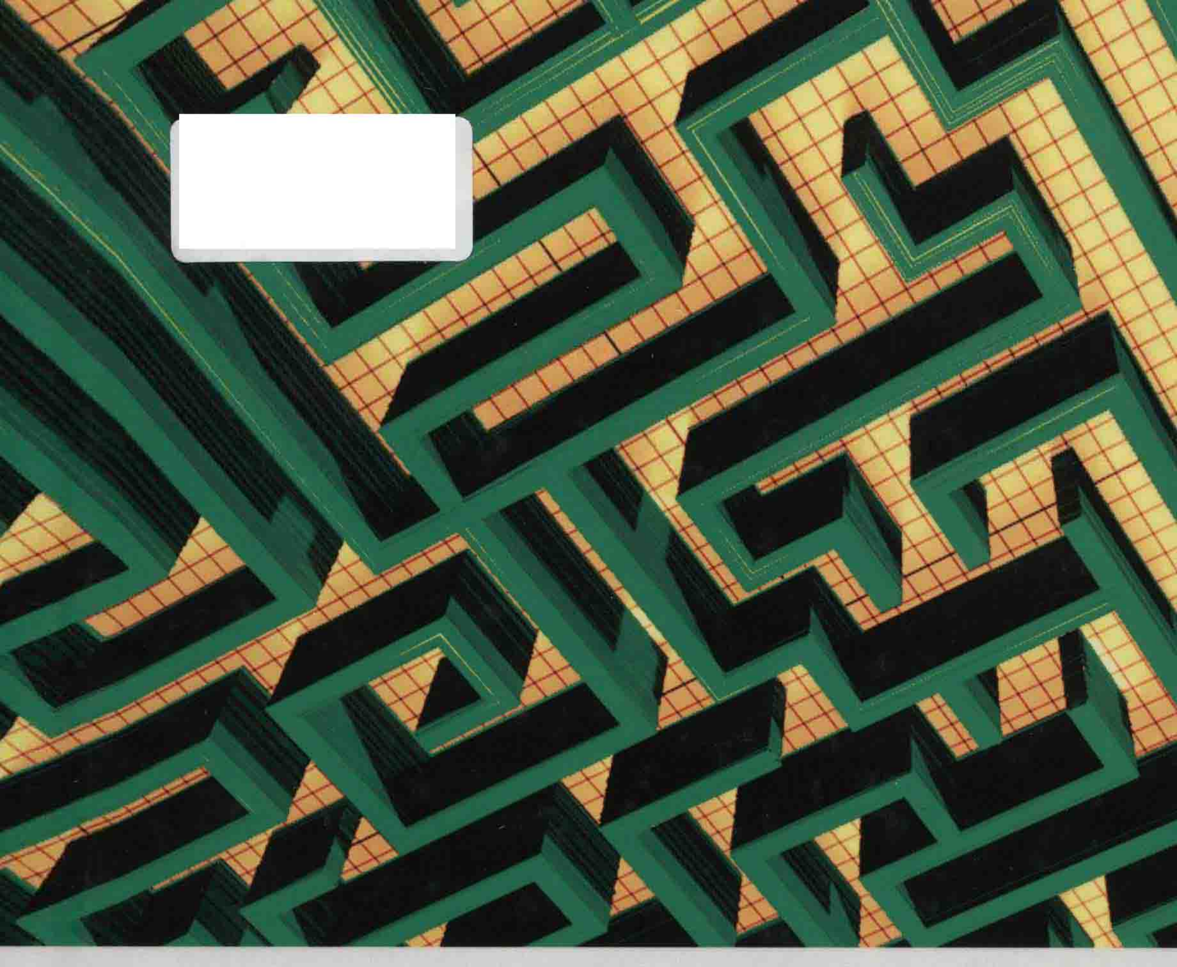

Abstract Domains in Constraint Programming

Marie Pelleau

ISTE PRESS

ELSEVIER

Series Editor
Narendra Jussien

Abstract Domains in Constraint Programming

Marie Pelleau

ELSEVIER

First published 2015 in Great Britain and the United States by ISTE Press Ltd and Elsevier Ltd

ISTE Press Ltd
27-37 St George's Road
London SW19 4EU
UK

www.iste.co.uk

Elsevier Ltd
The Boulevard, Langford Lane
Kidlington, Oxford, OX5 1GB
UK

www.elsevier.com

British Library Cataloguing in Publication Data
A CIP record for this book is available from the British Library
Library of Congress Cataloging in Publication Data
A catalog record for this book is available from the Library of Congress
ISBN 978-1-78548-010-2

Printed and bound in the UK and US

Abstract Domains in Constraint Programming

To Eliott, Joaquim and Ambrym

Preface

Constraint programming aims at solving hard combinatorial problems, with a computation time increasing in practice exponentially. Today, the methods are efficient enough to solve large industrial problems in a generic framework. However, solvers are dedicated to a single variable type: integer or real. Solving mixed problems relies on *ad hoc* transformations. In another field, abstract interpretation offers tools to prove program properties by studying an abstraction of their concrete semantics, that is, the set of possible values of the variables during an execution. Various representations for these abstractions have been proposed. They are called abstract domains. Abstract domains can mix any type of variables, and even represent relationship between the variables. In this book, we define abstract domains for constraint programming so as to build a generic solving method, dealing with both integer and real variables. We will also study the octagons abstract domain already defined in abstract interpretation. Guiding the search by the octagonal relations, we obtain good results on a continuous benchmark. Then, we define our solving method using abstract interpretation techniques in order to include existing abstract domains. Our solver, AbSolute, is able to solve mixed problems and use relational domains.

Marie PELLEAU
February 2015

Introduction

Recent advances in computer science are undeniable. Some are visible, and others are less known to the general public: today, we are able to quickly solve many problems that are known to be difficult (requiring a long computation time). For instance, it is possible to automatically place thousands of objects of various shapes in a minimum number of containers in tens of seconds, while respecting specific constraints: accessibility of goods, non-crush, etc. [BEL 07]. Constraint programming (CP) formalizes such problems using constraints that describe a result we want to achieve (accessibility of certain objects, for example). These constraints come with efficient algorithms to solve greatly combinatorial problems. In another research area, semantics, abstract interpretation (AI) attacks an insoluble problem in the general case: the correction of programs. With strong theoretical tools developed from its creation (fixed-point theorems), AI manages to prove the properties of programs. In this area, the effectiveness of methods makes it possible for impressive applications to be solved: tools in AI have, for instance, managed to prove that there was no overflow error in the flight controls of the Airbus A380 which contains almost 500,000 lines of code.

The work presented in this book is at the interface between CP and AI, two research areas in computer science with *a priori* quite different problematics. In CP, the goal is usually to obtain a good computation time for problems that are, in general, nondeterministic polynomial time (NP), or to extend existing tools to handle more problems. In AI, the goal is to analyze very large programs by capturing a maximum of properties. Despite their differences, there is a common concern in these two disciplines: identifying an impossible or difficult (computationally) space to compute precisely (the solutions set in CP and the semantics of the program in AI). It concerns computing the relevant overapproximations of this space. CP proposes methods to carefully surround this space (consistency and propagation), always with Cartesian overapproximations (boxes in $\mathbb{R}^n$ or $\mathbb{Z}^n$). AI uses often less accurate overapproximations but not only Cartesian: they may have various different shapes (not only boxes but also octagons, ellipsoids, etc.). These non-Cartesian approximations facilitate more properties to be captured.

In this book, we exploit the similarities of these overapproximation methods to integrate AI tools in the methods of CP. We redefine tools in CP from notions of AI (abstract domains). This is not only an intellectual exercise. Indeed, by generalizing the description of overapproximations, there is a significant gain in the expressiveness of CP. In particular, the problems are treated uniformly for real and integer variables, which is not currently the case. We also develop the octagon abstract domain, showing that it is possible to exploit the relationships captured by this particular domain to solve continuous problems more effectively. Finally, we perform the opposite task: we define CP as an abstract operation in AI, and develop a solver capable of handling practically all abstract domains.

I.1. Context

As mentioned before, the CP and AI have a common concern: computing efficiently and as accurately as possible an approximation of a difficult or impossible space. However, the issues and problems of these two areas are different, and hence so are their fields of application.

I.1.1. *Constraint programming*

CP, whose origins date back to 1974 [MON 74], is based on the formalization of problems such as a combination of first-order logic formulas, i.e. the constraints. A constraint defines a relationship between the variables of a problem: for example, two objects placed in the same container have an empty geometric intersection, that is to say, a heavy object should be placed under a fragile object. This is known as declarative programming. CP provides efficient generic solution methods for many combinatorial problems. Academic and industrial applications are varied: job-shop scheduling problems [GRI 11, HER 11a], design of substitution tables in cryptography [RAM 11], scheduling problems [STØ 11], prediction of the ribonucleic acid (RNA) secondary structure in biology [PER 09], optical network design [PEL 09] or automatic harmonization in music [PAC 01].

One of the limitations of the expressiveness of CP methods is that they are dedicated to the nature of the problem: solvers used for discrete variable problems are fundamentally different from techniques dedicated to continuous variable problems. In a way, the semantics of the problem is different depending on whether one deals with discrete or continuous problems.

However, many industrial problems are mixed: they contain both integer and real variables. This is, for example, the case of the problem of fast power grid repair after a natural disaster [SIM 12] to restore the power as quickly as possible in the affected areas. In this problem, we try to establish a plan of action and determine the routes that should be used by repair crews. Some of the variables are discrete; for example, each device (generator, line) is associated with a Boolean variable, indicating whether it is operational or not. Others are real, as the electrical power on a line. Another example of application is the design of the topology of a multicast transmission network [CHI 08]: we want to design a network that is reliable. A network is said to be reliable when it is still effective even when one of its components is defective, so that all user communications can pass into the network with the least possible delay. Again, some of the variables are integers (the number of lines in the network) while others are continuous (the flow of information passing over the network average).

The convergence of discrete and continuous constraints in CP is both an industrial need and a scientific challenge.

I.1.2. *Abstract interpretation*

The basis of AI was established in 1976 by Cousot and Cousot [COU 76]. AI is the theory of semantic approximation [COU 77b] in which one of the applications is programs proof. The goal is to verify and prove that a program does not contain a bug, that is to say, runtime errors. Industrial stakes are high. Indeed, many bugs have made history, such as the Year 2000 bug, or Y2K, due to system design error. On January 1, 2000, some systems showed the date of January 1, 1900. This bug may be repeated on January 19, 2038, on some UNIX systems [ROB 99]. Another example of a bug is that of the infamous inaugural flight of the Ariane 5 rocket, which, due to an error in the navigation system, caused the destruction of the rocket only 40 s after takeoff.

Every day, new softwares are being developed, corresponding to thousands or millions of lines of code. To test or verify these programs manually would require a considerable amount of time. The soundness of programs cannot be proven in a generic way; thus, AI implements methods to automatically analyze certain properties of a program. The analyzers are based on operations on the semantics of programs, that is, the set of values that can be taken by the variables of the program during its execution. By computing an overapproximation of these semantics, the analyzer can, for example, prove that the variables do not take values beyond the permitted ranges (*overflow*).

Many analyzers are developed and used for various application areas, such as aerospace [LAC 98, SOU 07], radiation [POL 06] and particle physics [COV 11].

I.2. Problematic

In this book, we focus on CP solving methods, known as *complete*, that find the solution set or prove that it is empty, if necessary. These methods are based on an exhaustive search of the space of all possible values, also called search space. Using operations to restrict the space to visit (consistency and propagation), these methods can be accelerated. Existing methods are dedicated to a certain type of variables, discrete or continuous. Facing a mixed problem, containing both discrete and continuous variables, CP offers no real solution and the techniques available are often limited. Typically, variables are artificially transformed so that they are all discrete as in the solver Choco [CHO 10], or all continuous as in the solver RealPaver [GRA 06]. In AI, analyzed programs often, if not always, contain different types of variables. Theories of AI integrate many types of domains, and helped develop analyzers uniformly dealing with discrete and continuous variables.

We propose to draw inspiration from the work of the AI community on the different types of domains to provide new solving methods in CP. These new methods should be able, in particular, to approximate with various shapes and solve mixed problems.

I.3. Outline of the book

This book is organized as follows: Chapter 1 gives the mandatory notions of AI and CP to understand our work and an analysis of the similarities and differences between these two research areas. Based on the similarities identified between CP and AI, we define abstract domains for CP in Chapter 2, with a resolution based on these abstract domains. The use of an example of abstract domain existing in AI in CP, the octagons, is detailed in Chapter 3. Chapter 4 deals with the solving method implementation details presented in Chapter 2 for octagons. Finally, Chapter 5 redefines the concepts of CP using the techniques and tools available in AI to define a method called abstract resolution. A prototype implementation, as well as experimental results, is finally presented.

I.4. Contributions

The work of this book aims to design new solving techniques for CP. There are two parts in this work. In the first part, the abstract domains are defined for CP, so as mandatory operators for the solving process. These new definitions allow us to define a uniform resolution framework that no longer depends on the variables type or on the representation of the variables values. An example of a solver using the octagon abstract domain and respecting the framework is implemented in a continuous solver Ibex [CHA 09a], and tested on examples of continuous problems. In the second part, the different CP operators needed to solve are defined in AI, allowing us to define a solving method with the existing operators in AI. This method was then implemented over Apron [JEA 09], a library of abstract domains.

Most theoretical and practical results of Chapters 2–5 are the subject of publications in conferences or journals [TRU 10, PEL 11, PEL 13, PEL 14].

Contents

PREFACE ix

INTRODUCTION xi

CHAPTER 1. STATE OF THE ART 1

1.1. Abstract interpretation 1
1.1.1. Introduction to abstract interpretation 1
1.1.2. General presentation 4
1.1.3. Conclusion 24
1.2. Constraint programming 25
1.2.1. Principles 26
1.2.2. Propagation 32
1.2.3. Exploration 40
1.2.4. Resolution scheme 41
1.2.5. Exploration strategies 43
1.2.6. Discrete/continuous comparison 46
1.2.7. Conclusion 47
1.3. Synthesis 48
1.3.1. Links between AI and CP 48
1.3.2. Analysis 50

CHAPTER 2. ABSTRACT INTERPRETATION FOR THE CONSTRAINTS 53

2.1. Introduction 53
2.2. Unified components 54
2.2.1. Consistency and fixpoint 55
2.2.2. Splitting operator 60
2.2.3. Abstract domains 63
2.3. Unified solving 64
2.4. Conclusion 68

CHAPTER 3. OCTAGONS 69

3.1. Definitions 69
3.2. Representations 72
3.2.1. Matrix representation 73
3.2.2. Intersection of boxes representation 77
3.3. Abstract domain components 79
3.3.1. Octagonal splitting operator 80
3.3.2. Octagonal precision 80
3.4. Abstract domains 82
3.5. Conclusion 84

CHAPTER 4. OCTAGONAL SOLVING 85

4.1. Octagonal CSP 85
4.2. Octagonal consistency and propagation 89
4.2.1. Octogonal consistency 89
4.2.2. Propagation scheme 90
4.3. Octagonal solver 94
4.3.1. Variables heuristics 94
4.3.2. Octogonalization heuristic 96
4.4. Experimental results 99
4.4.1. Implementation 100
4.4.2. Methodology 100
4.4.3. Results 102
4.4.4. Analysis 102
4.5. Conclusion 108

CHAPTER 5. AN ABSTRACT SOLVER: ABSOLUTE 111
5.1. Abstract solving method 112
5.1.1. Concrete solving as concrete semantics 112
5.1.2. Abstract domains existing in CP 113
5.1.3. Abstract domains operators 114
5.1.4. Constraints and consistency 117
5.1.5. Disjunctive completion and split 118
5.1.6. Abstract solving 123
5.2. The AbSolute solver 125
5.2.1. Implementation 125
5.2.2. Experimental results 133
5.3. Conclusion 137

CONCLUSION AND PERSPECTIVES 139

BIBLIOGRAPHY 143

INDEX 159