ submitting $[\preceq]$ as shown in Figure 8.2. With **majority-tops**, i.e., ranking outcomes by most top-level positions, we get $W_{\text{majority-tops}}([\preceq]) = A \succ B \succ E \succ C \succ D$. Alternatively, we could apply **Borda**-ranking, i.e., assigning 4 points to the best option, 3 to the second best, etc. for every candidate first and rank the outcomes by the highest score second. Then, we get $W_{\text{borda}}([\preceq]) = B(12) \succ A(9) \succ E(8) \succ C(6) \succ D(5)$. Finally, by picking an order of the outcomes, say $[A, B, C, D, E]$, we can perform pairwise elimination, i.e., only keeping the option that wins with a pairwise majority. This procedure is called **Condorcet** voting. For instance $A$ wins against $B$ by $2 : 2$ votes and alphabetic tie-breaking, it then wins against $C$ by

$3 : 1$, and against $D$ and $E$, again by tie-breaking. Completing the ordering with $[B, C, D, E]$, we obtain $W_{\text{condorcet}}([\preceq]) = A \succ B \succ C \succ E \succ D$.

Again, we see that the choice of welfare function has a significant effect on the resulting welfare ordering. Even worse, for Condorcet voting it is well-known that no proper ordering has to emerge, i.e., cycles are possible. There are many more paradoxical outcomes such as that Condorcet voting is not necessarily Pareto-efficient (an option is preferred to another one by all agents but it is ranked worse in the welfare ordering), or that the absence or presence of an additional option influences the welfare ranking of the other options. These and other phenomena are explained in detail by Shoham and Leyton-Brown [2008]. Hence, we see that there are a plethora of possible social welfare functions (see, e.g., [Pacuit, 2012; Arrow et al., 2010]) and a natural question to ask is "which one is the best?".

### 8.1.2 Restrictions imposed by Arrow's Theorem

To classify welfare functions, it is common in social choice theory to state the desirable properties of an unknown social welfare function $W$ as axioms and reduce the set of possible functions that fulfill all axioms. This methodology is similar to specifying algorithms using contracts and relying on the abstract type's asserted post-conditions instead of the concrete implementation.

```
// CONTRACT: Sorts a given array and returns int[] sorted
// POST: returnVal is the sorted permutation of values
int[] sort(int[] values);

// now use *any* implementation that adheres to this specification
// PRE: values.length > 0
int getMinimum(int[] values) {
  int[] sorted = sort(values);
  return sorted[0];
}
```

Each valid implementation of `sort` would result in `getMinimum([5,2,36,8]) = 2`. Similarly, we can state desirable postconditions that $W$ has to satisfy for a given preference profile $[\preceq]$. Two very natural requirements are Pareto-efficiency (PE) and independence of irrelevant alternatives (IIA). PE is a formalization of "uniformity":

**Definition 8.3 – Pareto-efficiency (PE)**
$W$ is *Pareto-efficient* if for arbitrary $o_1, o_2 \in O$, $\forall i\, o_1 \succ_i o_2$ implies that $o_1 \succ_W o_2$.

If *all* agents agree on the ranking of two outcomes, it should also be placed in the same order in the welfare ordering. Put into our optimization context, this means that if *every* voter's PVS prefers some assignment $\theta$, say a schedule, to another one $\theta'$, we should also prefer $\theta$ in the collective optimization. Unsurprisingly, a Pareto-product exactly leads to that conclusion.

A second, seemingly innocent property is that the ranking of two outcomes $o_1$ and $o_2$ should not be affected by the presence or absence of a third choice $o_3 \neq o_1 \neq o_2$. It is best illustrated by a little anecdote that is attributed to Sidney Morgenbesser [Rothe et al., 2011]:

> Morgenbesser, ordering dessert, is told by the waitress that he can choose between apple pie and blueberry pie. He orders the apple pie. Shortly thereafter, the waitress comes back and says that cherry pie is also an option; Morgenbesser says: "In that case, I'll have the blueberry pie."

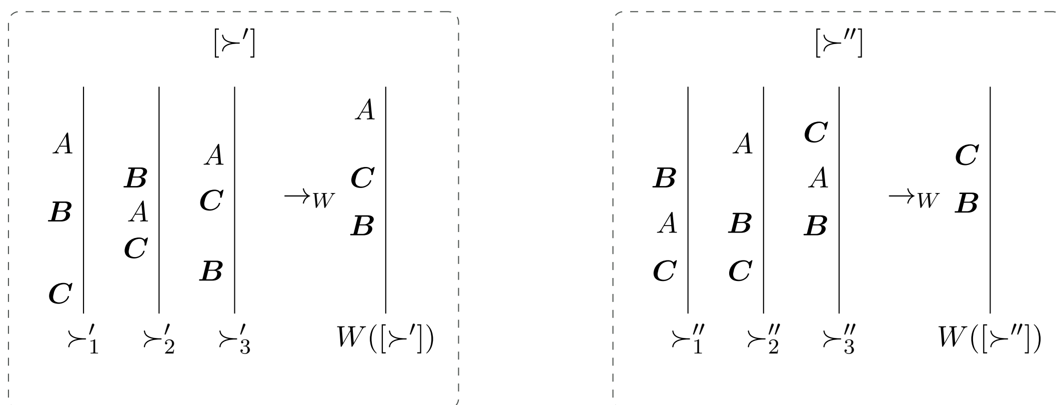



Figure 8.3: Independence of irrelevant alternatives forces some decisions in preference profiles based on similarities to other preference profiles. Here, the positions of $B$ and $C$ relative to each other are identical in $[\succ']$ and $[\succ'']$ for all agents in $N = \{1, 2, 3\}$. Therefore, since $C \succ_{W([\succ'])} B$, it must also be that $C \succ_{W([\succ''])} B$, regardless of $A$'s position in $[\succ']$ and $[\succ'']$

As intuitive as this idea is, the formal definition is a bit more involved (see Figure 8.3 for an illustration):

**Definition 8.4 – Independence of irrelevant alternatives (IIA)**
$W$ is *independent of irrelevant alternatives* if for arbitrary $o_1, o_2 \in O$ and arbitrary preference profiles $[\succ'], [\succ''] \in T(O)^n$, it holds that $\forall i\, (o_1 \succ'_i o_2 \Leftrightarrow o_1 \succ''_i o_2)$ implies that $(o_1 \succ_{W([\succ'])} o_2 \Leftrightarrow o_1 \succ_{W([\succ''])} o_2)$ holds.

Hence, relations such as $C \succ_{W([\succ''])} B$ in Figure 8.3 are enforced whenever the precondition holds in any two preference profiles – which indicates that IIA is a rather *strong* assumption despite its natural appeal.

Certainly, as a third requirement, we would never want a "social" welfare function to have a single agent forcing their preference relation as the output for *every* profile:

**Definition 8.5 – Dictatorship**
$W$ is *dictatorial* if there is an $i \in N$ such that for every preference profile $[\succ]$ it holds that $\forall o_1, o_2 (o_1 \succ_i o_2 \Rightarrow o_1 \succ_{W([\succ])} o_2)$.

Now, Arrow [1951] proved his seminal result that we cannot have a social welfare function $W$ simultaneously satisfying PE, IIA, and non-dictatorship.

**Theorem 8.1** ([Arrow, 1951]). *Assume a set of options $O$ with $|O| \geq 3$ and a set of voters $N$ with $|N| \geq 2$. A social welfare function $W$ that is both Pareto-efficient and independent of irrelevant alternatives must be dictatorial.*

Shoham and Leyton-Brown [2008] provide an accessible constructive proof of the theorem. With respect to our contracts analogy,

```
// CONTRACT: Aggregates n preferences over some options from 0 to m-1 into one, respecting PE and IIA
// PRE: profile.size() = n, for each i in 0..n-1: profile.get(i).size() = m
// POST: returnVal is picked such that PE holds on all inputs (profile)
// POST: returnVal is picked such that IIA holds on all inputs (profile)
List<Integer> welfare(List<List<Integer>> profile);
```

we come to the sobering conclusion that the only possible implementation can be:

```
int dictator = ...; // one between 0 and n-1
List<Integer> welfare(List<List<Integer>> profile) {
  return profile.get(dictator);
}
```

A truly practical consequence of the theorem for our work is that, unless we indeed restrict ourselves to choosing according to a single PVS (which is quite the contrary of a social choice), we have to sacrifice either PE or IIA for our optimization to avoid dictatorship. Depending on the application at hand, either property is not enforced – most often though, PE should be kept. In multi-criteria optimization, by comparison, we are *at least* interested in Pareto-optimal solutions, not to mention deciding which solution of the Pareto-front we should eventually pick.

It is therefore disputable if IIA is truly as compelling as it seems [Brandt et al., 2016], or if it is simply too restrictive. There are many more weaker requirements in the body of literature on social choice, although here, we restrict ourselves to the presented ones.

### 8.1.3 Soft Constraints and Voting – Related Work

Soft constraints and social choice theory are a natural match since soft constraints are concerned with expressing a partial order over solutions and social choice theory offers means to aggregate multiple orderings into one. For the single-agent case in soft constraints, we extended a *CSP* $(X,D,C)$ to an *SCSP* by adding a PVS $M$ along with a set of soft constraints $C_{\mathrm{s}}^{M}$ mapping from assignments $[X \to D]$ to $|M|$ and aggregating the soft constraints' valuations by $\cdot_M$ to obtain an overall solution degree. For every assignment, $\theta$, its overall valuation is then $C_{\mathrm{s}}^{M}(\theta) = \prod_M \wr \mu(\theta) \mid \mu \in C_{\mathrm{s}}^{M} \int$ and the ordering $\leq_M$ is used to distinguish the assignments. In the multi-agent case, it is natural to assume that $n$ agents submit $n$ PVS $(M_i)_{i\in\{1,\ldots,n\}}$ with accompanying sets of soft constraints $C_{\mathrm{s}}^{M_i}$. But instead of, say, the Pareto-ordering $\leq_{M_1 \times \ldots \times M_n}$, we want to use a voting procedure to aggregate the orderings on solutions by each of the $n$ agents. Hence, in terms of voting theory, the set of outcomes $O$ maps to $[X \to D]$, we have a preference profile $[\succ]$ consisting of $n$ quasi-orderings $\prec_i$ over $[X \to D]$ by defining $\theta \prec_i \theta' \Leftrightarrow C_{\mathrm{s}}^{M_i}(\theta) \leq_M C_{\mathrm{s}}^{M_i}(\theta')$, and a social welfare function $W$ now mapping from $Q([X \to D])^n$ to $Q([X \to D])$. The goal is to find optima according to $W([\succ])$. We can also frame the multi-agent case as a social *choice* function problem where we are only interested in a "winning" assignment $C([\succ])$. It is not clear yet if there are social welfare functions that give rise to another PVS (cf. Chapter 7 for lexicographic combinations).

The idea of combining soft constraints and voting is not entirely new. Dalla Pozza et al. [2010] presented the first algorithm to solve a problem specified with $n$ c-semirings (specifically, fuzzy constraints) using sequential voting, i.e., defining a voting ordering over the variables of the *SCSP* and assigning each variable a value from its domain by means of a voting procedure. They propose to use sequential voting in order to avoid having to enumerate *all* assignments along with the preference degree as a ballot. The same authors went on to improve on the theoretical results they received in [Dalla Pozza et al., 2011]. Specifically, the relationship between voting axioms assumed for *local* voting rules (i.e., voting over a single variable's assignment) to the *global* level (i.e., the set of all assignments) were investigated. For instance,

it is a necessary but not sufficient condition that all local rules be IIA in order for the global rule to be IIA as well. It also suffices for a single local rule to be non-dictatorial for the global rule to be non-dictatorial. Although their approach is promising in terms of satisfied axioms, it suffers from the fine granularity that agents have to vote on: domain items for a single variable. This neglects the combinatorial nature of many optimization problems and leads to overly optimistic estimations on behalf of the agents when voting, as we describe in Section 8.1.4. For example, assuming that the combinations "(fish, white wine)" and "(meat, red wine)" are deemed desirable with a slight preference for fish, an agent would place their bet on "white wine" although they might end up with the least desirable option "(meat, white wine)". For sequential voting with weighted or fuzzy constraints, Pini et al. [2013] furthermore proved that strategic bribing is NP-complete. Rossi [2014] then provided a strategic overview of collective decision making using constraint technology. She brings forward the argument that

> "First, we need to allow for very general and flexible frameworks to model constraints, optimization criteria, and preferences. In this respect, we should allow for both constraint optimization as well as quantitative and qualitative preference modeling and reasoning frameworks."

This, we believe, ideally fits the description of MiniBrass in its generality. The line of research initiated by Dalla Pozza et al. [2010] is, however, the only attempt at combining soft constraints based on algebraic structures and voting, so far. We believe that, although their proposal provides a good starting point, there are important questions of practicality left out (in particular their approach is not Pareto-efficient) and a more general treatment on the level of PVS instead of specific soft constraint formalisms (weighted and fuzzy) is desirable. Since MiniBrass is currently the only modeling language capable of supporting a variety of soft constraint formalisms, it should also offer voting operators to aggregate multiple agents' PVS.

In the area of distributed constraint optimization (DCOP) the standard model is to assume a number of numeric cost functions distributed across agents (perhaps representing their individual utility) which is summed as the global "social welfare" utility [Fioretto et al., 2018]. Netzer and Meisels [2011] extended this model to "distributed social constraint optimization problems" where social welfare functions take precedence over the summation. They illustrated their hill-climbing approach with a pickup-and-delivery case study. Still, their approach calculates a single score for each assignment based on the agents' individual (numeric) valuations and is thus likely to suffer from the bias problems mentioned in the introduction to this chapter. Moreover, they assume some form of commensurability of utilities in the sense that an operator such as "maximize the unhappiest agent's value" is meaningful – if agents operate on distinct (esp. non-numeric) partial orders, this is not the case.

### 8.1.4 A Counterexample for Sequential Voting

Given the uniqueness of the sequential voting approach presented by Dalla Pozza et al. [2010] in terms of preference aggregation for multi-agent soft constraint problems, we present it in more detail. In particular, we provide a counterexample to Pareto-efficiency which arguably reduces its relevance to settings where at least Pareto-optimal solutions are sought. The sequential voting algorithm for soft CSPs is applied in two phases:[1] First, directional arc-consistency is

[1]Moreover, the authors assume that the soft constraint problem contains only binary and unary soft constraints which can be enforced by means of Cartesian products on the original variables and domains.

applied to obtain more informed unary soft constraints: Assume $\mu_x$ to be a unary and $\mu_{xy}$ to be a binary constraint with scopes $\{x\}$ and $\{x, y\}$, respectively. Both map to some PVS $M$. Then $\mu'_x(\theta) = \max_{\leq_M}\{\mu_x(\theta_{y\mapsto d}) \cdot_M \mu_{xy}(\theta_{y\mapsto d}) \mid d \in D_y\}$ where for now, we assume a total ordering $\leq_M$ that makes this operator well-defined.[2] This process is repeated until a fixed point in terms of the involved soft constraints is reached, i.e., $\mu' = \mu$ for all soft constraints $\mu$. The idea is that for conventional single-agent soft constraint problems, if the problem's constraint graph has a tree-shape, we can use directional arc consistency to perform backtrack-free search for an optimal assignment [Meseguer et al., 2006]. If, e.g., we have $X = \{x, y\}$ with $D_x = D_y = \{1, 2\}$ and a cost function PVS with soft constraints $\mu_x$ and $\mu_{xy}$ such that $\mu_x(1) = 2$ and $\mu_x(2) = 4$, as well as $\mu_{xy} = \{(1, 1) \mapsto 4, (1, 2) \mapsto 3, (2, 1) \mapsto 2, (2, 2) \mapsto 4\}$ (cf. Figure 8.4, Agent 1). Then we get $\mu'_x(1) = 5$ and $\mu'_x(2) = 6$ (where $\mu_x(d)$ is shorthand for $\mu(\{x \mapsto d, y \mapsto d', \ldots\})$).

The second phase of the sequential voting procedures consists in picking a fixed variable ordering that determines the sequence of votings of a variable's assignment. For variable $x$, the available options are $D_x$ and agents vote according to the directed arc-consistent value they obtain for a given domain value. In our previous example, we would vote for "$x \mapsto 1$" instead of "$x \mapsto 2$" since we would prefer costs of 5 to those of 6. The agents submit their preference orderings over domain items for the variable to be decided and a (local) social choice function outputs a winning domain item for the assignment. Taking, for example, majority-tops, if three of five agents prefer "$x \mapsto 1$" to "$x \mapsto 2$", they win and $x$ is indeed assigned to one. The sequential voting approach is parameterizable in the used c-semirings as well as the social choice function (e.g., majority-tops, Borda, or Condorcet, see Example 8.1).

**Example 8.2 – Sequential voting [Dalla Pozza et al., 2010] is not Pareto-efficient on the solution level**
Figure 8.4 presents an exemplary problem instance where we have a shared constraint problem definition and three sets of soft constraints each mapping to PVS (here, cost function networks with summation as aggregation and minimization as ordering) representing three agents. As the social choice function, we use majority-tops (i.e., the domain item with most top-placements wins). The agents use the directional arc-consistent version of their problems depicted in Figure 8.5 to cast votes. Assume furthermore, they agreed on the sequence $[x, y, z]$ for the sequential voting. Following the voting decisions depicted on the right side of Figure 8.5, we end up with the assignment $\theta^* = \{x \mapsto 1, y \mapsto 1, z \mapsto 2\}$ which is Pareto-dominated by $\theta' = \{x \mapsto 2, y \mapsto 1, z \mapsto 1\}$. □

In the previous example, all agents basically agreed that they would rather accept $\theta'$ than $\theta^*$ but this is not what the voting procedure (deterministically!) provided. In reality, it would be hard to justify why the (complete) solver would not propose the uniformly accepted better solution.[3] The algorithm suffers from greedy phenomena in the sense that, e.g., agent 1 places their bets on "$x \mapsto 1$" which is influenced by the prospects "$y \mapsto 2$" and "$z \mapsto 2$" for an agent-locally optimal assignment with costs of 12. Retrospectively, had agent 1 been aware of, e.g., the strong preference for "$y \mapsto 1$" issued by both other agents, agent 1 would have been more inclined to vote for "$x \mapsto 2$". But in the strict sequential approach, no choices can be reverted

[2] If necessary, we could resort to the free c-semiring presented in Section 6.4 which collects all incomparable optima in a set.

[3] If the solver were incomplete, we could at least blame the suboptimal decision on reduced search effort and capabilities.

Shared *CSP* with $X = \{x, y, z\}$ and $D_x = D_y = D_z = \{1, 2\}$

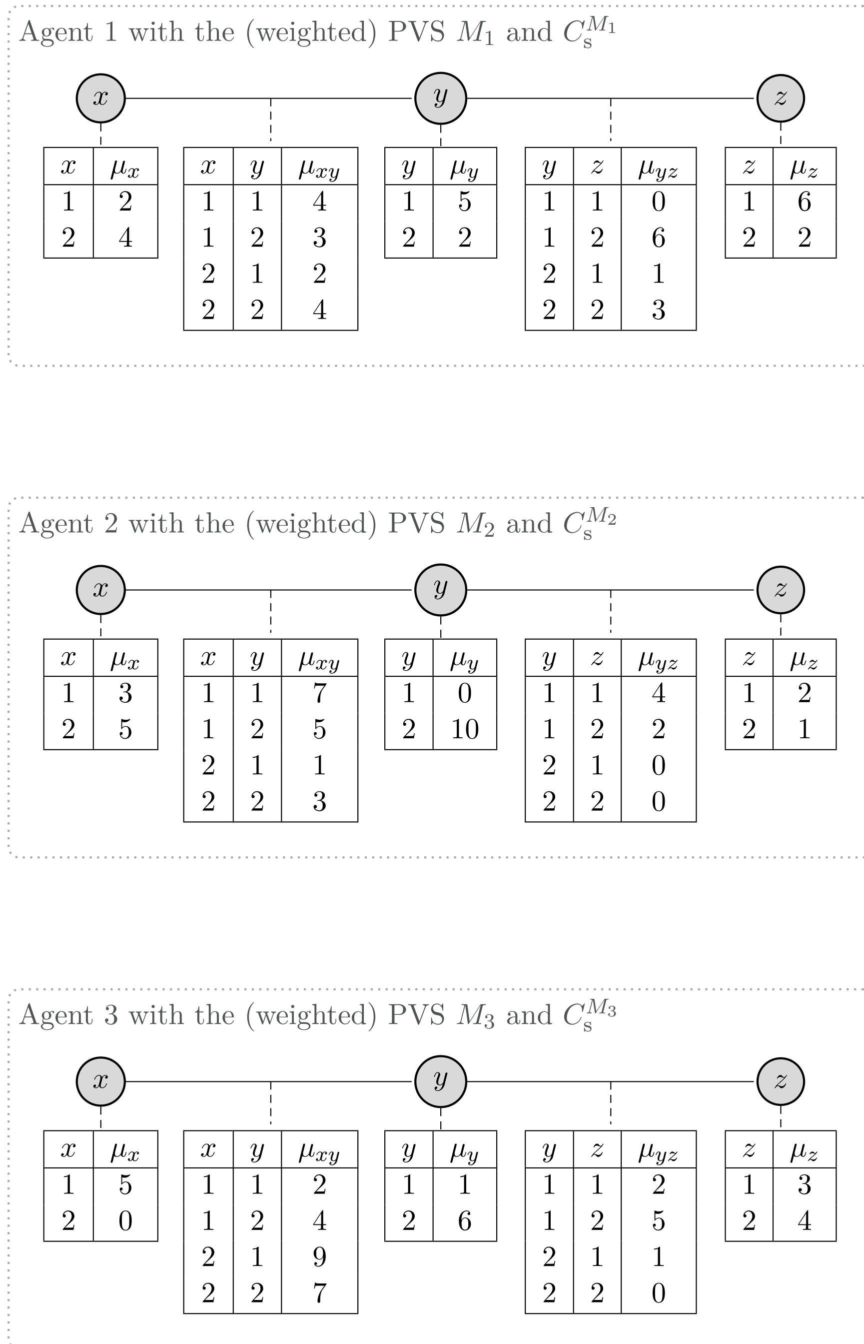
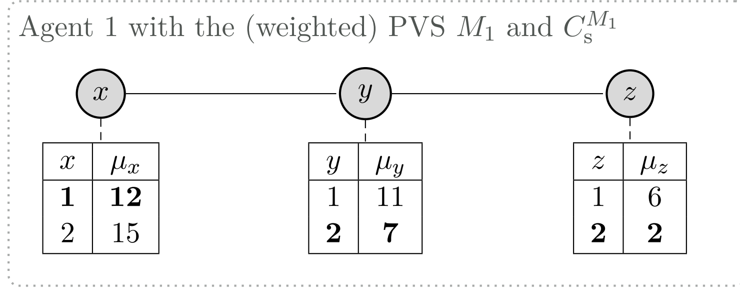


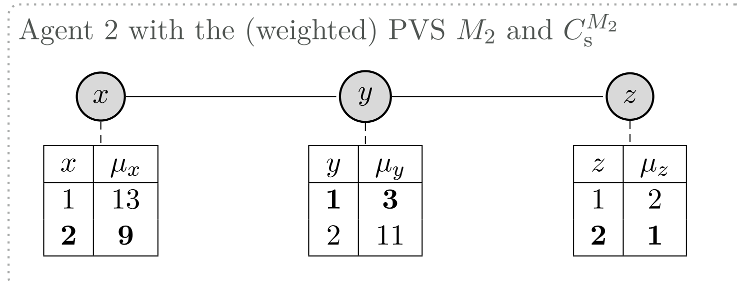

Figure 8.4: An exemplary *CSP* with three PVS representing three agents' preferences (here, cost function networks) over the shared variables $\{x, y, z\}$ and domain $\{1, 2\}$. No hard constraints are involved in this illustration.

Shared *CSP* with $X = \{x, y, z\}$ and $D_x = D_y = D_z = \{1, 2\}$



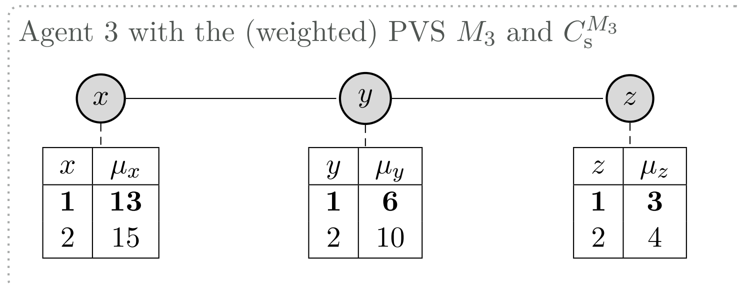

Voting the value of $x$:

| | | |
|---|---|---|
| Agent 1 | 1 | (12 vs 15) |
| Agent 2 | 2 | (9 vs 13) |
| Agent 3 | 1 | (13 vs 15) |
| **Winner** | **1** | |




Voting the value of $y$:

| | | |
|---|---|---|
| Agent 1 | 2 | (7 vs 11) |
| Agent 2 | 1 | (3 vs 11) |
| Agent 3 | 1 | (6 vs 10) |
| **Winner** | **1** | |




Voting the value of $z$:

| | | |
|---|---|---|
| Agent 1 | 2 | (2 vs 6) |
| Agent 2 | 2 | (1 vs 2) |
| Agent 3 | 1 | (3 vs 4) |
| **Winner** | **2** | |

| | x | y | z | Overall agent 1 | Overall agent 2 | Overall agent 3 |
|---|---|---|---|---|---|---|
| $\theta^*$ | 1 | 1 | 2 | 19 | 13 | 17 |
| $\theta'$ | 2 | 1 | 1 | **17** | **12** | **15** |

Figure 8.5: To cast votes, directed soft arc consistency is calculated for every agent with respect to the original problems shown in Figure 8.4. Then, a winning assignment $\theta^*$ is formed by voting on variables' assignments in the order $[x, y, z]$. For instance, for the variable $x$, agents one and three prefer $x \mapsto 1$ to $x \mapsto 2$ and agent two would rather assign $x \mapsto 2$. Hence $x \mapsto 1$ is picked due to the majority rule. Finally, the algorithm selects $\theta^* = \{x \mapsto 1, y \mapsto 1, z \mapsto 2\}$ which is Pareto-dominated by $\theta' = \{x \mapsto 2, y \mapsto 1, z \mapsto 1\}$, i.e., each agent prefers $\theta'$ to $\theta^*$ due to lower individual costs. Overall costs are given with respect to the problem in Figure 8.4.

which makes the algorithm rather inflexible and unintuitive for practical situations. As a consequence of this result, we believe that voting algorithms for multi-agent soft constraint problems should rather operate on the level of assignments $\theta \in [X \to D]$ or, alternatively, solution degrees in $|M|$ instead of domain items on isolated variables.

## 8.2 Voting in MiniBrass

Following the relationship between voting and soft constraint problems presented in Section 8.1.3, we are able to implement some aspects directly in MiniBrass by using MiniSearch and generating appropriate search predicates – as we did in Section 4.4.2 for Pareto and lexicographic products. Indeed, some social welfare functions are better suited for the combination with constraint optimization than others. For example, Borda voting would require us to enumerate all, say, $k$ solutions, rank the best solution with $k-1$, the next best with $k-2$ and so forth, for every agent. Clearly though, for discrete optimization problems, this is likely to involve an exponential number of assignments and is rather impractical as the full search space has to be enumerated before any voting can even start. On the other hand, Arrow's theorem proves relevant for any approach that exhibits some form of local search such as large-neighborhood search (see Section 3.3.3). More specifically, a social welfare function $W$ violating IIA can lead to unexpected results: If $n$ voters are confronted with a subset of the available options $O \subset [X \to D]$, e.g., a neighborhood $\mathcal{N}(o) \subseteq [X \to D]$ of an option $o$, the local ordering obtained by applying $W$ to the profile over $O$ can be different from the ordering over $O$ when the whole search space $[X \to D]$ is present. In practice, this means that even though the agents agree to switch from $o_1 \in O$ to $o_2 \in \mathcal{N}(o_2)$ when voting over the neighborhood's options, they would rank $o_1$ better than $o_2$ if all options were up for election. This fact should be considered during the design of voting procedures for MiniBrass. In summary, we ideally need welfare functions that i) do not require a full ranking on all options and ii) are independent of irrelevant alternatives (at least for local search). We propose several voting functions that offer such a useful restriction to be exploited for practical implementations.

### 8.2.1 Approval Voting

A first attempt at voting for PVS-based soft constraint problems can be achieved if we restrict the type of preference structures to "escape" Arrow's theorem. Instead of ordering solutions arbitrarily, we can allow agents to partition the search space into "acceptable" and "unacceptable" solutions, generalizing Max-CSP to multi-agent settings. Then, a solution with maximal support is picked as the winner. This procedure is known in voting theory as "approval voting" and turns out to have very beneficial theoretical properties: Approval voting is a non-dictatorial social welfare function that satisfies PE and IIA – due to the restriction to only two options, approved or disapproved; Arrow's theorem applies for votings with $|O| \geq 3$ [Shoham and Leyton-Brown, 2008].

Formally, we assume that every agent $i \in N$ declares an "approved" set with $I_i \subseteq O$ and we let $I_{i,1} = I_i$ be the first level and $I_{i,2} = O \setminus I_{i,1}$ be the second level. Interpreted order-theoretically, we have a total quasi-ordering $\succeq_i$ with $\forall o_1 \in I_{i,1}, o_2 \in I_{i,2} : o_1 \succ_i o_2$ and $\forall o_1, o_2 \in I_{i,k} : o_1 \succeq o_2 \land o_2 \succeq o_1$ for $k \in \{1, 2\}$. A preference profile $[\succ]$ in the approval setting may only consist of such approval-derived orderings. The social welfare function $W_{\text{app}}$ maps $[\succ]$ to a total quasi-ordering by counting the number of approvals any option gets. For an option $o$ and preference profile $[\succ]$, its score is denoted by $s_{[\succ]}(o) = |\{i \mid o \in I_{i,1}, i \in N\}|$. We

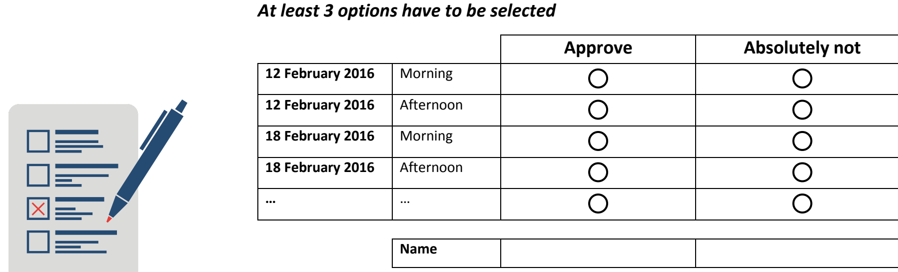



Figure 8.6: The voting "ballots" used for exam appointment scheduling presented in Example 8.3. Each ballot is translated into a boolean PVS in MiniBrass and aggregated by the `vote` operator using `approval`.

obtain the welfare ordering $\succ_{W_{\text{app}}}$ by ranking the options according to the scores $o_1 \succ_{W_{\text{app}}} o_2 \leftrightarrow s(o_1) > s(o_2)$.

**Lemma 8.2.** *Approval voting satisfies PE, IIA, and is not dictatorial.*

*Proof.* First, we begin with Pareto-efficiency: Assume there are outcomes $o_1$ and $o_2$ such that $o_1 \succ_i o_2$ for all agents $i \in N$. In the approval setting, this can only be if $o_1 \in I_i$ and $o_2 \notin I_i$ for all $i \in N$. Therefore, $s(o_1) = n > s(o_2) = 0$ and thus $o_1 \succ_{W_{\text{app}}} o_2$.

For IIA, assume two profiles $[\succ]$ and $[\succ']$ such that for two outcomes $o_1$, $o_2$, we have $o_1 \succ_i o_2$ if and only if $o_1 \succ'_i o_2$. With respect to the approval sets, this means that $o_1 \in I_i$ if and only if $o_1 \in I'_i$. Consequently, $s(o_1) = s'(o_1)$ and $s(o_2) = s'(o_2)$ which leaves us with $W_{\text{app}}([\succ]) = W_{\text{app}}([\succ]')$ – which obviously respects IIA.

Finally, we show that $W_{\text{app}}$ is not dictatorial: Assume $j \in N$ were a dictator with approval set $I_j$ in a profile $[\succ]$ and therefore $\succ_{W_{\text{app}}([\succ])} = \succ_j$ ; we can construct the preference profile $[\succ']$ where each other agent $i \in N \setminus \{j\}$ votes (opposite to the dictator) for $I_{i,1} = O \setminus I_{j,1}$. If $|N| = 2$ then we get equality for each option since $\forall o \in O : s'(o) = 1$ (the "dictator" $j$ is not able to strictly enforce their preferences anymore). For $|N| > 2$, the agents in $N \setminus \{j\}$ are able to outvote $j$ since for all $o_1 \in I_{j,1}$, we have $s'(o_1) = 1$ and for all $o_2 \in I_{j,2}$, we have $s'(o_2) = |N| - 1 > 1$. □

Again, the caveat is that the orderings degenerate to two levels of satisfaction which impedes the generality but preserves good voting-theoretic properties since Arrow is only relevant for three or more options. Moreover, the generation of an optimization predicate (or even numeric objective in this case) is straightforward. For a given assignment $\theta$, we simply count the number of approving agents. There are many use cases that can be formulated in terms of approval voting: Either agents get their preferred situation or they do not.

**Example 8.3 – Exam appointment scheduling**
A frequently occurring problem in a university is that of assigning oral exam appointments to students (see Section 2.3.1). There is a limited number of students that can take their exam on any given date. Usually, the problem is solved by using the first-come-first-served principle

on a written sheet of paper. Clearly though, students may have preferences regarding their schedules – some like to take an exam in the morning, others want to avoid collisions with other exams, some want to take it early, or some at the latest possible date. To improve upon first-come-first-served, we can have students declare preferred dates and attempt to assign each student one of those dates while respecting capacity constraints regarding maximal students per date. In addition to the mere approval set, we could allow students to also state a (reasonably small) “impossible” set – a benefit from having a constraint model underneath the voting (not depicted here):

```
% Exam scheduling example with restricted capacities
int: d; % maximum number of dates
set of int: DATE = 1..d;

int: minPerDate;
int: maxPerDate;

int: s; % number of students
set of int: STUDENT = 1..s;

% the actual decisions
array[STUDENT] of var DATE: scheduled;

constraint global_cardinality_low_up ( scheduled, [d | d in DATE],
                                                  [minPerDate | d in DATE],
                                                  [maxPerDate | d in DATE]);

var DATE: scheduledDates;
constraint  nvalue(scheduledDates, scheduled);

% redundant constraint
int: lowerBound = (s div maxPerDate) + ( s mod maxPerDate > 0) * 1  ;
constraint scheduledDates >= lowerBound;

solve search miniBrass();
```

On top of the constraint model, we can collect several PVS representing the students’ preferences and aggregate them using a `vote` statement in MiniBrass:

```
PVS: marc = new BooleanPvs("marc") {
  soft-constraint c1: 'scheduled[marc] = tueMorning';
};

PVS: andre = new BooleanPvs("andre") {
  soft-constraint c1: 'scheduled[andre] = monMorning';
};

PVS: tom = new BooleanPvs("tom") {
  soft-constraint c1: 'scheduled[tom] in {monEvening, tueMorning}';
};

solve vote([marc,andre,tom], approval);
```

In practice, this model has been used for a lecture in self-organizing systems in two terms, assigning 30 – 35 students to at most twelve dates with four students each. Fortunately, we were able to find solutions satisfying *all* students’ wishes in under a second of runtime – despite the (theoretical) search space consisting of $35^{12} \approx 1.66 \cdot 10^{18}$ elements which is substantially reduced by propagating the students-per-date constraint that is formulated using the global cardinality constraint [Régin, 1996].

The MiniBrass implementation for the PVS-based search for a model involving a `vote` statement relies on generating an appropriate `postGetBetter` predicate (see Section 4.4.2). Comparatively speaking, once we see a solution, we must enforce (by constraint) that more agents approve of the next one. Since for approval voting, we know that the type of every

| | |
|---|---|
| 35 voters: | $A \succ C \succ B$ |
| 33 voters: | $B \succ A \succ C$ |
| 32 voters: | $C \succ B \succ A$ |

Figure 8.7: A preference profile that leads to different outcomes in Condorcet voting depending on the elimination ordering. Example taken from [Shoham and Leyton-Brown, 2008].

agents' PVS must be boolean, this is straightforward, as the following code shows:

```
predicate postGetBetterApproval(array[1..N] of var bool: overallAgents) =
   post(
     sum(i in 1..N) (bool2int(overallAgents[i])) >
       sum(i in 1..N) (bool2int(sol(overallAgents[i])))
   );
```

where `sol(`*MZN*`)` represents the value of the MiniZinc expression *MZN* at the current solution and `bool2int` converts *true* to 1 and *false* to 0. This generated MiniZinc predicate is analogous to those generated for other MiniBrass concepts and can be used with the MiniBrass library "PVS-based search". Here, we can even formulate a numeric objective

```
var int: topLevelObjective = sum(i in 1..nScs) (bool2int(overallAgents[i]));
```

to use approval voting with MiniZinc only (instead of MiniSearch).

Compared to conventional PVS-based models, approval voting loses its generality due to the focus on boolean PVS. A simple extension to arbitrary PVS $M$ is *majority-tops* where we count a solution as approved if and only if it achieves the top grading $\varepsilon_M$ and mark it disapproved otherwise. We then optimize according to the number of agents that get their best possible outcome. At first, this seems too restrictive for, e.g., cost function networks or fuzzy constraints where we would count the number of 0 costs and 1.0 satisfaction degrees, respectively. However, for violation-based formalisms such as Max-CSP or constraint preferences, we would then simply count the number of agents that get all their wishes satisfied, i.e., obtain a violation set of $\emptyset$. In MiniBrass, we write, e.g.:

```
solve vote([marc,andre,tom], majorityTops);
```

Still, we search for social welfare functions that support more general orderings.

### 8.2.2 Condorcet Voting

It turns out that Condorcet-voting, also known as pairwise elimination, can be easily integrated into a constraint-based search and optimization algorithm. Recall from Example 8.1 that Condorcet-voting proceeds by fixing an ordering of the outcomes $O$, compares two adjacent options with respect to the pairwise majority, keeps the winning outcome, and returns the last remaining outcome. Figure 8.7 provides an example. Assuming, we fix the order $[A, B, C]$. Then $B$ wins against $A$ with $65 : 35$ votes and goes on to compete against $C$ where now $C$ wins with $67 : 33$ votes. Note, however, that $C$ would lose again in a direct competition against $A$ – which already highlights one of the major weaknesses of Condorcet-voting: the possibility of Condorcet-cycles. Conversely, if one option fulfills the Condorcet-criterion (winning a pairwise competition against every other option), it is called the Condorcet-winner. This voting procedure is guaranteed to find the Condorcet-winner.

Moreover, Condorcet-voting lends itself towards a useful implementation in MiniBrass on top of MiniSearch since only two options (i.e., solutions) need to be compared. Expressed in terms of constraint-based search we can proceed as follows:

1. Find the next solution (call it $\theta$).
2. Once you have $\theta$, impose a constraint that enforces that the *next* solution $\theta'$ must be preferred by a *majority* of voters and search for the next solution.
3. Repeat until no such solution can be found (or a cycle is detected).

This procedure can again be implemented using the generic PVS-based search in MiniBrass and an appropriate `postGetBetter` MiniZinc procedure. As long as the individual agents' `is_worse` predicates can be reified (i.e., have a boolean variable take its truth value which requires the predicate not to have free variables, see Section 4.4.2), this is possible, as the following pseudocode shows ($M_i$ refers to the specific PVS element type of PVS $i$).

```
predicate postGetBetterCondorcet(array[1..N] of var M_i : overall) =
  % M_i represents the PVS of agent i
   post(
     % the number of agents that find the current solution worse than the next
5    sum(i in 1..N) (bool2int(is_worse_i(sol(overall[i]), overall[i] ) )
     >
     % the number of agents that find the next solution worse than the current one
     sum(i in 1..N) (bool2int(is_worse_i(overall[i], sol(overall[i]) ) )
   );
```

If a solution is indeed a Condorcet-winner, the generic MiniSearch procedure instantiated with the above predicate will find it. If there is a Condorcet-cycle, there is no guarantee with respect to the winning outcome since the moment of termination depends on the ordering of the solutions resulting from the solver's output stream. However, such a cycle is easily detected by inspecting the trace of solutions.

**Example 8.4 – Lunch Deselection**

Establishing a meal plan is obviously "a matter of taste". Assume the participants of an annual retreat are rather pleased with the overall plan but want to change only one or two meals – at most. They can decide by Condorcet-voting which meal should be deselected using whatever specific PVS formalism they feel most comfortable with since the result is based only on individual "`isBetter`/`isWorse`" decisions.

```
% Lunch deselection problem with hard constraints regulating how many
% dishes be kept etc.
int: f;
set of int: FOOD = 1..f;
5 %[...]

% the actual decisions
var set of FOOD: selected;
array[FOOD] of var bool: selectedBool;
10 var FOOD: deselected;

constraint not(deselected in selected);
constraint link_set_to_booleans(selected, selectedBool);
constraint sum(s in FOOD)(bool2int(selectedBool[s])) == f-1;
15
solve search miniBrass();
```

On top of that constraint model, we again collect several PVS representing the voters' nutritional preferences and aggregate them using a `vote` statement parameterized by Condorcet:

```
PVS: a = new ConstraintPreferences("a") {
   soft-constraint keepGoulash: 'goulash in selected';
```

| | goulash | pork roast | bolognese | BBQ | w. salad | burger |
|---|---|---|---|---|---|---|
| goulash | – | 5 / 5 / 4 | 2 / 6 / 6 | 4 / 4 / 6 | 6 / 5 / 3 | 4 / 5 / 5 |
| pork roast | 4 / 5 / 5 | – | 3 / 6 / 5 | 1 / 7 / 6 | 8 / 4 / 2 | 4 / 4 / 6 |
| bolognese | 6 / 6 / 2 | 5 / 6 / 3 | – | 3 / 6 / 5 | 8 / 2 / 4 | 4 / 5 / 5 |
| BBQ | 6 / 4 / 4 | 6 / 7 / 1 | 5 / 6 / 3 | – | 7 / 4 / 3 | 5 / 6 / 3 |
| w. salad | 3 / 5 / 6 | 2 / 4 / 8 | 4 / 2 / 8 | 3 / 4 / 7 | – | 3 / 4 / 7 |
| burger | 5 / 5 / 4 | 6 / 4 / 4 | 5 / 5 / 4 | 3 / 6 / 5 | 7 / 4 / 3 | – |

Figure 8.8: A matrix visualizing the pairwise comparisons between solutions to the lunch deselection problem introduced in Example 8.4. Solutions refer to the value of `deselected` and a label "$x$ / $y$ / $z$" indicates that the row solution won $x$ times, there were $y$ indifferences, and the column solution won $z$ times. For instance, when choosing between "deselect pork roast" and "deselect goulash", four people voted to deselect pork roast, five were indifferent, and five would rather deselect goulash.

```
    soft-constraint keepPorkRoast: 'porkRoast in selected';
    soft-constraint keepBBQ: 'bbq in selected';
5
    crEdges : '[| mbr.keepBBQ, mbr.keepPorkRoast | mbr.keepGoulash, mbr.keepPorkRoast |]';
    useSPD: 'false' ;
};
PVS: b = new MaxCsp("b") {
10  soft-constraint c1: 'deselected = bolognese';
};
solve vote([a,b], condorcet);
```

Note how the PVS type (`ConstraintPreferences` or `MaxCsp`) is not relevant for Condorcet-voting. In practice, this problem has been used for the 2018 retreat involving 14 voters with the result depicted in Figure 8.8. Interestingly, there was a clear Condorcet-winner in terms of being deselected, "BBQ", and a Condorcet-loser in terms of being deselected (i.e., an option that all agents basically agreed to keep), "wurst salad" (sausage salad). Due to interventions outside the voting procedure, BBQ was eventually saved from being eliminated.

Results of Condorcet-voting can be visualized in an appealing way by drawing a voting "heatmap", i.e., a Matrix that shows the results of a duel between two options (here, solutions). This can provide additional insight into decision making without hiding details behind a numeric utility function. Figure 8.8 provides an instance for the aforementioned lunch deselection example. MiniBrass offers a fully automated Condorcet-analysis. Clearly, such an analysis only makes sense if the sets of solutions are manageable but for small problems it offers more insight into informed group decision making.

It is well known that Condorcet-voting is not Pareto-efficient if there is no Condorcet-

winner [Shoham and Leyton-Brown, 2008] – just as the sequential voting by Dalla Pozza et al. [2010] is not. However, in MiniBrass we can always enforce Pareto-efficiency by completing the voting procedure with a search for a Pareto-dominating solution, i.e., one that is weakly preferred by all agents and strictly preferred by at least one agent. The relevant search predicate is simply that obtained by a Pareto-product of the involved voting PVS (see Section 4.4.1).

**Chapter Summary and Outlook**

In this chapter, we motivated the need to augment a soft constraint framework with voting algorithms for unbiased preference aggregation. This is, in particular, relevant for collaborative settings where collective decisions need to be made that (in the best case) agree with most participants' wishes – possibly without monetary transactions involved. While reviewing existing approaches at the intersection of voting theory and soft constraints, we analyzed shortcomings of the state of the art in terms of applicability and provided a counterexample to Pareto-efficiency for a sequential voting procedure. Still, the exponential nature of the outcome space proves to be a significant challenge for voting approaches. Our preliminary results address this problem by either restricting the voters' preference relations to the approval setting or turning to a very local voting rule – that of Condorcet, with all its known issues.

We expect to deepen these preliminary investigations in the future by looking for practical, yet voting-theoretically appealing procedures. For instance, a hierarchical decision-making process could move from voting over coarse directions (e.g., "restaurant", "home-cooking", or "beer garden") to more fine-grained ones ("Sushi place A" or "Sushi place B") in an iterative fashion. Alternatively, agents could propose candidate solutions for voting independently of each other to reduce the search space (and henceforth the set of possible outcomes) and then vote over the union of the proposed solutions with social welfare functions.

Chapter **9**

# Evaluation

**Summary.** To show the general applicability and relationship of MiniBrass with related work, we present empirical evaluation results on a set of standardized benchmark problems. Aspects of interest include comparing the MiniBrass encoding approach to a native WCSP solver, the influence of the selected PVS type on the performance, and the effect of a PVS-specific search heuristic.

**Publication.** The main results of this chapter are published in [Schiendorfer et al., 2018]. Moreover, some of the case studies were published in [Schiendorfer et al., 2015b], [Schiendorfer et al., 2014a], and [Kosak et al., 2016].

We begin with a general evaluation of MiniBrass on a set of benchmark problems to show the generality of the approach which is not coupled tightly to any of the motivating application scenarios. Besides the feasibility check confirming that MiniBrass is well-suited for our application scenarios, we want to evaluate the performance of solvers operating on MiniBrass models versus a dedicated soft constraint solver found in the literature. We decided to model soft constraint problems using the PVS type *constraint preferences* (see Chapter 5) and used MiniZinc benchmark problems[1] as the underlying constraint models. These are taken from several editions of the MiniZinc challenge [Stuckey et al., 2014] and represent typical application scenarios of constraint solvers such as scheduling, rostering, and planning. Moreover, the models are not hand-crafted by ourselves but are public domain – to avoid unfair bias. Optimizing according to constraint preferences requires set-based variables and compatibility with MiniSearch (see Table 9.3 for a detailed compatibility matrix). By applying morphisms to the constraint preference models, as described in Section 4.4, we obtain *weighted CSP* versions that are compatible with a much wider range of solvers including those that are able to take an integer variable or expression as their objective.

Alternatively, we could have resorted to the existing cost function networks benchmark library that also offers MiniZinc models for a tabularized encoding.[2] However, conventional constraint solvers have already been shown to be dominated on these problems by Toulbar2. Moreover, these problems only address one particular PVS. Optimizing according to the Smyth-ordering has not been addressed before. In particular, this ordering introduces some partiality

[1] `https://github.com/MiniZinc/minizinc-benchmarks`

[2] `https://carlit.toulouse.inra.fr/cgi-bin/awki.cgi/BenchmarkS` and `https://github.com/MiniZinc/minizinc-benchmarks/blob/master/proteindesign12/wcsp.mzn`

in the models which generally makes the task of finding optima more demanding due to reduced pruning.

The problems were selected according to features that justify an encoding approach (i.e., efficient conventional propagation), the feasibility of decompositions for many solvers, and meaningful soft constraint addition. Certainly, there are cost function network problems (e.g., in bioinformatics) that are out of reach for conventional solvers, as [Hurley et al., 2016] demonstrated for Toulbar2. However, we argue that there are many practical cases with relatively few soft constraints and many conventional constraints. Among those, we investigate the following problems:

**Soft N-Queens** is a toy *SCSP* that adds three artificial soft constraints with a preference relation over them to a classical $N$-queens problem (such as, e.g., having a queen in the center of the grid), not to be mistaken with the $M$-queens optimization problem.

**Photo Placement** asks to place people close to their friends – in its original version it was already designed to handle preferences but we (morally questionably) allowed for some friends to be *more important* to stand close to than others.

**Talent Scheduling** aims at scheduling movie scenes including various actors cost-effectively. We augmented the conventional problem with preferences to avoid being simultaneously on set with a rival actor and early/late times for specific scenes.

**On-call Rostering** requires to assign staff members to days in a rostering period, respecting work constraints and unavailabilities. The original formulation already contained preferences for not being on-call for more than two days in a row or not being on-call for a weekend and a consecutive day. We modeled these existing preferences in MiniBrass and added additional ones regarding preferred co-workers.

**Multi-Skilled Project Scheduling** (MSPSP) is a variant of resource-constrained project scheduling and asks to assign a set of tasks to workers such that the required set of skills for a task is provided by its assigned worker. To add soft constraints, we again allowed workers to state with whom they would like to work and which tasks they would like to work on or which ones they would rather avoid.

More detailed information about the changed and added aspects can be found online.[3] Each problem is tested with three to six instances, totaling 28 benchmark instances (Table 9.1 summarizes the key data in terms of experiment size). Since most of these problems already were formulated as constraint optimization problems we had to deal with two objectives: the original one and the soft constraint objective. First, we converted the problems to constraint satisfaction problems by imposing the original objective value to lie within a 10%–15% boundary around the (previously determined) optimal value, and eventually used the soft constraint objective. In MiniBrass, we write this as

```
solve search presolveMin(origObj, 0.15) /\ miniBrass();
```

We solved the resulting models (including parameters that will be subject to the respective experiment questions) using branch-and-bound[4] (cf. Section 4.4.2) with the classical constraint solvers Gecode 5.0.0 [Schulte et al., 2006], JaCoP 4.4.0 [Kuchcinski and Szymanek, 2013], Google OR-Tools 5.1.0 (CP solver) [Google, 2017], Choco 4.0.3 [Jussien et al., 2008], and

[3] https://github.com/isse-augsburg/minibrass/tree/master/evaluation/problems

[4] We experimented with large neighborhood search as well but did not find it to be effective enough for the selected evaluation problems.

Table 9.1: Key data that shows how many actual experiments, i.e., attempted MiniZinc runs have been pursued with various parameter/solver configurations.

| | |
|---|---|
| Configurations | 16 |
| Problems | 5 |
| Instances | 28 |
| Solvers | 7 |
| **Attempted problems** | 1793 |
| **Solved problems** | 1289 (71.9%) |
| **Optimally solved problems** | 1250 (69.7%) |

Table 9.2: Compatibility matrix of the solvers used in the evaluation based on their expressivity and tool support. The symbol ✓ indicates full compatibility, (✓) refers to partial support by morphisms to weighted CSP, $x$ means no support at all. The dedicated soft constraint solver Toulbar2 is marked by an asterisk.

| | Gecode | JaCoP | OR-Tools | Choco | G12 | Toulbar2 * |
|---|---|---|---|---|---|---|
| Free PVS | ✓ | ✓ | $x$ | $x$ | $x$ | $x$ |
| Constraint Preferences | ✓ | ✓ | $x$ | $x$ | $x$ | $x$ |
| Fuzzy CSP | ✓ | ✓ | (✓) | (✓) | (✓) | (✓) |
| Probabilistic CSP | ✓ | ✓ | (✓) | (✓) | (✓) | (✓) |
| Max CSP | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Weighted CSP | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Cost Function Networks | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

G12 1.6.0 [Stuckey et al., 2005], as well as with the only competitive cost function network solver Toulbar2 0.9.8 [Allouche et al., 2010], accessed via Numberjack 1.1.0 [Hebrard et al., 2010]. Each presented experiment was run on a machine having 4 Intel Xeon CPU 3.20 GHz cores and 14.7 GB RAM on a 64 bit Ubuntu 15.04 with a timeout set to 10 minutes per instance. Whenever we average relative values (such as a speedup for runtimes) that are normalized to one solver/heuristic, we use the *geometric mean* as the only valid choice for such data [Fleming and Wallace, 1986]. Concerning statistical tests for runtime comparisons, we used the Wilcoxon signed-rank test (alpha level $\alpha = 0.05$) since the measured runtimes showed a heavy-tailed distribution instead of a normal one.

Our primary goal is to demonstrate the *feasibility* of implementing soft constraint formalisms more generally than a numeric objective at low runtime overheads – a capability that is not shared by any state-of-the-art soft constraint solver. Besides, even in the realm of cost function networks and weighted constraints, it can pay off to use an encoding approach with a conventional constraint solver as opposed to a dedicated soft constraint solver. Furthermore, we examine the effects of our proposed search heuristics for weighted constraints.

## 9.1 Comparing Performance: Encoded Weighted CSP versus Native Toulbar2

If we want to obtain a comparative view on the performance of MiniBrass models, we have to use cost function networks. For one thing, they are the native formalism Toulbar2 supports, for another thing, the task boils down to minimizing a numeric value in conventional models which is directly supported by MiniZinc. On the one hand, Toulbar2 can be seen as the only true state-of-the-art alternative to MiniBrass (given that `WSimply` [Ansótegui et al., 2013] has no MiniZinc or Numberjack interface, only runs on a 32 bit Linux distribution, and is no longer maintained) – on the other hand it serves as a well-supported backend. Therefore, this evaluation cannot be truly seen in a competitive light as MiniBrass is a modeling language. Here, the central question is:

*How fast and effectively (in terms of finding optima) can WCSP instances be solved by encoding them as COPs versus using a dedicated solver?*

Table 9.3 presents the results for this first question with times and objectives being averaged across all instances for the respective problem, ranked by runtimes.[5] Values in parentheses denote *averaged relative values* with respect to the minimum (geometric mean of ratios for time or arithmetic mean for excess penalty violation, resp.) for each instance – as opposed to *relative average values.* Therefore, the relative overhead does not necessarily correlate with the absolute values (e.g., Toulbar2 versus Gecode on Talent Scheduling). The number of wins indicates how many instances a solver won (i.e., being fastest) within the respective problem. If an instance was not solved at all within the specified time limit, the maximally possible violation for it was assumed.

We observed a fairly even distribution of solvers performing well with OR-Tools being among the top three on all problems, showing the most reliable contribution of all conventional constraint solvers. In addition to the table, we noted that across all problems, OR-Tools had the lowest average runtime (97.39 secs) and the lowest average objective value (6.18), whereas Gecode achieved the most wins (12). Interestingly, Toulbar2 managed to achieve the best (or second best) average runtimes for three problems, excelling in On-call Rostering. However, the memory-intensive decompositions required for MSPSP and Talent Scheduling had Toulbar2 fail during model creation without returning a solution. To conclude, even though Toulbar2 is a strong choice when dealing with cost function networks, there are cases where only an encoding approach succeeded at all (MSPSP) – or substantially faster (Talent Scheduling). With problems modeled in MiniBrass, both options remain.

## 9.2 Comparing Models: Smyth-Optimization versus Weighted-Optimization

Upon learning that weighted instances can be solved efficiently by conventional constraint solvers, we turn to optimization according to the Smyth-ordering. MiniBrass was explicitly designed for more general orderings than numeric objectives – in particular, Smyth as the ordering of the free PVS. We want to quantify how expensive the partiality of an original

[5]The fact that Choco shows a higher average objective on Photo Placement albeit claiming to have proved optimality results from a bug in the solver induced by the problem-specific search heuristics.

Table 9.3: Comparison of solvers' performance on the weighted CSP representations. Values in parentheses denote *averaged relative values* with respect to the minimum (ratio for time or excess penalty violation).

| Solver | Time (secs) | | # Wins | Objective | | % Solved | % Optimal |
|---|---|---|---|---|---|---|---|
| MSPSP (8 instances) | | | | | | | |
| Gecode | 0.32 | (1.00) | 8 | 2.50 | (0.00) | 100.00 | 100.00 |
| G12 | 0.32 | (1.01) | 0 | 2.50 | (0.00) | 100.00 | 100.00 |
| OR-Tools | 0.33 | (1.04) | 0 | 2.50 | (0.00) | 100.00 | 100.00 |
| JaCoP | 0.52 | (1.71) | 0 | 2.50 | (0.00) | 100.00 | 100.00 |
| Choco | 0.70 | (2.40) | 0 | 2.50 | (0.00) | 100.00 | 100.00 |
| Toulbar2 | 312.56 | (654.69) | 0 | 29.13 | (26.63) | 0.00 | 0.00 |
| On-call Rostering (7 instances) | | | | | | | |
| Toulbar2 | 40.73 | (1.28) | 3 | 1.57 | (0.00) | 100.00 | 100.00 |
| OR-Tools | 275.23 | (2.66) | 2 | 3.71 | (2.14) | 100.00 | 57.14 |
| Gecode | 275.23 | (2.64) | 1 | 4.57 | (3.00) | 100.00 | 57.14 |
| G12 | 276.36 | (2.84) | 1 | 5.57 | (4.00) | 100.00 | 57.14 |
| JaCoP | 276.63 | (3.27) | 0 | 5.14 | (3.57) | 100.00 | 57.14 |
| Choco | 276.72 | (3.82) | 0 | 5.14 | (3.57) | 100.00 | 57.14 |
| Photo Placement (3 instances) | | | | | | | |
| Toulbar2 | 0.80 | (1.11) | 0 | 13.33 | (0.00) | 100.00 | 100.00 |
| Choco | 0.83 | (1.18) | 2 | 25.00 | (11.67) | 100.00 | 100.00 |
| OR-Tools | 1.49 | (1.51) | 1 | 13.33 | (0.00) | 100.00 | 100.00 |
| JaCoP | 3.18 | (3.08) | 0 | 13.33 | (0.00) | 100.00 | 100.00 |
| Gecode | 22.24 | (5.11) | 0 | 13.33 | (0.00) | 100.00 | 100.00 |
| G12 | 27.40 | (23.06) | 0 | 13.33 | (0.00) | 100.00 | 100.00 |
| Soft N-Queens (3 instances) | | | | | | | |
| OR-Tools | 0.03 | (1.00) | 3 | 0.33 | (0.00) | 100.00 | 100.00 |
| Toulbar2 | 0.30 | (9.96) | 0 | 0.33 | (0.00) | 100.00 | 100.00 |
| Choco | 0.35 | (12.51) | 0 | 0.33 | (0.00) | 100.00 | 100.00 |
| JaCoP | 57.22 | (56.00) | 0 | 0.33 | (0.00) | 100.00 | 100.00 |
| Gecode | 210.02 | (29.31) | 0 | 1.67 | (1.33) | 100.00 | 66.67 |
| G12 | 210.02 | (32.26) | 0 | 1.67 | (1.33) | 100.00 | 66.67 |
| Talent Scheduling (7 instances) | | | | | | | |
| OR-Tools | 113.29 | (1.01) | 3 | 12.29 | (0.00) | 100.00 | 85.71 |
| JaCoP | 117.71 | (1.61) | 0 | 12.29 | (0.00) | 100.00 | 85.71 |
| Choco | 129.12 | (2.55) | 1 | 12.29 | (0.00) | 100.00 | 85.71 |
| Toulbar2 | 158.27 | (9.29) | 0 | 28.43 | (16.14) | 28.57 | 28.57 |
| Gecode | 183.29 | (2.22) | 3 | 12.29 | (0.00) | 100.00 | 85.71 |
| G12 | 194.91 | (2.36) | 0 | 12.29 | (0.00) | 100.00 | 85.71 |

Table 9.4: Comparing the solvers' performance on a Smyth-based model and the weighted CSP representations. Times and objectives are averaged over all instances for a given problem and can be compared. We only considered *solved* instances in this evaluation. Bold-face highlighting indicates the faster model per solver. Runtimes of Toulbar2 on the weighted instances are given "out of competition" where applicable – i.e., if the decomposition succeeded on all competing instances. Times are given in seconds.

| Solver | Time Smyth | Time Weighted | Time Toulbar2 | Obj. Smyth | Obj. Weighted |
|---|---|---|---|---|---|
| MSPSP (6 instances) | | | | | |
| Gecode | 12.74 | **0.34** | - | 5.50 | 2.67 |
| Native Gecode | 7.82 | **0.26** | - | 5.80 | 2.80 |
| JaCoP | 4.18 | **0.45** | - | 6.00 | 2.00 |
| On-call Rostering (5 instances) | | | | | |
| Gecode | 220.46 | **133.32** | *14.52* | 7.20 | 3.20 |
| Native Gecode | 192.50 | **133.32** | *14.52* | 25.20 | 3.20 |
| JaCoP | 194.06 | **135.28** | *14.52* | 26.80 | 3.20 |
| Photo Placement (3 instances) | | | | | |
| Gecode | 6.69 | **1.03** | *0.68* | 13.00 | 13.00 |
| Native Gecode | **9.96** | 22.22 | *0.80* | 13.33 | 13.33 |
| JaCoP | 15.73 | **3.18** | *0.80* | 13.33 | 13.33 |
| Soft N-Queens (3 instances) | | | | | |
| Gecode | **3.45** | 210.02 | *0.30* | 2.00 | 1.67 |
| Native Gecode | **3.49** | 210.02 | *0.30* | 1.33 | 1.67 |
| JaCoP | **3.94** | 57.22 | *0.30* | 1.00 | 0.33 |
| Talent Scheduling (6 instances) | | | | | |
| Gecode | **7.78** | 158.94 | - | 14.25 | 12.50 |
| Native Gecode | **13.50** | 141.09 | - | 14.67 | 12.33 |
| JaCoP | **15.63** | 120.42 | - | 14.17 | 12.33 |

model is with respect to the totalization obtained by weighting constraints. To solve these models, only Gecode and JaCoP are applicable, as they are both compatible with MiniSearch and support set-based variables to the necessary extent. For these solvers, we compare the running times and objective values[6] for the original Smyth-based model and the (morphed) weighted CSP. Gecode is provided in a native version directly accessed by MiniSearch (see Section 3.5) and a FlatZinc-based version – with the latter being more recent than the native one. JaCoP is only available using FlatZinc. Where applicable, i.e., if Toulbar2 solved the instances, we additionally provide its reference values (Toulbar2 is restricted to the weighted version). Here, the central question is:

*Is optimizing according to the Smyth-ordering much more expensive than solving a weighted counterpart obtained by a morphism?*

Table 9.4 presents our results answering this question. Note that, for this evaluation, the Smyth-based models have been solved with strict domination BaB since this is the only way the totalized weighted version can operate. We expected the weighted problems to be much easier to solve since there is possibly stronger pruning and propagation involved. To our surprise, we noticed that, whereas for most instances (87.8%), the weighted counterpart was indeed easier

[6] Note that the "objective values" for the Smyth-model are provided only for comparative reasons. Optimization was done purely according to the Smyth-ordering on the set of violated soft constraints.

Table 9.5: Comparison of runtimes between searching for all optima instead of a strict domination improvement. We only considered *solved* instances in this evaluation. Bold-face highlighting indicates the faster search type. Values are averaged over instances and solvers. Times are given in seconds. An asterisk (*) indicates statistical significance as reported by a Wilcoxon signed-pair test at $\alpha = 0.05$.

| Problem | Time Non-Dominated BaB | Time Strict BaB | Absolute Overhead | Relative Overhead |
|---|---|---|---|---|
| MSPSP | **7.31** | 8.89 | −1.58 | 0.78 |
| On-call Rostering* | 329.44 | **199.21** | 130.23 | 1.70 |
| Photo Placement* | 55.09 | **7.51** | 47.58 | 5.32 |
| Soft N-Queens | **2.24** | 3.65 | −1.41 | 0.56 |
| Talent Scheduling* | 33.44 | **12.24** | 21.21 | 1.81 |
| *Overall* | 102.00 | **57.20** | 44.80 | 1.47 |

to solve, there were instances where the constraint preferences version took substantially less time – as in Talent Scheduling and Soft N-Queens. A possible explanation is that optimality can be easier proved using propagation of the witness function of the Smyth-ordering. Put differently, there could be better solutions in terms of weights but not Smyth, therefore search can be pruned earlier. We may also notice that, on these instances, Toulbar2 can provide much better performance than the constraint solvers on the weighted counterparts – when applicable. This is mostly due to the fact that Choco and OR-Tools are left out (as opposed to Section 9.1) since they currently do not support set variables. In terms of objective values, even though optimality is proven in most cases, the Smyth and weighted versions yield different values, which is not surprising as, again, a "weight-better" solution need not be "Smyth-better". Thus, there are generally lower values to be expected using the weighted version. The attentive reader will notice that the average objective for the Smyth-model is, in fact, *lower* than for the weighted model in Soft N-queens solved by the native Gecode solver. In fact, the solver timed out on one weighted instance at the sub-optimal objective value 4 whereas the Smyth-based variant happened to yield a Smyth-optimal solution that is also weight-optimal with objective value 2.

With strict BaB only, we only get one optimal solution – at best. The advantage of using partial orders clearly is having multiple incomparable optima at the modeling stage of the process and not having to totalize the ordering by weighting. However, searching for a whole set of optima (as done in *non-domination* BaB) obviously leads to longer runtimes than stopping at the first found optimum (as done in *strict* BaB). We investigate the differences in Table 9.5 (recall that TPD has to be used for non-domination search, see Section 4.3.2). On the examined benchmark problems, we observe a slowdown factor between one and five *when* non-domination leads to longer runtimes. However, in some cases the difference between non-domination and strict BaB was negligible (i.e., MSPSP and Soft N-Queens), even showing a small (not statistically significant) speedup for non-domination search – mostly due to the set of optima actually being small where strict and non-domination BaB converge to similar search trees.

Table 9.6: Runtime difference between models with MIF activated and deactivated. Negative values indicate that MIF led to faster solving times. Winning ratios are given with respect to the number of instances. Relative runtime differences normalize MIF to default (aggregated by the geometric mean). Solved configurations include Smyth-based and weighted models. For the actual runtimes, see Figure 9.1. Times are given in seconds. An asterisk (*) indicates statistical significance as reported by a Wilcoxon signed-pair test at $\alpha = 0.05$.

Grouped by solvers

| | Choco* | G12 | Gecode | Native Gecode | JaCoP | Toulbar2 | OR-Tools |
|---|---|---|---|---|---|---|---|
| Instances | 28 | 28 | 28 | 28 | 28 | 28 | 28 |
| Runtime difference | **−73.14** | **−17.57** | **−18.42** | **−18.53** | 16.15 | 36.63 | 19.05 |
| Rel. runtime diff. | 0.65 | 0.72 | 0.79 | 0.80 | 1.04 | 1.11 | 1.50 |
| Ratio MIF wins | **0.64** | 0.32 | 0.29 | 0.18 | 0.46 | **0.57** | 0.32 |

Grouped by problems

| | MSPSP* | On-call Rostering* | Photo Placement* | Soft N-Queens* | Talent Scheduling |
|---|---|---|---|---|---|
| Instances | 56 | 49 | 21 | 21 | 49 |
| Runtime difference | **−0.68** | **−26.63** | 145.93 | **−98.15** | **−24.96** |
| Rel. runtime diff. | 1.04 | 0.87 | 8.64 | 0.16 | 0.66 |
| Ratio MIF wins | 0.36 | **0.51** | 0.05 | **0.52** | 0.43 |

## 9.3 Comparing Search Heuristics: Most Important First versus Default

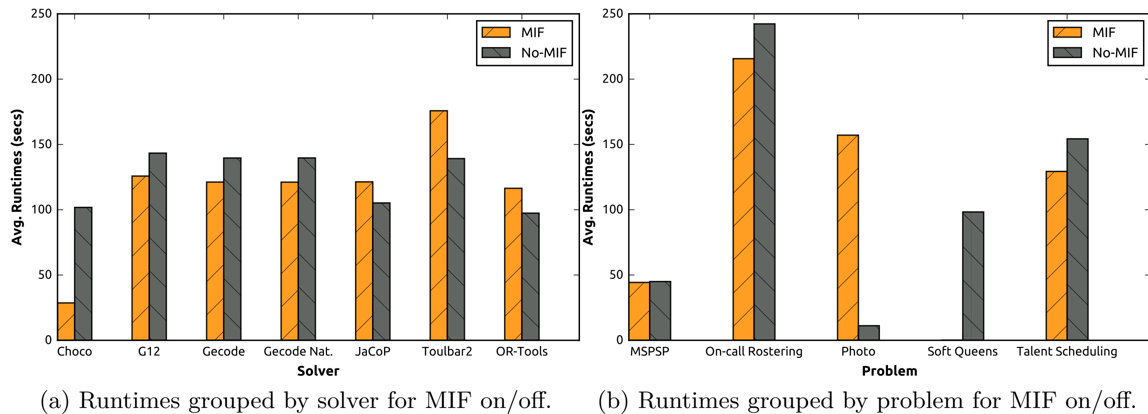
Lastly, with abstract higher level preference models, we can use generic search heuristics that align with the optimization goals – dependent on the PVS type in use. Here, our simple strategy (shown in Section 4.3.1) is to try and assign `true` to the boolean variables reifying[7] the satisfaction of soft constraints in the order of decreasing weight (i.e., importance). We refer to this heuristic as *most important first* (MIF). Some of the benchmark problems already shipped with a problem-specific variable ordering heuristic. In such cases, activated MIF prepends the reified satisfaction variables to the existing heuristic. We compare the effects of MIF on various types of searches (strict, non-domination, weighted), problems, and solvers. The central question is:

*Can a generic heuristic (MIF) for soft constraint problems speed up the search for optima?*

Over all 168 runs across solvers, problem instances, and search types, the MIF heuristic led to a faster runtime in 73 cases (43 %) with the average runtime reduced by 6.22 seconds. Yet, MIF is not uniformly beneficial but affects some solvers more than others. Similarly, some problems are more likely to be improved. Table 9.6 presents results in a more fine-grained fashion, grouping the evaluated data by problems and solvers, respectively. We find that MIF seems to negatively influence the performance compared to the built-in default search strategies in particular for OR-Tools, JaCoP, and Toulbar2 but can lead to tremendous improvements for Choco (cf. Figure 9.1a) – which showed the only statistically significant difference when grouped by solvers – due to instances of all problems being compared.

On the other hand, when grouping by problem, On-call Rostering and Talent Scheduling benefited the most from MIF (cf. Figure 9.1b), both showing a speedup on average (relative

[7] Certainly, some global constraints cannot be reified directly yet, only support half-reification or need a decomposition but we expect them to increasingly do so [Beldiceanu et al., 2013].




(a) Runtimes grouped by solver for MIF on/off. (b) Runtimes grouped by problem for MIF on/off.

Figure 9.1: Average runtimes for instances solved with and without MIF activated for several models (Smyth or weighted). Figures correspond to the data showing differences in Table 9.6.

differences) – although the difference was not significant for Talent Scheduling: over all instances, MIF produced faster but also slower runtimes about the same number of times while the average still favors MIF. Contrary to that, for On-call Rostering, the runtime difference – albeit small on average – was statistically significant since MIF improved the performance in about half of the instances – and in the others, it did no harm. We suspect that for the other problems, either the built-in heuristics were effective enough or MIF led to thrashing behavior if the best solutions still violated many soft constraints. Then, MIF initiates many "pointless" searches by setting all soft constraints to be satisfied and propagation fails to prove infeasibility fast enough. For problems such as Photo Placement or MSPSP, we admittedly better use the default search strategy. MIF clearly is no silver bullet. However, since activating a search heuristic in MiniBrass only amounts to placing `pvsSearchHeuristic` in front of the search procedure (cf. Listing 4.1), this still is an easy first step in tweaking the performance. Interestingly, our experiments also reveal that the effectiveness of the heuristic depends more strongly on the problem at hand than it does on the particular solver.

**Chapter Summary and Outlook**

This chapter provided an empirical evaluation of constraint preferences within the scope of MiniBrass. For this purpose, we converted standard constraint models in MiniZinc into soft constraint models for MiniBrass by adding suitable soft constraints and integrating the original objective. We showed that optimizing according to the Smyth-ordering is feasible and comparable to classical weighted CSP optimization. Classical constraint solvers outperform a dedicated soft constraint solver if the models contain sufficiently many hard, global constraints with efficient propagators. Nevertheless, MiniBrass offers both options as backends to choose at runtime. For constraint preferences, we also demonstrated that the most-important-first heuristic often leads to improved runtime and can be easily activated in MiniBrass.

Chapter **10**

# Conclusion and Outlook

Over-constrained optimization problems are ubiquitous not only but especially in self-organizing and autonomous systems. Despite significant theoretical advances in solid algebraic frameworks, the widespread usage of soft constraints is hindered by the lack of available implementations. This dissertation ameliorates this status by offering several contributions – most importantly, an extensible, high-level soft-constraint modeling language that can be used in combination with a variety of state-of-the-art solvers. Nonetheless, en route towards the implementation of MiniBrass, we identified several open theoretical problems and provided solutions for them.

## 10.1 Achieved Contributions

Motivated by a diverse set of application scenarios, we identified the need for a soft constraint modeling language. By introducing a qualitative formalism to specify preferences over constraints, we found that *partial valuation structures* are suitable as a generalizing algebraic structure for many preference formalisms proposed in the literature.

In the practical part of the dissertation, we presented and implemented *MiniBrass*, a soft constraint modeling language building on MiniZinc that closes the gap between algebraic soft constraint frameworks and state-of-the-art solvers. MiniBrass allows designing complex preference structures using product operators (for lexicographic and Pareto combinations) and morphisms (for type conversions) separately from conventional constraints. It is extensible by design but already incorporates many existing formalisms.

In the theoretical part, we motivated why the concept of free constructions is an appropriate tool to facilitate the transition from partial orders to PVS and from PVS to c-semirings with the least structural overhead and provided proofs. Since these constructions involve several sorts of algebraic structures, we employ the unifying language of category theory for the mathematical underpinning. Collapsing elements of PVS equalize unequal elements upon multiplication and render lexicographic products invalid. For such cases, we introduced the concept of optima-simulation. We exemplified this methodology with a $p$-norm-based PVS that can optima-simulate a PVS using the maximum operator for its aggregation. To combine the goals specified by multiple PVS in a more practically suitable way than what direct or lexicographic products offer, we suggested using social welfare functions, i.e., voting methods introduced by social choice theory. Preliminary results indicate that, despite mathematical restrictions imposed by the very nature of social choice, some welfare functions are well-suited

for implementation in constraint-based optimization tools. This “democratic” perspective is, in particular, relevant in optimization problems stemming from autonomous systems composed of multiple self-interested agents.

Finally, in the empirical part of the dissertation, we evaluated MiniBrass on a set of “softened” benchmark problems originating from the MiniZinc challenge and found that on these problems an encoding-approach is competitive with dedicated soft constraint solving, optimizing with the Smyth-ordering is only slightly more expensive than weighted problems, and our proposed most-important-first heuristic can lead to significant runtime savings. The results demonstrate that MiniBrass is not only a tool of theoretical interest but also leads to compact modeling and efficient solving, in practice.

## 10.2 Outlook and Future Work

For future work, we consider applying MiniBrass in multi-agent settings the most important direction. The more human societies are permeated by “intelligent” devices taking over decisions for people, the more important it becomes that *fair* and socially acceptable aggregation strategies are applied. We expect to achieve positive results in at least some classes of multi-agent optimization problems such as repeated task or resource allocation where techniques from fair scheduling in concurrent systems could be borrowed and applied.

Besides smart energy systems, the field of reconfigurable mobile robot systems provides manifold applications for soft constraint optimization. For instance, task and resource allocation problems for unmanned aerial vehicles (UAV) with reconfigurable sensors promise to be a rich source of challenging optimization problems under soft constraints such as task quality, frequency of reconfigurations, aptitude for tasks, etc. Kosak et al. [2016] already demonstrated that the hardware commonly installed in modern UAVs is capable of reliably solving task allocation problems specified in MiniZinc. Hanke et al. [2018] proposed a reconfiguration algorithm that equips UAVs with appropriate sensors in a decentralized fashion, based on local MiniZinc models. Soft constraints have not been considered, yet, but the more aspects we model, the more likely it is that task and resource allocation becomes over-constrained. We expect synergy effects from research conducted in that area.

On a technical note, we should be aware that there is a subtlety in the way MiniBrass commonly addresses Pareto-combinations or voting operators. The order in which the solutions are found is heavily influenced by the involved variable and value orderings shaping the search tree. Consequently, when searching a whole Pareto-front or even several incomparable optima, it certainly should not be the case that important real-life decisions depend on a solver’s specific implementation. Particular care should be taken for Condorcet-voting or similar algorithms that depend strongly on the solution ordering. Related to this issue, randomization can also be a vital part of a voting method, especially in so-called lottery-based voting.

To further disseminate MiniBrass, we plan to investigate more practical case studies and develop tools for more efficient constraint and preference modeling – suitable for both MiniZinc and MiniBrass. A web and mobile interface should help everyday decision-making with multiple users expressing their individual preferences. We expect that profound and interdisciplinary research has to be conducted in this emerging topic, as it addresses the core foundations of how we, as a society, want to agree on common decision-making with autonomous devices.

# Bibliography

Allen, T. E., Chen, M., Goldsmith, J., Mattei, N., Popova, A., Regenwetter, M., Rossi, F., and Zwilling, C. (2015). Beyond Theory and Data in Preference Modeling: Bringing Humans into the Loop. In *Proc. 4$^{th}$ Intl. Conf. Algorithmic Decision Theory (ADT'15)*, volume 9346 of *Lect. Notes Comp. Sci.*, pages 3–18. Springer.

Allouche, D., de Givry, S., Katsirelos, G., Schiex, T., and Zytnicki, M. (2015). Anytime Hybrid Best-first Search with Tree Decomposition for Weighted CSP. In *Proc. 21$^{st}$ Intl. Conf. Principles and Practice of Constraint Programming (CP'15)*, volume 9255 of *Lect. Notes Comp. Sci.*, pages 12–29. Springer.

Allouche, D., de Givry, S., and Schiex, T. (2010). Toulbar2, an Open-source Exact Cost Function Network Solver. Technical report, INRIA.

Allouche, D., Traoré, S., André, I., De Givry, S., Katsirelos, G., Barbe, S., and Schiex, T. (2012). Computational Protein Design as a Cost Function Network Optimization Problem. In *Proc. 18$^{th}$ Intl. Conf. Principles and Practice of Constraint Programming (CP'12)*, pages 840–849. Springer.

Amadio, R. M. and Curien, P.-L. (1998). *Domains and Lambda-Calculi.* Cambridge Tracts in Theoretical Computer Science 46. Cambridge University Press.

Anders, G. (2017). *Self-Organized Robust Optimization in Open Technical Systems: Self-Organization and Computational Trust for Scalable and Robust Resource Allocation under Uncertainty.* Dissertation, Universität Augsburg.

Anders, G., Siefert, F., Schiendorfer, A., Seebach, H., Steghöfer, J.-P., Eberhardinger, B., Kosak, O., and Reif, W. (2016). Specification and Design of Trust-Based Open Self-Organising Systems. In Reif, W., Anders, G., Seebach, H., Steghöfer, J.-P., André, E., Hähner, J., Müller-Schloer, C., and Ungerer, T., editors, *Trustworthy Open Self-Organising Systems*, volume 7 of *Autonomic Systems*, pages 17–53. Springer.

Anders, G., Siefert, F., Steghöfer, J.-P., and Reif, W. (2013). Trust-Based Scenarios – Predicting Future Agent Behavior in Open Self-Organizing Systems. In *Proc. 7$^{th}$ Intl. Workshop on Self Organizing Systems (IWSOS) 2013.* Springer.

Andréka, H., Ryan, M., and Schobbens, P.-Y. (2002). Operators and Laws for Combining Preference Relations. *Journal of Logic and Computation*, 12(1):13–53.

Ansótegui, C., Bofill, M., Palahí, M., Suy, J., and Villaret, M. (2011). W-MiniZinc: A Proposal for Modeling Weighted CSPs with MiniZinc. In *Proc. 1st Intl. Ws. MiniZinc (MZN'11)*.

Ansótegui, C., Bofill, M., Palahí, M., Suy, J., and Villaret, M. (2013). Solving Weighted CSPs with Meta-Constraints by Reformulation into Satisfiability Modulo Theories. *Constraints*, 18(2):236–268.

von Appen, J., Braun, M., Stetz, T., Diwold, K., and Geibel, D. (2013). Time in the Sun: The Challenge of High PV Penetration in the German Electric Grid. *Power and Energy Magazine*, 11(2):55–64.

Arrow, K. J. (1951). *Social Choice and Individual Values.* Yale University Press.

Arrow, K. J., Sen, A., and Suzumura, K. (2010). *Handbook of Social Choice and Welfare*, volume 2. Elsevier.

Arroyo, J. M. and Conejo, A. J. (2004). Modeling of Start-up and Shut-down Power Trajectories of Thermal Units. *Transactions on Power Systems*, 19(3):1562–1568.

Audi (2018). Audi TechDay Smart Factory. [Online at `https://www.audi-mediacenter.com/en/presskits/audi-techday-smart-factory-7008`; accessed 08-March-2018].

Awodey, S. (2010). *Category Theory.* Oxford University Press.

Barr, M. and Wells, C. (1990). *Category Theory for Computing Science.* Prentice Hall.

Becket, R. (2014). Specification of FlatZinc-Version 1.6. [Online at `http://www.minizinc.org/downloads/doc-1.6/flatzinc-spec.pdf`; accessed 08-March-2018].

Beldiceanu, N., Carlsson, M., Demassey, S., and Petit, T. (2007). Global Constraint Catalogue: Past, Present and Future. *Constraints*, 12(1):21–62.

Beldiceanu, N., Carlsson, M., Flener, P., and Pearson, J. (2013). On the Reification of Global Constraints. *Constraints*, 18(1):1–6.

van der Bellen, A. (1976). *Mathematische Auswahlfunktionen und gesellschaftliche Entscheidungen.* Interdisciplinary Systems Research. Birkhäuser. in German.

Benhamou, F. and Granvilliers, L. (2006). Continuous and Interval Constraints. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 16. Elsevier.

Bertele, U. and Brioschi, F. (1973). On Non-serial Dynamic Programming. *Journal on Combinatorial Theory, Series A*, 14(2):137–148.

Bessiere, C. (2006). Constraint Propagation. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 3. Elsevier.

Bickel, B. and Schuster, M. (2005). *Trends, Methoden und Grundsätze moderner Fabrik- und Produktionsplanung*. Hochschule Pforzheim. in German.

Bistarelli, S. (2004). *Semirings for Soft Constraint Solving and Programming*, volume 2962 of *Lect. Notes Comp. Sci.* Springer.

Bistarelli, S., Codognet, P., and Rossi, F. (2002). Abstracting Soft Constraints: Framework, Properties, Examples. *Artificial Intelligence*, 139:175–211.

Bistarelli, S., Frühwirth, T., Marte, M., and Rossi, F. (2004). Soft Constraint Propagation and Solving in Constraint Handling Rules. *Computational Intelligence*, 20(2):287–307.

Bistarelli, S., Fung, S. K. L., Lee, J. H. M., and Leung, H. (2003). A Local Search Framework for Semiring-based Constraint Satisfaction Problems. In *Proc. Ws. Soft Constraints (Soft'03)*.

Bistarelli, S., Montanari, U., and Rossi, F. (1997). Semiring-based Constraint Satisfaction and Optimization. *Journal of the ACM*, 44(2):201–236.

Bistarelli, S., Montanari, U., Rossi, F., Schiex, T., Verfaillie, G., and Fargier, H. (1999). Semiring-Based CSPs and Valued CSPs: Frameworks, Properties, and Comparison. *Constraints*, 4(3):199–240.

Bockmayr, A. and Hooker, J. N. (2005). Constraint Programming. *Handbooks in Operations Research and Management Science*, 12:559–600.

Bohnenblust, H. F. (1940). An Axiomatic Characterization of $L_p$-spaces. *Duke Math. J.*, 6:627–640.

Borning, A., Freeman-Benson, B., and Wilson, M. (1992). Constraint Hierarchies. *LISP and Symbolic Computation*, 5:223–270.

Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., and Poole, D. (2004). CP-nets: A Tool for Representing and Reasoning with Conditional Ceteris Paribus Preference Statements. *Journal on Artificial Intelligence Research*, 21:135–191.

Boutilier, C., Brafman, R. I., Geib, C., and Poole, D. (1997). A Constraint-based Approach to Preference Elicitation and Decision Making. In *AAAI Spring Symp. Qualitative Decision Theory*, pages 19–28.

Brandt, F., Conitzer, V., and Endriss, U. (2013). Computational Social Choice. In Weiß, G., editor, *Multiagent Systems*, chapter 6, pages 213–283. MIT Press, 2nd edition.

Brandt, F., Conitzer, V., Endriss, U., Lang, J., and Procaccia, A. D. (2016). *Handbook of computational social choice*. Cambridge University Press.

Cooper, M. C., de Givry, S., Sánchez, M., Schiex, T., Zytnicki, M., and Werner, T. (2010). Soft Arc Consistency Revisited. *Artificial Intelligence*, 174(7):449–478.

Cooper, M. C. and Schiex, T. (2004). Arc Consistency for Soft Constraints. *Artificial Intelligence*, 154(1):199–227.

CPLEX (2013). IBM ILOG CPLEX Optimizer. Online at `http://www-01.ibm.com/software/commerce/\optimization/cplex-optimizer/`, last accessed December 2013:.

Dalla Pozza, G., Pini, M. S., Rossi, F., and Venable, K. B. (2011). Multi-agent Soft Constraint Aggregation via Sequential Voting. In *Proc. 22$^{nd}$ Intl. Joint Conf. Artificial Intelligence (IJCAI'11)*, pages 172–177.

Dalla Pozza, G., Rossi, F., and Venable, K. B. (2010). Multi-agent Soft Constraint Aggregation: A Sequential Approach. In *Proc. 3$^{rd}$ Intl. Conf. Agents and Artificial Intelligence (ICAART'11*, volume 11.

Dash, R. K., Vytelingum, P., Rogers, A., David, E., and Jennings, N. R. (2007). Market-based Task Allocation Mechanisms for Limited-capacity Suppliers. *Transactions on Systems, Man and Cybernetics*, 37(3):391–405.

Dechter, R. (1999). Bucket Elimination: A Unifying Framework for Reasoning. *Artificial Intelligence*, 113(1):41–85.

Dechter, R. (2003). *Constraint Processing*. Morgan Kaufmann.

Dechter, R. and Frost, D. (2002). Backjump-based Backtracking for Constraint Satisfaction Problems. *Artificial Intelligence*, 136(2):147–188.

Dekker, J. J., de la Banda, M. G., Schutt, A., Stuckey, P. J., and Tack, G. (2018). Solver-Independent Large Neighbourhood Search. In *Proc. 24$^{th}$ Intl. Conf. Constraint Programming (CP'18)*, volume 11008 of *Lect. Notes Comp. Sci.*, pages 81–98.

Diaconescu, R. (1994). *Category-based Semantics for Equational and Constraint Logic Programming*. PhD thesis, Oxford University, Oxford.

Doyle, J. and McGeachie, M. (2003). Exercising Qualitative Control in Autonomous Adaptive Survivable Systems. In *Proc. 2$^{nd}$ Intl. Conf. Self-adaptive Software: Applications (IWSAS'01)*, Lect. Notes Comp. Sci., pages 158–170. Springer.

Ehrgott, M. (2005). *Multicriteria Optimization*, volume 491. Springer Science & Business Media.

Fargier, H. and Lang, J. (1993). Uncertainty in Constraint Satisfaction Problems: A Probabilistic Approach. In *Proc. Europ. Conf. Symbolic and Quantitative Approaches to Reasoning and Uncertainty (ECSQARU'93)*, volume 747 of *Lect. Notes Comp. Sci.*, pages 97–104. Springer.

Fioretto, F., Pontelli, E., and Yeoh, W. (2016). Distributed constraint optimization problems and applications: A survey. CoRR abs/1602.06347.

Fioretto, F., Pontelli, E., and Yeoh, W. (2018). Distributed constraint optimization problems and applications: A survey. *Journal of Artificial Intelligence Research*, 61:623–698.

Fleming, P. J. and Wallace, J. J. (1986). How Not to Lie with Statistics: The Correct Way to Summarize Benchmark Results. *Communications of the ACM*, 29(3):218–221.

Freuder, E. C. and Wallace, R. J. (1992). Partial constraint satisfaction. *Artificial Intelligence*, 58(1–3):21–70.

Frisch, A. M., Harvey, W., Jefferson, C., Martínez-Hernández, B., and Miguel, I. (2008). Essence: A Constraint Language for Specifying Combinatorial Problems. *Constraints*, 13(3):268–306.

Gadducci, F., Hölzl, M., Monreale, G. V., and Wirsing, M. (2013). Soft Constraints for Lexicographic Orders. In *Proc. 12th Mexican Intl. Conf. Artificial Intelligence (MICAI'13)*, volume 8265 of *Lect. Notes Comp. Sci.*, pages 68–79. Springer.

Gale, D. and Shapley, L. S. (1962). College Admissions and the Stability of Marriage. *The American Mathematical Monthly*, 69(1):9–15.

Ge, D., Jiang, X., and Ye, Y. (2011). A Note on the Complexity of Lp Minimization. *Mathematical programming*, 129(2):285–299.

Gent, I. P., MacIntyre, E., Presser, P., Smith, B. M., and Walsh, T. (1996). An Empirical Study of Dynamic Variable Ordering Heuristics for the Constraint Satisfaction Problem. In *Proc. 2nd Intl. Conf. Principles and Practice of Constraint Programming (CP'96)*, volume 1118 of *Lect. Notes Comp. Sci.*, pages 179–193. Springer.

Google (2017). Google Optimization Tools. [Online at `https://developers.google.com/optimization`, accessed 2017/06/29].

Güdemann, M., Ortmeier, F., and Reif, W. (2006). Formal Modeling and Verification of Systems with Self-x Properties. In *International Conference on Autonomic and Trusted Computing*, pages 38–47. Springer.

Guns, T., Dries, A., Nijssen, S., Tack, G., and De Raedt, L. (2017). MiningZinc: A Declarative Framework for Constraint-based Mining. *Artificial Intelligence*, 244:6–29.

Gusfield, D. and Irving, R. W. (1989). *The Stable Marriage Problem: Structure and Algorithms.* MIT press.

Hanke, J., Kosak, O., and Schiendorfer, A. (2018). Self-organized Resource Allocation for Reconfigurable Robot Ensembles. In *Proc. 12th Intl. Conf. "Self-Adaptive and Self-Organizing Systems" (SASO18)*. IEEE. to appear.

Hansson, S. O. (1994). Decision Theory – A Brief Introduction.

Hebrard, E. (2008). Mistral, a Constraint Satisfaction Library. *Proc. 3rd Intl. CSP Solver Competition*, 3:3.

Hebrard, E., O'Mahony, E., and O'Sullivan, B. (2010). Constraint Programming and Combinatorial Optimisation in Numberjack. In *Proc. 7th Intl. Conf. Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'10)*, volume 6140 of *Lect. Notes Comp. Sci.*, pages 181–185. Springer.

van Hentenryck, P. (1999). *The OPL Optimization Programming Language.* MIT Press.

van Hentenryck, P. and Coffrin, C. (2014). Teaching Creative Problem Solving in a MOOC. In *Proc. 45th ACM Technical Symposium on Computer Science Education*, pages 677–682. ACM.

Heuck, K., Dettmann, K.-D., and Schulz, D. (2010). *Elektrische Energieversorgung.* Vieweg+Teubner. in German.

Higham, N. J. (1992). Estimating the Matrixp-Norm. *Numerische Mathematik*, 62(1):539–555.

Hinrichs, C., Lehnhoff, S., and Sonnenschein, M. (2013). COHDA: A Combinatorial Optimization Heuristic for Distributed Agents. In *Proc. 5th Intl. Conf. Agents and Artificial Intelligence (ICAART'13)*, pages 23–39. Springer.

van Hoeve, W.-J. (2011). Over-constrained Problems. In Milano, M. and van Hentenryck, P., editors, *Hybrid Optimization*, volume 45 of *Optimization and its Applications*, pages 191–225. Springer.

van Hoeve, W.-J. and Katriel, I. (2006). Global Constraints. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 7. Elsevier.

Hoffmann, A., Nafz, F., Schierl, A., Seebach, H., and Reif, W. (2011). Developing self-organizing robotic cells using organic computing principles. In *Bio-Inspired Self-Organizing Robotic Systems*, pages 253–273. Springer.

Hooker, J. N. (2007). *Integrated Methods for Optimization*, volume 100. Springer Science & Business Media.

Hoos, H. H. and Tsang, E. (2006). Local Search Methods. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 8. Elsevier.

Hosobe, H. (2009). Constraint Hierarchies as Semiring-based CSPs. In *Proc. 21st Intl. Conf. Tools with Artificial Intelligence (ICTAI'09)*, pages 176–183. IEEE.

Hundt, M., Barth, R., Sun, N., Wissel, S., and Voß, A. (2009). Verträglichkeit von erneuerbaren Energien und Kernenergie im Erzeugungsportfolio. *Technisch-ökonomische Aspekte. Studie des Instituts für Energiewirtschaft und rationelle Energieanwendung (IER) im Auftrag der E. ON Energie AG. Stuttgart.* in German.

Hurley, B., O'Sullivan, B., Allouche, D., Katsirelos, G., Schiex, T., Zytnicki, M., and de Givry, S. (2016). Multi-language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*, 21(3):413–434.

Hölzl, M., Meier, M., and Wirsing, M. (2009). Which Soft Constraints do you Prefer? In *Proc. 7th Intl. Ws. Rewriting Logic and its Applications (WRLA'08)*, volume 238(3) of *Electronic Notes in Theoretical Computer Science*, pages 189–205.

Junker, U. (2009). Outer Branching: How to Optimize under Partial Orders? In *Proc. 7th Intl. Conf. Multiobjective Programming and Goal Programming (MOPGP'06)*, volume 618 of *Lect. Notes Econom. Math. Syst.*, pages 99–109. Springer.

Jussien, N., Rochart, G., and Lorca, X. (2008). Choco: An Open-source Java Constraint Programming Library. In *Proc. Ws. Open-source Software for Integer and Constraint Programming (OSSICP'08)*, pages 1–10.

Kaci, S., Patel, N., and Prince, V. (2014). From NL preference expressions to comparative preference statements: A preliminary study in eliciting preferences for customised decision support. In *Proc. 26st Intl. Conf. Tools with Artificial Intelligence (ICTAI'14)*, pages 591–598. IEEE.

Karl, J. (2012). *Dezentrale Energiesysteme: Neue Technologien im liberalisierten Energiemarkt*. Oldenbourg. in German.

Kießling, W. and Köstler, G. (2002). Preference SQL: Design, Implementation, Experiences. In *Proc. 28th Intl. Conf. Very Large Data Bases (VLDB'02)*, pages 990–1001. Morgan Kaufmann.

Knapp, A. and Schiendorfer, A. (2014). Embedding Constraint Relationships into C-Semirings. Technical Report 2014-03, Institut für Software & Systems Engineering, Universität Augsburg.

Knapp, A., Schiendorfer, A., and Reif, W. (2014). Quality over Quantity in Soft Constraints. In *Proc. 26th Intl. Conf. Tools with Artificial Intelligence (ICTAI'14)*, pages 453–460.

Kosak, O., Wanninger, C., Angerer, A., Hoffmann, A., Schiendorfer, A., and Seebach, H. (2016). Towards Self-organizing Swarms of Reconfigurable Self-aware Robots. In *Proc. 1st Intl. Ws. Engineering Collective Adaptive Systems (ECAS'16)*, pages 204–209. IEEE.

Kuchcinski, K. and Szymanek, R. (2013). JaCoP — Java constraint programming solver. In *Proc. Ws. CP Solvers: Modeling, Applications, Integration, and Standardization.*

Léauté, T., Ottens, B., and Szymanek, R. (2009). FRODO 2.0: An Open-source Framework for Distributed Constraint Optimization. In *Proc. Intl. Ws. Distributed Constraint Reasoning (DCR'09)*, pages 160–164.

Leenen, L., Anbulagan, Meyer, T., and Ghose, A. K. (2007). Modeling and Solving Semiring Constraint Satisfaction Problems by Transformation to Weighted Semiring Max-SAT. In *Proc. 20th Australian Joint Conf. Artificial Intelligence (ACAI'07)*, volume 4830 of *Lect. Notes Comp. Sci.*, pages 202–212. Springer.

Lenstra, J. K. and Kan, A. R. (1979). Computational Complexity of Discrete Optimization Problems. In *Annals of Discrete Mathematics*, volume 4, pages 121–140. Elsevier.

Levasseur, N., Boizumault, P., and Loudni, S. (2007). A Value Ordering Heuristic for Weighted CSP. In *Proc. 19th IEEE Intl. Conf. Tools with Artificial Intelligence (ICTAI'07)*, volume 1, pages 259–262. IEEE.

LEW (2018). Smart operator. [Online (in German) at `https://www.lew.de/energiezukunft/smart-operator`; accessed 08-March-2018].

Li, S. and Ying, M. (2008). Soft Constraint Abstraction based on Semiring Homomorphism. *Theoretical Computer Science*, 403(2-3):192–201.

Lovász, L. and Plummer, M. D. (2009). *Matching Theory*, volume 367. American Mathematical Society.

Marriott, K. and Stuckey, P. J. (1998). *Programming with Constraints: An Introduction.* MIT press.

Meseguer, P., Rossi, F., and Schiex, T. (2006). Soft Constraints. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 9. Elsevier.

Miller, S., Ramchurn, S. D., and Rogers, A. (2012). Optimal Decentralised Dispatch of Embedded Generation in the Smart Grid. In *Proc. 11th Intl. Conf. Autonomous Agents and Multiagent Systems (AAMAS'12)*, volume 1, pages 281–288. International Foundation for Autonomous Agents and Multiagent Systems.

Minton, S., Johnston, M. D., Philips, A. B., and Laird, P. (1992). Minimizing Conflicts: A Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Artificial Intelligence*, 58(1-3):161–205.

Modi, P. J., Shen, W.-M., Tambe, M., and Yokoo, M. (2005). ADOPT: Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, 161(1-2):149–180.

Mohr, R. and Henderson, T. C. (1986). Arc and Path Consistency Revisited. *Artificial Intelligence*, 28(2):225–233.

Nafz, F. (2012). *Verhaltensgarantien in Selbst-Organisierenden Systemen.* Logos Verlag. in German.

Nafz, F., Seebach, H., Steghöfer, J.-P., Anders, G., and Reif, W. (2011). Constraining Self-organisation Through Corridors of Correct Behaviour: The Restore Invariant Approach. In Müller-Schloer, C., Schmeck, H., and Ungerer, T., editors, *Organic Computing — A Paradigm Shift for Complex Systems*, volume 1 of *Autonomic Systems*, pages 79–93. Springer.

Nethercote, N. (2014). Converting minizinc to flatzinc. [Online at `http://www.minizinc.org/downloads/doc-1.6/mzn2fzn.pdf`; accessed 08-March-2018].

Nethercote, N., Stuckey, P. J., Becket, R., Brand, S., Duck, G. J., and Tack, G. (2007). MiniZinc: Towards a standard CP modelling language. In *Proc. 13th Intl. Conf. Principles and Practice of Constraint Programming (CP'07)*, volume 4741 of *Lect. Notes Comp. Sci.*, pages 529–543. Springer.

Netzer, A. and Meisels, A. (2011). SOCIAL DCOP – Social Choice in Distributed Constraints Optimization. In *Proc. 5th Intl. Symp. Intelligent Distributed Computing (IDC'11)*, pages 35–47. Springer.

Nieße, A., Sonnenschein, M., Hinrichs, C., and Bremer, J. (2016). Local Soft Constraints in Distributed Energy Scheduling. In *Proc. 5th Intl. Ws. Smart Energy Networks & Multi-Agent Systems (SEN-MAS'16)*, pages 1517–1525. IEEE.

Nisan, N. and Ronen, A. (1999). Algorithmic Mechanism Design. In *Proc. 31st Ann. ACM Symp. Theory of Computing (STACS'99)*, pages 129–140. ACM.

Pacuit, E. (2012). Voting Methods. `http://plato.stanford.edu/archives/win2012/entries/voting-methods/`.

Padhy, N. P. (2004). Unit Commitment- A Bibliographical Survey. *Transactions on Power Systems*, 19(2):1196–1205.

Petit, T., Régin, J.-C., and Bessière, C. (2000). Meta-Constraints on Violations for Over Constrained Problems. In *Proc. 12$^{th}$ Intl. Conf. Tools with Artificial Intelligence (ICTAI'00)*, pages 358–365.

Piekutowski, M. and Rose, I. (1985). A Linear Programming Method for Unit Commitment Incorporating Generator Configurations, Reserve and Plow Constraints. *Transactions on Power Apparatus and Systems*, PAS-104(12):3510–3516.

Pierce, B. C. (1991). *Basic Category Theory for Computer Scientists*. MIT Press.

Pini, M. S., Rossi, F., and Venable, K. B. (2013). Bribery in Voting with Soft Constraints. In *Proc. 27$^{th}$ Nat. Conf. Artificial Intelligence (AAAI'13)*, pages 803–809. AAAI Press.

Pisinger, D. and Ropke, S. (2010). Large Neighborhood Search. In *Handbook of Metaheuristics*, pages 399–419. Springer.

Ramchurn, S. D., Vytelingum, P., Rogers, A., and Jennings, N. R. (2012). Putting the 'Smarts' into the Smart Grid: A Grand Challenge for Artificial Intelligence. *Communications of the ACM*, 55(4):86–97.

Régin, J.-C. (1994). A Filtering Algorithm for Constraints of Difference in CSPs. In *Proc. 12$^{th}$ Nat. Conf. Artificial Intelligence (AAAI'94)*, volume 94, pages 362–367. AAAI Press.

Régin, J.-C. (1996). Generalized Arc Consistency for Global Cardinality Constraint. In *Proc. 13$^{th}$ Nat. Conf. Artificial Intelligence (AAAI'96)*, volume 1, pages 209–215.

Rendl, A., Guns, T., Stuckey, P. J., and Tack, G. (2015). MiniSearch: A Solver-independent Meta-search Language for MiniZinc. In *Proc. 21$^{st}$ Intl. Conf. Constraint Programming (CP'15)*, volume 9255 of *Lect. Notes Comp. Sci.*, pages 376–392.

Rendl, A., Tack, G., and Stuckey, P. J. (2014). Stochastic MiniZinc. In *Proc. 20$^{th}$ Intl. Conf. Principles and Practice of Constraint Programming (CP'14)*, volume 8656 of *Lect. Notes Comp. Sci.*, pages 636–645. Springer.

Rentmeesters, M. J., Tsai, W. K., and Lin, K.-J. (1996). A Theory of Lexicographic Multi-criteria Optimization. In *Proc. 2$^{nd}$ Intl. Conf. Engineering of Complex Computer Systems (ICECCS'96)*, pages 76–79. IEEE.

Reny, P. J. (2001). Arrow's theorem and the Gibbard-Satterthwaite theorem: a unified approach. *Economics Letters*, 70(1):99–105.

Rollón, E. (2008). *Multi-objective Optimization in Graphical Models*. Dissertation, Universitat Politècnica de Catalunya, Barcelona.

Rossi, F. (2014). Collective decision making: a great opportunity for constraint reasoning. *Constraints*, 19(2):186–194.

Rossi, F. and Pilan, I. (2003). Abstracting Soft Constraints: Some Experimental Results on Fuzzy CSPs. In Apt, K. R., Fages, F., Rossi, F., Szeredi, P., and Váncza, J., editors, *Sel. Papers Joint ERCIM/CologNET Intl. Ws. Constraint Solving and Constraint Logic Programming (CSCLP'03)*, volume 3010 of *Lect. Notes Comp. Sci.*, pages 107–123. Springer.

Rossi, F., Van Beek, P., and Walsh, T. (2006). *Handbook of Constraint Programming*. Elsevier.

Rossi, F., Venable, K. B., and Walsh, T. (2008a). Preferences in Constraint Satisfaction and Optimization. *AI Magazine*, 29(4):58.

Rossi, F., Venable, K. B., and Walsh, T. (2008b). Preferences in Constraint Satisfaction and Optimization. *AI Magazine*, 29(4):58–68.

Rothe, J., Baumeister, D., Lindner, C., and Rothe, I. (2011). Einführung in Computational Social Choice. in German.

Ruttkay, Z. (1994). Fuzzy Constraint Satisfaction. In *Proc. 3rd Intl. Fuzzy Systems Conf.*, pages 1263–1268. IEEE.

Sannella, D. and Tarlecki, A. (2012). *Foundations of Algebraic Specification and Formal Software Development*. EATCS Monographs in Theoretical Computer Science. Springer.

Schiendorfer, A., Anders, G., Steghöfer, J.-P., and Reif, W. (2015a). Abstraction of Heterogeneous Supplier Models in Hierarchical Resource Allocation. *Transactions on Computational Collective Intelligence XX*, 9420:23–53.

Schiendorfer, A., Eberhardinger, B., Reif, W., and André, E. (2015b). Back-to-Back Testing a Soft Constraint Model for a Smart Exhibition Space. In *Proc. 14th Intl. Ws. "Constraint Modelling and Reformulation" (ModRef'15)*.

Schiendorfer, A., Knapp, A., Anders, G., and Reif, W. (2018). MiniBrass: Soft Constraints for MiniZinc. *Constraints*.

Schiendorfer, A., Knapp, A., Steghöfer, J.-P., Anders, G., Siefert, F., and Reif, W. (2015c). Partial Valuation Structures for Qualitative Soft Constraints. In Nicola, R. D. and Hennicker, R., editors, *Software, Services and Systems — Essays Dedicated to Martin Wirsing on the Occasion of His Emeritation*, volume 8950 of *Lect. Notes Comp. Sci.*, pages 115–133. Springer.

Schiendorfer, A., Steghöfer, J.-P., and Reif, W. (2014a). Synthesised Constraint Models for Distributed Energy Management. In *Proc. 3rd Intl. Ws. Smart Energy Networks & Multi-Agent Systems (SEN-MAS'14)*, pages 1529–1538.

Schiendorfer, A., Steghöfer, J.-P., Knapp, A., Nafz, F., and Reif, W. (2013). Constraint Relationships for Soft Constraints. In *Proc. 33rd SGAI Intl. Conf. Innovative Techniques and Applications of Artificial Intelligence (AI'13)*, pages 241–255. Springer.

Schiendorfer, A., Steghöfer, J.-P., and Reif, W. (2014b). Synthesis and Abstraction of Constraint Models for Hierarchical Resource Allocation Problems. In *Proc. 6th Intl. Conf. Agents and Artificial Intelligence (ICAART'14)*, volume 2, pages 15–27. SciTePress.

Schiex, T. (1992). Possibilistic Constraint Satisfaction Problems or "How to handle Soft Constraints?". In *Proc. 8th Conf. Uncertainty in Artificial Intelligence (UAI'92)*, pages 268–275. Elsevier.

Schiex, T., Fargier, H., and Verfaillie, G. (1995). Valued Constraint Satisfaction Problems: Hard and Easy Problems. In *Proc. 14th Intl. Joint Conf. Artificial Intelligence (IJCAI'95)*, volume 1, pages 631–639. Morgan Kaufmann.

Schulte, C. and Carlsson, M. (2006). Finite Domain Constraint Programming Systems. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 14. Elsevier.

Schulte, C., Lagerkvist, M. Z., and Tack, G. (2006). Gecode: Generic Constraint Development Environment. In *INFORMS Ann. Meeting*.

Seebach, H. (2011). *Konstruktion Selbst-Organisierender Softwaresysteme*. Logos Verlag. in German.

Seebach, H., Nafz, F., Steghöfer, J.-P., and Reif, W. (2011). How to Design and Implement Self-organising Resource-flow Systems. In *Organic Computing—A Paradigm Shift for Complex Systems*, pages 145–161. Springer.

Shapiro, L. G. and Haralick, R. M. (1981). Structural Descriptions and Inexact Matching. *Transactions on Pattern Analysis and Machine Intelligence*, 3(5):504–519.

Shaw, P. (1998). Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. In *Proc. 4th Intl. Conf. Principles and Practice of Constraint Programming (CP'98)*, volume 1520 of *Lect. Notes Comp. Sci.*, pages 417–431. Springer.

Shoham, Y. and Leyton-Brown, K. (2008). *Multiagent Systems: Algorithmic, Game-theoretic, and Logical Foundations*. Cambridge University Press.

Siefert, F. (2017). *Selbst-organisiertes, trust-bewusstes Supply Demand Management in Smart Grids*. Dissertation, Universität Augsburg. in German.

Smith, B. M. (2006). Modelling. In Rossi, F., van Beek, P., and Walsh, T., editors, *Handbook of Constraint Programming*, chapter 11. Elsevier.

Steghöfer, J.-P. (2014). *Large-Scale Open Self-Organising Systems: Managing Complexity with Hierarchies, Monitoring, Adaptation, and Principled Design*. Dissertation, Universität Augsburg.

Stuckey, P. J., de la Banda, M. G., Maher, M., Marriott, K., Slaney, J., Somogyi, Z., Wallace, M., and Walsh, T. (2005). The G12 Project: Mapping Solver Independent Models to Efficient Solutions. In *Proc. 11th Intl. Conf. Principles and Practice of Constraint Programming (CP'05)*, volume 3709 of *Lect. Notes Comp. Sci.*, pages 13–16. Springer.

Stuckey, P. J., Feydy, T., Schutt, A., Tack, G., and Fischer, J. (2014). The MiniZinc Challenge 2008–2013. *AI Magazine*, 35(2):55–60.

Stuckey, P. J. and Tack, G. (2013). MiniZinc with Functions. In *Proc. 10th Intl. Conf. Integration of Artificial Intelligence and Operations Research Techniques in Constraint Programming (CPAIOR'13)*, volume 7874 of *Lect. Notes Comp. Sci.*, pages 268–283. Springer.

Sánchez, M., Allouche, D., de Givry, S., and Schiex, T. (2009). Russian Doll Search with Tree Decomposition. In *Proc. 21st Intl. Joint Conf. Artificial Intelligence (IJCAI'09)*, pages 603–608.

Wahbi, M., Ezzahir, R., Bessiere, C., and Bouyakhf, E.-H. (2011). DisChoco 2: A Platform for Distributed Constraint Reasoning. In *Proc. Intl. Ws. Distributed Constraint Reasoning (DCR'11)*, volume 11, pages 112–121.

Wanninger, C., Eymüller, C., Hoffmann, A., Kosak, O., and Reif, W. (2018). Synthesising Capabilities for Collective Adaptive Systems from Self-Descriptive Hardware Devices – Bridging the Reality Gap. In *Proc. 8th Intl. Symp. Leveraging Applications of Formal Methods, Verification and Validation (ISOLA'18).*

Wells, M. B. (1971). *Elements of Combinatorial Computing.* Pergamon.

Yeoh, W., Felner, A., and Koenig, S. (2010). BnB-ADOPT: An Asynchronous Branch-and-bound DCOP Algorithm. *Journal of Artificial Intelligence Research*, 38:85–133.

Yokoo, M. and Hirayama, K. (2000). Algorithms for Distributed Constraint Satisfaction: A Review. *Autonomous Agents and Multi-Agent Systems*, 3(2):185–207.