Constraint logic programming lies at the intersection of logic programming, optimisation and artificial intelligence. It has proved a successful tool for application in a variety of areas including production planning, transportation scheduling, numerical analysis and bioinformatics. Its migration from academic discipline to software development has been due in part to the availability of software systems that realise the underlying methodology; ECL^{i}PS^e is one of the leading such systems. It is exploited commercially by Cisco, and is freely available and used for teaching and research in over 500 universities.

This book has a two-fold purpose. It's an introduction to constraint programming, appropriate for a one-semester course for upper undergraduate or graduate students of computer science or for programmers wishing to master the practical aspects of constraint programming. By the end of the book, the reader will be able to understand and write constraint programs that solve complex problems.

Second, it provides a systematic introduction to the ECL^{i}PS^e system through carefully chosen examples that guide the reader through the language, illustrate its power and versatility, and clarify the gain achieved by this approach, which, ultimately allows the reader to better understand the proper use of constraints in solving real-world problems.

Krzysztof R. Apt received his PhD in 1974 in mathematical logic from the University of Warsaw in Poland. He is a senior researcher at Centrum voor Wiskunde en Informatica, Amsterdam and Professor of Computer Science at the University of Amsterdam. He is the author of three other books: *Verification of Sequential and Concurrent Programs* (with E.-R. Olderog), *From Logic Programming to Prolog*, and *Principles of Constraint Programming*, and has published 50 journal articles and 15 book chapters.

He is the founder and the first editor-in-chief of the ACM Transactions on Computational Logic, and past president of the Association for Logic Programming. He is a member of the Academia Europaea (Mathematics and Informatics Section).

After completing a degree at Oxford in Mathematics and Philosophy, Mark Wallace joined the UK computer company ICL, who funded his PhD at Southampton University, which was published as a book: *Communicating with Databases in Natural Language*.

He has been involved in the ECL^{i}PS^e constraint programming language since its inception and has led several industrial research collaborations exploiting the power of constraint programming with ECL^{i}PS^e. Currently he holds a chair in the Faculty of Information Technology at the Monash University, Victoria, Australia and is involved in a major new constraint programming initiative funded by National ICT Australia (NICTA), and in the foundation of a Centre for Optimisation in Melbourne. He has published widely, chaired the annual constraint programming conference, and is an editor for three international journals.

Advance praise for *Constraint Logic Programming using ECL^{i}PS^e*

The strength of *Constraint Logic Programming using ECL^{i}PS^e* is that it simply and gradually explains the relevant concepts, starting from scratch, up to the realisation of complex programs. Numerous examples and ECL^{i}PS^e programs fully demonstrate the elegance, simplicity, and usefulness of constraint logic programming and ECL^{i}PS^e.

The book is self-contained and may serve as a guide to writing constraint applications in ECL^{i}PS^e, but also in other constraint programming systems. Hence, this is an indispensable resource for graduate students, practioners, and researchers interested in problem solving and modelling.

Eric Monfroy, Université de Nantes

ECL^{i}PS^e is a flexible, powerful and highly declarative constraint logic programming platform that has evolved over the years to comprehensively support constraint programmers in their quest for the best search and optimisation algorithms. However, the absence of a book dedicated to ECL^{i}PS^e has presented those interested in this approach to programming with a significant learning hurdle. This book will greatly simplify the ECL^{i}PS^e learning process and will consequently help ECL^{i}PS^e reach a much wider community.

Within the covers of this book readers will find all the information they need to start writing sophisticated programs in ECL^{i}PS^e. The authors first introduce ECL^{i}PS^e's history, and then walk the reader through the essentials of Prolog and Constraint Programming, before going on to present the principal features of the language and its core libraries in a clear and systematic manner.

Anyone learning to use ECL^{i}PS^e or seeking a course book to support teaching constraint logic programming using the language will undoubtedly benefit from this book.

Hani El-Sakkout, Cisco Systems, Boston, Massachusetts

It has been recognized for some years now within the Operations Research community that Integer Programming is needed for its powerful algorithms, but that logic is a more flexible modelling tool. The case was made in most detail by John Hooker, in his book *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*.

The ECL^{i}PS^e system is a highly successful embodiment of these ideas. It draws on ideas coming from logic programming, constraint programming, and the Prolog language. I strongly recommend this book as a systematic account of these topics. Moreover, it gives a wealth of examples showing how to deploy the power thus made available via the ECL^{i}PS^e system.

Maarten van Emden, University of Victoria, Canada

This is an impressive introduction to Constraint Logic Programming and the ECL^{i}PS^e system by two pioneers in the theory and practice of CLP. This book represents a state-of-the-art and comprehensive coverage of the methodology of CLP. It is essential reading for new students, and an essential reference for practioners.

Joxan Jaffar, National University of Singapore

CONSTRAINT LOGIC PROGRAMMING USING ECL^{i}PS^e

KRZYSZTOF R. APT AND MARK WALLACE

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 2RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521866286

First published 2007

Printed in the United Kingdom at the University Press, Cambridge

A catalogue record for this publication is available from the British Library

ISBN-13 978-0-521-86628-6 hardback
ISBN-10 0-521-86628-6 hardback

Contents

Introduction

The subject of this book is constraint logic programming, and we will present it using the open source programming system ECL^{i}PS^e, available at `http://www.eclipse-clp.org`. This approach to programming combines two programming paradigms: logic programming and constraint programming. So to explain it we first discuss the origins of these two programming paradigms.[1]

Logic programming

Logic programming has roots in the influential approach to automated theorem proving based on the resolution method due to Alan Robinson. In his fundamental paper, Robinson [1965], he introduced the resolution principle, the notion of unification and a unification algorithm. Using his resolution method one can *prove* theorems formulated as formulas of first-order logic, so to get a 'Yes' or 'No' answer to a question. What is missing is the possibility to *compute* answers to a question.

The appropriate step to overcome this limitation was suggested by Robert Kowalski. In Kowalski [1974] he proposed a modified version of the resolution that deals with a a subset of first-order logic but allows one to generate a substitution that satisfies the original formula. This substitution can then be interpreted as a result of a computation. This approach became known as ***logic programming***. A number of other proposals aiming to achieve the same goal, viz. to compute with the first-order logic, were proposed around the same time, but logic programming turned out to be the simplest one and most versatile.

In parallel, Alain Colmerauer with his colleagues worked on a program-

[1] In what follows we refer to the final articles discussing the mentioned programming languages. This explains the small discrepancies in the dateline.

ming language for natural language processing based on automated theorem proving. This ultimately led in 1973 to creation of ***Prolog***. Kowalski and Colmerauer with his team often interacted in the period 1971–1973, which explains the relation between their contributions, see Colmerauer and Roussel [1996]. Prolog can be seen as a practical realisation of the idea of logic programming. It started as a programming language for applications in natural language processing, but soon after, thanks to contributions of several researchers, it was successfully transformed into a general purpose programming language.

Colmerauer, when experimenting with Prolog, realised some of its important limitations. For example, one could solve in it equations between terms (by means of unification) but not equations between strings. Using the current terminology, Prolog supports only one constraint solver. This led Colmerauer to design a series of successors, Prolog II, Prolog III, and Prolog IV. Each of them represents an advance in the logic programming paradigm towards constraint programming. In particular Prolog III, see Colmerauer [1990], included a support in the form of solving constraints over Booleans, reals and lists and can be viewed as the first realisation of constraint logic programming.

Constraint programming

Let us turn now our attention to ***constraint programming***. The formal concept of a ***constraint*** was used originally in physics and combinatorial optimisation. It was first adopted in computer science by Ivan Sutherland in Sutherland [1963] for describing his interactive drawing system Sketchpad. In the seventies several experimental languages were proposed that used the notion of constraints and relied on the concept of constraint solving. Also in the seventies, in the field of artificial intelligence (AI), the concept of a ***constraint satisfaction problem*** was formulated and used to describe problems in computer vision. Further, starting with Montanari [1974] and Mackworth [1977], the concept of ***constraint propagation*** was identified as a crucial way of coping with the combinatorial explosion when solving constraint satisfaction problems using top-down search.

Top-down search is a generic name for a set of search procedures in which one attempts to construct a solution by systematically trying to extend a partial solution through the addition of constraints. In the simplest case, each such constraint assigns a value to another variable. Common to most top-down search procedures is ***backtracking***, which can be traced back to the nineteenth century. In turn, the ***branch and bound search***, a

top-down search concerned with optimisation, was defined first in the context of combinatorial optimisation.

In the eighties the first constraint programming languages of importance were proposed and implemented. The most significant were the languages based on the logic programming paradigm. They involve an extension of logic programming by the notion of constraints. The main reason for the success of this approach to constraint programming is that constraints and logic programming predicates are both, mathematically, relations; backtracking is automatically available; and the variables are viewed as unknowns in the sense of algebra. The latter is in contrast to the imperative programming in which the variables are viewed as changing, but each time known entities, as in calculus.

The resulting paradigm is called ***constraint logic programming***. As mentioned above, Prolog III is an example of a programming language realising this paradigm. The term was coined in the influential paper Jaffar and Lassez [1987] that introduced the operational model and semantics for this approach and formed a basis for the CLP($\mathcal{R}$) language that provided support for solving constraints on reals, see Jaffar et al. [1992].

Another early constraint logic programming language is CHIP, see Dincbas et al. [1988] and for a book coverage Van Hentenryck [1989]. CHIP incorporated the concept of a constraint satisfaction problem into the logic programming paradigm by using constraint variables ranging over user-defined finite domains. During the computation the values of the constraint variables are not known, only their current domains. If a variable domain shrinks to one value, then that is the final value of the variable. CHIP also relied on top-down search techniques originally introduced in AI.

The language was developed at the European Computer-Industry Research Centre (ECRC) in Munich. This brings us to the next stage in our historical overview.

ECL^{i}PS^e

ECRC was set up in 1984 by three European companies to explore the development of advanced reasoning techniques applicable to practical problems. In particular three programming systems were designed and implemented. One enabled complex problems to be solved on multiprocessor hardware, and eventually on a network of machines. The second supported advanced database techniques for intelligent processing in data-intensive applications. The third system was CHIP. All three systems were built around a common foundation of logic programming.

In 1991 the three systems were merged and ECL^{i}PS^e was born. The constraint programming features of ECL^{i}PS^e were initially based on the CHIP system, which was spun out from ECRC at that time in a separate company. Over the next 15 years the constraint solvers and solver interfaces supported by ECL^{i}PS^e have been continuously extended in response to users' requirements.

The first released interface to an external state-of-the-art linear and mixed integer programming package was in 1997. The integration of the finite domain solver and linear programming solver, supporting hybrid algorithms, came in 2000. In 2001 the `ic` library was released. It supports constraints on Booleans, integers and reals and meets the important demands of practical use: it is sound, scalable, robust and orthogonal. ECL^{i}PS^{e}also includes as libraries some constraint logic programming languages, for example CLP($\mathcal{R}$), that were developed separately. By contrast with the constraint solving facilities, the parallel programming and database facilities of ECL^{i}PS^e have been much less used, and over the years some functionality has been dropped from the system.

The ECL^{i}PS^e team was involved in a number of European research projects, especially the Esprit project CHIC – Constraint Handling in Industry and Commerce (1991–1994). Since the termination of ECRC's research activities in 1996, ECL^{i}PS^e has actively been further developed at the Centre for Planning and Resource Control at Imperial College in London (IC-Parc), with funding from International Computers Ltd (ICL), the UK Engineering and Physical Sciences Research Council, and the Esprit project CHIC-2 – Creating Hybrid Algorithms for Industry and Commerce (1996–1999). The Esprit projects played an important role in focussing ECL^{i}PS^e development. In particular they emphasised the importance of the end user, and the time and skills needed to learn constraint programming and to develop large scale efficient and correct programs.

In 1999, the commercial rights to ECL^{i}PS^e were transferred to IC-Parc's spin-off company Parc Technologies, which applied ECL^{i}PS^e in its optimisation products and provided funding for its maintenance and continued development. In August 2004, Parc Technologies, and with it the ECL^{i}PS^e platform, was acquired by Cisco Systems.

ECL^{i}PS^e is in use at hundreds of institutions for teaching and research all over the world, and continues to be freely available for education and research purposes. It has been exploited in a variety of applications by academics around the world, including production planning, transportation scheduling, bioinformatics, optimisation of contracts, and many others. It is also being used to develop commercial optimisation software for Cisco.

Overview of the book

In this book we explain constraint logic programming using ECL^{i}PS^e. Since ECL^{i}PS^e extends Prolog, we explain the latter first. The reader familiar with Prolog may skip the first five chapters or treat them as a short reference manual.

In Part I of the book we focus on the logic programming paradigm. We begin by introducing in **Chapter 1** a subset of Prolog called pure Prolog and explaining the underlying computation model. In **Chapter 2** we clarify the programming features implicitly supported by pure Prolog by discussing a toy procedural programming language and explaining how pure Prolog can be translated into it.

Part II is devoted to a short exposition of Prolog. In **Chapter 3** we explain the Prolog approach to arithmetic. In particular, arithmetic constraints are allowed, but only for testing. We call constraints used in this way ***passive***. In **Chapter 4** we discuss control in Prolog. Also, we explain there how Prolog supports meta-programming, i.e., the possibility of writing programs that use other programs as data. Next, in **Chapter 5** we explain how Prolog allows us to inspect, compare and decompose terms. This exposition of Prolog is incomplete: the standard Prolog has 102 built-ins and we cover only those we need later.[2]

Part III begins with a primer on constraint programming, provided in **Chapter 6**. We introduce here the concepts of constraints, constraint satisfaction problems (CSPs) and constraint optimisation problems (COPs), and discuss the basic techniques used to solve CSPs and COPs. Then in **Chapter 7** we return to ECL^{i}PS^e by introducing a large suite of iterators that allow us to write programs without recursion. They are extensively used in the remainder of the book. Next, in **Chapter 8** we study the top-down search in presence of passive constraints by considering search algorithms for both complete and incomplete search.

One of the limitations of Prolog is that arithmetic constraints, when selected too early cause a run-time error. In **Chapter 9** we introduce the `suspend` library of ECL^{i}PS^e that allows us to circumvent this problem by automatically delaying various types of constraints, including arithmetic constraints, until they reduce to tests.

Part IV forms the main part of the book. We begin by introducing in **Chapter 10** two ECL^{i}PS^e libraries, `sd` and `ic`, that treat constraints as ***active constraints***, by allowing their evaluation to affect their variables.

[2] For those wishing to acquire the full knowledge of Prolog we recommend Bratko [2001] and Sterling and Shapiro [1994].

This is achieved through the process of ***constraint propagation***. In **Chapter 11** we explain how to combine the constraint propagation with ECL^{i}PS^e facilities that support top-down search. Next, in **Chapter 12** we discuss the `branch_and_bound` library that allows us to solve COPs using ECL^{i}PS^e.

The last two chapters deal with constraints on continuous variables. In **Chapter 13** we explain how the approach to solving CSPs and COPs based on the constraint propagation and top-down search can be modified to deal with such constraints. Finally, in **Chapter 14** we consider linear and non-linear constraints over continuous and integer variables. In ECL^{i}PS^e they are solved using the `eplex` library that provides an interface to a package for solving mixed integer programming problems.

Resources on ECL^{i}PS^e

The main website of ECL^{i}PS^e is `www.eclipse-clp.org`. It provides a substantial body of documentation to aid users in getting the most out of the system.

There are a tutorial and three manuals: a User Manual, a Reference Manual and a Constraint Library Manual. There are documents about ECL^{i}PS^e embedding and interfacing, and about applications development in ECL^{i}PS^e. Also there are example ECL^{i}PS^e programs for solving a variety of puzzles and applications in planning and scheduling. Finally, there is a great deal of on-line help available, covering all (currently 2644) built-in predicates.

The programs presented in this book can be found at `www.cwi.nl/~apt/eclipse`.

Acknowledgements

Krzysztof Apt would like to thank his colleagues who during their stay at CWI used ECL^{i}PS^e and helped him to understand it better. These are: Sebastian Brand, Sandro Etalle, Eric Monfroy and Andrea Schaerf. Also, he would like to warmly thank three people who have never used ECL^{i}PS^e but without whose support this book would never have been written. These are: Alma, Daniel and Ruth.

Mark Wallace offers this book as a testament both to the ECRC CHIP team, who set the ball rolling, to Micha Meier at ECRC, and to the IC-Parc ECL^{i}PS^e team, including Andy Cheadle, Andrew Eremin, Warwick Harvey, Stefano Novello, Andrew Sadler, Kish Shen, and Helmut Simonis – of both the CHIP and ECL^{i}PS^e fame – who have designed, built and

maintained the functionality described herein. Last but not least, he thanks Joachim Schimpf, a great colleague, innovative thinker and brilliant software engineer, who has shaped the whole ECL^{i}PS^e system.

Since dragging his reluctant family halfway round the world to Australia, Mark has buried himself in his study and left them to it. With the completion of this book he looks forward to sharing some proper family weekends with Duncan, Tessa and Toby and his long-suffering wonderful wife Ingrid.

The authors acknowledge helpful comments by Andrea Schaerf, Joachim Schimpf, Maarten van Emden and Peter Zoeteweij. Also, they would like to thank the School of Computing of the National University of Singapore for making it possible for them to meet there on three occasions and to work together on this book. The figures were kindly prepared by Ren Yuan using the `xfig` drawing program.

To Alma, Daniel and Ruth and
to Duncan, Tessa, Toby and Ingrid