TLP **4** (5 & 6): 753–762, 2004.
Printed in the United Kingdom

Book reviews

DOI: 10.1017/S1471068404212182

Logic for Learning: Learning Comprehensible Theories from Structured Data by John W. Lloyd, Springer-Verlag, 2003, hard cover: ISBN 3-540-42027-4, x + 256 pages.

The publication of a new book by John Lloyd, nearly twenty years after the first edition of his classic *Foundations of Logic Programming* saw the light, has to count as a major event in the computational logic community. Much has happened in two decades, and those expecting a third, extended edition of *Foundations of Logic Programming* will be in for a surprise. This time, John has tried his hand at machine learning, and his main aim in *Logic for Learning* is to demonstrate "the rich and fruitful interplay between the fields of computational logic and machine learning".

In this book review I will try to assess to which extent this aim has been achieved. In doing so I will attempt to approach matters from the point of view of a computational logician (even though I am more of a machine learner myself). As with any book review, the main questions are "who should read this book?" and "what's in it for them?". I should perhaps add that John Lloyd and I are no strangers, our appointments in Bristol having overlapped for about two years in 1997–1999. After John's return to Australia we continued to publish joint papers on subjects closely related to *Logic for Learning*. While this means I am sympathetic towards the general approach advocated in *Logic for Learning*, I do not believe this seriously affects my capacity for critical assessment, which of course in an academic review is chiefly related to the question "would I have written the same book?".

The most obvious connection between machine learning and computational logic is in the area of knowledge representation. Both data input to a learning system and models output by it need to be expressed in some kind of declarative formalism (assuming a symbolic learner; non-symbolic learners such as neural networks only need a declarative formalism for the data). The sophistication required of the declarative language depends on the nature of the data. *Logic for Learning* is concerned with learning from rich and complex data, witness its subtitle *Learning Comprehensible Theories from Structured Data*. The logic used in this book is correspondingly rich: a typed higher-order logic, with a rewrite-based inference system combining aspects of functional and logic programming. This will sound familiar to some readers, as Lloyd has been working in the past on the declarative programming language Escher along these lines. However, Escher is not mentioned in the book, presumably because a fully-fledged Escher implementation is neither available nor needed for the kind of machine learning applications considered in this book.

The fact that the logic is higher-order means that a wider range of data types is available than the usual data constructor types that can be used to build tree-like structures. A set can be identified with its characteristic function which can be represented by a λ-abstraction. By the same token, we can have multisets or bags

(λ-abstractions mapping into integers rather than booleans). Sets and multisets are represented extensionally as finite look-up tables: for instance, the set $\{1, 2, 3\}$ is represented by the higher-order term

$$\lambda x.\mathit{if}\ x = 1\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = 2\ \mathit{then}\ \top\ \mathit{else}\ \mathit{if}\ x = 3\ \mathit{then}\ \top\ \mathit{else}\ \bot$$

Describing individuals that make up a data set for learning by higher-order terms is elegant and powerful. The type signature provides the meta-data that is exploited by the learning system to construct the search space for possible classification models. Furthermore, the inductive nature of the type signature facilitates the proof of formal results (e.g., that a distance measure defined on pairs of higher-order terms is a metric).

Higher-order logic is also used extensively in the construction of classification models. A wide range of classifiers is used in machine learning; *Logic for Learning* describes in some detail a learning system called ALKEMY that constructs higher-order decision trees. Internal nodes in such a tree contain predicates (i.e., formulas with one free variable corresponding to the type of individuals), and for a given individual we take one of two branches emanating from the node depending on whether the predicate is true of the individual. The tree's leaves are then labelled with predicted classes. The main task of a decision tree learner is to decide which predicates are to be used for the tree's internal nodes, and here again higher-order logic is used for the construction of such predicates using a predicate rewrite system. The advantage of using higher-order logic here is that these predicates can be described by function composition and variables can be dispensed with. Here is an example used in the book on a well-known toy problem classifying trains:

$$\mathit{setExists}_1(\wedge_2(\mathit{projLength} \circ (= \mathit{Short}))(\mathit{projKind} \circ (= \mathit{Closed})))$$

This predicate is true if the train, represented as a set of cars, contains a car satisfying a single predicate, which itself is a conjunction of two predicates, each of which consists of a projection function (since cars are represented by tuples) composed with a predicate comparing the result of the projection to a constant value. The use of higher-order logic here is elegant but not crucial, as it does not offer additional power over the first-order refinement operators or hypothesis grammars that are used in inductive logic programming (ILP).

Logic for Learning is a fairly slim volume consisting of six main chapters titled Introduction, Logic, Individuals, Predicates, Computation, and Learning. This organisation has an inherent logic, although it does mean that the reader will not see any learning until she has read more than 200 pages. However, the introductory chapter provides a very helpful "shortest path to the machine learning applications". Chapter 1 is also remarkable in another respect: it provides an excellent, intuitive and very readable introduction to the rest of the book. The remaining chapters are written in the terse and mathematical style familiar to readers of *Foundations of Logic Programming*. As such, the book is more geared towards computational logicians who are interested in machine learning, while machine learners who want to know what computational logic has to offer may find the mathematical approach a little bit unforgiving (but not unsurmountable). Literature references are collected at the end of each chapter rather than being sprinkled throughout the text. Each chapter also has a list of (almost exclusively mathematical) exercises.

https://doi.org/10.1017/S1471068404212182 Published online by Cambridge University Press

Of course the use of computational logic in machine learning is not new: ILP has been a very active research area for well over twenty years, with its own annual conference. However, readers interested in an overview of ILP and how the field has developed should look elsewhere. What *Logic for Learning* has to offer is a very personal view on the interplay between computational logic and machine learning. It is almost a rational reconstruction of what ILP could have been, had it used Escher-style higher-order logic rather than Prolog from the beginning. This is a worthy and fascinating exercise, as it could be argued that a considerable amount of work in ILP has been concerned with repairing the restrictions and shortcomings of Prolog, and it is refreshing – even for ILP experts – to see things approached from such a different perspective.

In conclusion, this book contributes to the trend towards inter-disciplinarity that can be observed in many scientific disciplines and particularly in computer science. We can also perceive this trend in computational logic, which is branching out to application areas such as Semantic Web and e-Science. As *Logic for Learning* argues convincingly, machine learning is another very promising application area for computational logic, and this book provides an excellent basis for logicians who want to move in that direction. I recommend that they follow it up with an introductory text on machine learning – such as Tom Mitchell's *Machine Learning* (McGraw-Hill, 1997) or Ian Witten and Eibe Frank's *Data Mining* (Morgan Kaufmann, 2000) – as *Logic for Learning* doesn't contain any material on heuristics, overfitting, tree-pruning, and other standard machine learning techniques. The book can also be used as a textbook in a mathematically oriented advanced graduate course.

My main criticism of the book is that it would have been even more convincing if it had demonstrated how a range of machine learning techniques – not just decision tree learning – could have benefited from the higher-order logic treatment. I would have wanted to see material on classification rule learning, association rule learning, learning of probabilistic models, and so on. As it stands, the book offers a wealth of technical material but only one or two examples are given of how it could be used in machine learning, which made me feel like someone who has climbed a tall tower only to be given a few minutes to admire the view. I would also have liked to see some serious applications on real-world, highly structured data. But it is always easy to say to an author: "This is great stuff, why didn't you write more?" The bottom line is that it is indeed great stuff, which deserves to be taken serious by any computational logician who wants to stay ahead of her game.

Peter Flach
University of Bristol, United Kingdom

DOI: 10.1017/S1471068404222189

Constraint Processing by Rina Dechter, Morgan Kaufmann Publishers, 2003, hard cover: ISBN 1-55860-890-7, xx + 481 pages.

"Constraint Processing" provides a comprehensive treatment of the theory of Constraint Satisfaction Problems (CSP). Constraint processing has made important contributions to the area of Artificial Intelligence. It has also contributed to

other areas such as programming languages, operations research and optimization, databases and applied mathematics. The book starts from the very basics of constraint satisfaction to the algorithms, inference and search techniques used in constraint processing. The focus is on a rigorous development of the fundamentals and principles of constraint satisfaction while at the same time being written in a readable textbook style.

The book consists of fifteen chapters divided into two main parts. Chapter 1 is an introduction. The first part from chapters 2 to 7 covers the basics of constraint processing. The second part from chapters 8 to 15 covers more advanced material.

In the first part, chapter 2 gives examples of constraint problems followed by the basic definitions of constraint networks and their properties. Chapter 3 develops the basic algorithms for constraint propagation. It starts with the arc consistency algorithm, followed by path consistency and k-consistency as well as the relaxation to bounds consistency. This is then applied to constraint propagation for numeric and Boolean constraints.

Chapter 4 investigates properties which allow for backtrack-free search leading to directional and adaptive consistency. The next important ingredient is backtracking search in Chapter 5. Enhancements to backtracking search using various look-ahead strategies are developed and heuristics for determining value and variable ordering search strategies are covered.

Chapter 6 improves naive backtracking using look-back schemes for backjumping and learning. Backjumping is also integrated with look-ahead. Chapter 6 also introduces the use of random problems for studying algorithms and the phase transition phenomena encountered when solving hard problems. Chapter 7 completes the basic section with stochastic greedy local search as a complement to backtracking search. It discusses greedy and random walk strategies for local search and then integrates inference and local search.

In the second part of the book, chapter 8 first extends consistency notions developed in the first part, which are mainly on binary constraint networks, to the general non-binary case using relational consistency. This is then applied to domain tightness and constraint tightness to show when local consistency is also globally consistent. The rest of the chapter develops constraint inference for special classes of constraints, namely, row convex, Boolean and and linear arithmetic constraints.

Chapter 9 covers acyclic networks for non-binary constraints and tree decomposition methods. Chapter 10 revisits hybrid algorithms using search and inference. This chapter expands on the time-space trade-offs when combining structural properties of the constraint graph with search. Chapter 13 deals with solving optimization problems, first using branch-and-bound search and then the bucket and mini-bucket elimination which is a dynamic programming approach to optimization.

There are two chapters devoted to special kinds of constraint networks. Chapter 12 deals with temporal constraint networks which arise from temporal reasoning. Chapter 14 shows how the techniques developed such as bucket elimination and the cycle cutset method can be used in the context of probabilistic/bayesian networks. Finally, there are two contributed chapters. Chapter 11 by David Cohen and Peter Jeavons presents the theory of tractable constraint languages. Chapter 15

https://doi.org/10.1017/S1471068404212182 Published online by Cambridge University Press

by Francesca Rossi describes the Constraint Logic Programming (CLP) scheme and its programming paradigm which integrates constraint processing into logic programming.

Throughout much of the book, constraint processing is applied to Boolean problems. Thus constraint processing for Boolean problems serves as a running example for the specialisation of the techniques discussed in the book. For example, the unit propagation algorithm is a special case of relational arc consistency for Boolean constraint propagation. Chapter 5 shows how using unit propagation with backtracking search defines the well known DPLL procedure for Boolean satisfiability. Chapter 6 adds learning to the SAT solver. Directional resolution is developed by specializing directional arc consistency in Chapter 8.

This is the first book with a rigorous treatment of the theory and algorithms for Constraint Satisfaction Problems. While it is not encyclopedic, it gives a broad coverage while covering the major CSP results. Algorithms are given an integrated treatment in a precise but understandable fashion. This is due in part to the systematic use of high level operations using constraint composition and projection in the algorithms and the presentation of algorithms as constraint inference. The time complexities of algorithms are covered. As mentioned, there is an emphasis on Boolean problems which is the consistent link across chapters. The author has presented experimental results in some of the more advanced material which is useful to understand practical performance and trade-offs of the various algorithms.

This book is on the theory of constraint processing. It is not about how to model problems, implementation of constraint solvers, heuristic for constraint problems (there is a small treatment in the book), how to solve combinatorial problems, etc.

To place the book in context, it is useful to compare it with two other books on constraints also published in 2003: (i) Thom Frühwirth and Slim Abdennadher, *Essentials of Constraint Programming*, Springer; and (ii) Krzysztof R. Apt, *Principles of Constraint Programming*, Cambridge. The approach of the first book is from a programming language perspective, namely, CLP. It also develops Constraint Handling Rules (CHR) as a programming language for writing constraint solvers. The second book covers both the constraint processing techniques in CSP as well as constraint programming. Like the first book the approach is also more rule based. The Dechter book differs from the above two in its focus on the theory and algorithms for CSP. Although it has a chapter on CLP, this mainly serves as an introduction and linkage to constraint programming.

To summarize, this book distills well over three decades worth of development in CSP and constraint processing in a single textbook. I wholeheartedly recommend it to students, researchers and practitioners in artificial intelligence, constraint programming and operations research who want to know more about the theory of constraint processing.

Roland H. C. Yap
National University of Singapore, Singapore

DOI: 10.1017/S1471068404232185

Principles of Constraint Programming by Krzysztof R. Apt, Cambridge University Press, 2003, hard cover: ISBN 0-521-82583-0, xii + 407 pages, price: 50 US $ or 35 £.[1]

Constraint programming is a very active research area that lies at the intersection of AI, OR, Programming Languages, Data Bases, and Graph Theory, and whose main aim is to model naturally and solve efficiently real-life problems that can be cast as a set of objects and a set of restrictions (that is, constraints) on how these objects can co-exist in the same problem. Constraint programming started a long time ago, with the first propagation algorithms by Waltz, Montanari, Mackworth and Freuder in the late 70's, then it found high-level language support in Constraint Logic Programming (CLP) in the late 80's, and since then it has evolved to a very complex discipline with many different modeling languages and solving techniques.

K. R. Apt's book on constraint programming (Apt, 2003) has been published in 2003 and is thus one the latest books published on this subject. To be able to contextualize the value of this book, we first need to draw a brief history of the books that have been published on this same subject. The first book on constraint programming was published in 1989 (Hentenryck, 1989) by Pascal Van Hentenryck, and it showed how to exploit some of the constraint techniques typical of AI in the CLP world, by using the CHIP language. A few years later, in 1993, Edward Tsang wrote a book on constraint solving (Tsang, 1993), which contained the main notions as well as some advanced solving techniques based on local search and genetic algorithms. This book had an AI flavor, and did not deal with programming languages based on constraints such as CLP. In the same year, Vijay Saraswat published a book on Concurrent Constraint Programming (CCP) (Saraswat, 1993), a concurrent abstract model of computation, originated from extensions of CLP, where constraints are not only a way to express real-life restrictions, but also a way for concurrent agents to communicate. Van Hentenryck's book was a good source of information for CLPers, but as time passed new techniques, propagation algorithms, and language constructs were proposed, so the book on CLP by Marriott and Stuckey (Marriott & Stuckey, 1998), that appeared in 1998, updated the readers by containing the basic notions of constraint propagation, and by also giving a formal syntax and semantics of the current computational model of CLP. So, by 1998 people knew where to find the main notions and results of constraint solving, as well as the syntax and semantics of constraint programming languages like CLP and CCP. However, what was still missing was a place where to find also the most advanced concepts and results, and also a formal and uniform way to look at the whole field.

In 2003 three new books on constraint programming came out, and they succeeded in solving both these problems. One is a very comprehensive book on most of the notions and results on constraint programming (Dechter, 2003), written by Rina Dechter, one of the main players in the field of constraint programming for more than 25 years, who exploited all her past work to make her book an invaluable source

[1] This book review was handled by Peter Stuckey.

of information for both beginners and experienced CP researchers. The second one is a book by Thom Frühwirth and Slim Abdennadher (Frühwirth & Abdennadher, 2003), which focuses on recent advances on CLP based on Constraint Handling Rules, a high level formalism for defining solvers via concurrent rewriting rules. The third book is the book which is the subject of this review, whose main aim is to give a general and formal view of most of the techniques used in constraint programming.

K. R. Apt is a relative newcomer to constraint programming: after a lot of pioneering and essential work in program verification and logic programming, he started getting interested in the field of constraint programming around the mid 90's. A book on a subject written by somebody who has recently entered the subject can be one of two things: either a collection of naive statements and algorithmic descriptions, or a fundamental new way of looking at constraint programming. Apt's book falls in this second category.

The main objective of the book is two-fold. First, to give a formal proof-theoretic view of constraint propagation and search, the two main ingredients of constraint programming. Second, to use this view to define an abstract setting of function iterations over partial orders, which allows for stating very general results that can then be instantiated to the various algorithms for constraint propagation. In this way, he not only provides a uniform view of the existing techniques, but he also sets the ground for applications of this theory to any constraint-related technique that could be developed in the future. The abstract framework introduced in the book is very general, but it does not lack applicability. In fact, Apt succeeds in identifying the properties which are sufficient to make the framework usable in many practical cases. All this is done with the highest level of precision and formality, without however making the reading task too heavy, as it is typical of Apt's writing style.

The book consists of nine chapters. After a short introductory chapter, Chapter 2 provides a long list of examples of constraint problems, ranging from n-queens to cryptoarithmetic, and from temporal to Boolean constraints. For each example, variable, domains, and constraints are defined. This chapter is not too original in the list of chosen examples, considering that other constraint programming books start in a similar way. However, I cannot think of another reasonable way to start a book on constraints. Also, here the level of detail is much higher, especially in the description of temporal and qualitative constraints, and also in the analysis of polyhedral scenes.

Chapter 3 tries to convey in an informal way most the notions and results that will later be given formally in the rest of the book. This is done to allow the reader to see the overall picture. More precisely, constraint solving is described here as the iterative application of preprocessing, constraint propagation, and problem splitting (into other problems where the overall procedure is then recursively applied), until we find a solution or the current problem is too small to be split. For each part of this procedure, the informal meaning is given, as well as some examples of its execution. This chapter is very important and original, since there is a big value in seeing the whole scenario rather than a separate description of all its ingredients, as most of the other books do. So I really appreciate the effort in giving an overview of

what it means to solve a problem via constraint programming. There are some small drawbacks in doing this, but I guess they are unavoidable. In fact, there is some risk in giving a lot of information to the reader, and informally, at the beginning of the book, since this may create some confusion. For example, constraint propagation is defined here, in general, as a part of the solving procedure, but later in the book solvers are described in a way (that is, applications of domain-reduction or transformation rules) that matches the structure of the constraint propagation part, and not that of the overall solving procedure. So inexperienced readers should be warned (see description of Chapter 4 and 5). Moreover, some classes of constraints are defined here without giving all the details, and then later they are necessarily re-defined in a complete way. However, the level of precision of the book helps in avoiding getting confused after such a short introduction to constraint programming. So I really think this chapter, despite its informal character in a book which is mainly formal, has great value.

Chapter 4 focuses on examples of complete solvers. Here by a solver is meant the application of some proof rules until no rule is applicable or its application does not change anything. Completeness is meant to be the ability to generate a problem where it is easy to check if it is solvable or not. The main notions used in the rest of the book for describing constraint propagation are given here: proof rules, derivations, and problem equivalence. After such definitions, some examples of classes of proof rules, and the corresponding constraint classes for which they have been designed, is given, to show that they are complete. The first chosen example is the unification algorithm for term equations, for which a very clear and detailed description is given, that takes advantage of the exceptional logic programming background of the writer. Then there is a section on linear equations over reals with the Gauss-Jordan algorithm, where the only drawback is the need to adjust the previously given notions of substitution and unifier, and finally a section on linear inequalities over reals. These three examples of complete solvers are mainly useful, in my view, to get some familiarity with the proof-theoretic description of constraint propagation. It should now be clear to the reader that if a solver is complete, then the solving procedure of the previous chapter degenerates to just the constraint propagation phase; that is, no splitting is necessary.

Chapter 5 introduces the main local consistency notions: node, arc, and path consistency, and their directional versions, which are then all generalized to k-consistency and finally to relational consistency. For each notion, the corresponding proof rules to obtain such a level consistency are then defined. Especially useful is the presence of many examples to show the relation among all these notions in a problem. There is only one section devoted to results on tractability, and it refers to the existing relation between the width of the graph representing a CSP and the level of local consistency which is necessary to assure the existence of a solution. It is a pity that no space has been found for other tractability results which can be found in the literature.

Chapter 6 shows some incomplete solvers: the classes of constraints considered are equality and disequality constraints, Boolean constraints, linear constraints on integer intervals (for which bound consistency is introduced), arithmetic constraints on integer intervals, and arithmetic constraints on reals. The amount of information

and level of detail for each class is here at its highest degree, so that a reader interested in understanding fully what happens when using such solvers can find here all the information. For some classes, alternative solvers are also provided. As in Chapter 4, it should now be clear that incomplete solvers fit the overall solving procedure of Chapter 3 by instantiating only the constraint propagation part.

Chapter 7 describes constraint propagation algorithms as instances of some very general iteration algorithms. Such algorithms consider a set of functions and at each step apply one of such functions to an element of a partial order, starting from the bottom and stopping when stabilization is reached. At this point, the current element of the partial order is the least common fixpoint of the used functions. Three different iteration algorithms are introduced, with different features, which can be used when the functions have certain properties. If the functions commute and are monotonic and idempotent, then the first algorithm (just apply each of them once in any order) can be used; if they lack commutativity but have semi-commutativity w.r.t. a certain order of the functions, and moreover they are inflationary, then the second algorithm can be used (just apply each of them once in the chosen order). Otherwise, when the functions are not even semi-commutative, the third algorithm can be used: apply all the functions possibly repeating each of them, until a common fixpoint is reached. Although the description of the three algorithms and their properties is very abstract, it is very well written and easy to follow, and it is a joy to see how this machinery is then applied very easily to several known constraint propagation notions. After defining the partial order for CSPs and the functions to use, which are directly related to the proof-rules of the previous chapters, node consistency is shown to be able to use the first algorithm, directional consistency the second one, and k-consistency the third one. In my view, this is the best written and most original chapter of the book.

Chapter 8 is devoted to search algorithms. Starting from labeling trees where all complete instantiations are considered, smaller search trees are then introduced step by step, which represent the trees generated by using forward checking, partial look-ahead, and full-lookahead (MAC). After defining such trees, the search algorithms which build them are then defined, again starting from the simplest case of an algorithm which never backtracks to general backtracking algorithms which include constraint propagation at each tree node. Branch and bound is also defined to deal with constrained optimization. This incremental way of introducing complex search algorithms is original as far as I know, and gives an interesting new point of view on them.

The last chapter gives a brief overview of the main issues which have not found space in the other chapters of the book but which have to be considered when using a constraint language or a solver: modeling, constraint languages (mainly constraint logic programming and ILOG solver), various issues about solvers like incrementality, other forms of search like limited discrepancy search and local search, over-constrained and soft constraint problems. By looking a this list, I wish Apt had devoted more space in the book (even a full chapter each) to at least two of these issues: local search and soft constraints. Local search is a way to look for solutions (or optimal solutions) which is completely different from the search approach considered in the book, and is gaining a lot of attention lately. So it

https://doi.org/10.1017/S1471068404212182 Published online by Cambridge University Press

would have been interesting to have a precise and formal account for it. As for soft constraints, I am biased having worked on this subject for long time, but since they can be easily cast in the abstract iteration framework of this book, they would be a nice additional proof of the generality of the framework. Also, a solving approach which is not considered in this book is the one based on variable elimination and dynamic programming, and which includes techniques like adaptive consistency and bucket elimination. It would have been interesting to see it described within the formal approach of this book.

Summarizing, even after having read several books on constraint programming, and having used some of them for my courses, I really enjoyed reading Apt's book and am using it for my current course on constraint programming, mainly for two reasons. First, it is extremely well written (but this was predictable knowing Apt's writing style), so much that the formal style never makes reading a heavy task. Second, it provides a very original abstract view of the main concepts of constraint propagation and constraint programming, which can be very useful both for beginners who need to understanding these concepts without getting lost, and also for experienced researchers who may want to see where the current frontier for an abstract approach to constraint programming is and whether it can be moved further.

Moreover, for those who want to use the book in a course, Apt (as also the authors of some other books on constraint programming (Frühwirth & Abdennadher, 2003; Marriott & Stuckey, 1998)) has made wonderful Latex-generated pdf slides that cover all the material contained in the first eigth chapters of the book (see at the URL `http://homepages.cwi.nl/~apt/pcp/`). So you don't need to prepare teaching material if your students can cope with English slides.

Since the first time I met K. Apt in 1989, when I attended a Ph.D. course on logic programming that he was teaching, I have always been stunned by his exceptional ability to be formal and clear. This book is yet another witness for this ability, and it is a great present to the constraint programming community, which will certainly advance scientifically because of its publication. We need good books to educate new people to Constraint Programming in the best way, and this book is a great way to do this.

References

Apt, K. R. (2003). *Principles of constraint programming*. Cambridge University Press.

Dechter, R. (2003). *Constraint processing*. Morgan Kaufmann.

Frühwirth, T. & Abdennadher, S. (2003). *Essentials of constraint programming*. Springer.

Hentenryck, P. Van. (1989). *Constraint satisfaction in logic programming*. MIT Press.

Marriott, K. & Stuckey, P. J. (1998). *Programming with constraints: an introduction*. MIT Press.

Saraswat, V. A. (1993). *Concurrent constraint programming*. MIT Press.

Tsang, Edward P. K. (1993). *Foundations of constraint satisfaction*. Academic Press.

Francesca Rossi

Dipartimento di Matematica Pura ed Applicata

Universitá di Padova, Padova, Italy

https://doi.org/10.1017/S1471068404212182 Published online by Cambridge University Press