SpringerBriefs in Intelligent Systems

Artificial Intelligence, Multiagent Systems, and Cognitive Robotics

More information about this series at http://www.springer.com/series/11845

Neng-Fa Zhou · Håkan Kjellerstrand
Jonathan Fruhman

Constraint Solving and Planning with Picat

Springer

Neng-Fa Zhou
Department of Computer and Information Science
Brooklyn College
Brooklyn, NY
USA

Jonathan Fruhman
Independent Researcher
New York, NY
USA

Håkan Kjellerstrand
hakank.org
Malmö
Sweden

ISSN 2196-548X ISSN 2196-5498 (electronic)
SpringerBriefs in Intelligent Systems
Artificial Intelligence, Multiagent Systems, and Cognitive Robotics
ISBN 978-3-319-25881-2 ISBN 978-3-319-25883-6 (eBook)
DOI 10.1007/978-3-319-25883-6

Library of Congress Control Number: 2015954588

Springer Cham Heidelberg New York Dordrecht London

Printed on acid-free paper

Springer International Publishing AG Switzerland is part of Springer Science+Business Media (www.springer.com)

Foreword

Logic programming languages entered the scene of computer science in the early 1970s as the answer to the need for paradigms capable of representing and reasoning about different kinds of knowledge. The big picture that was pursued by logic programming researchers was to create a black-box system that is able to transform declarative and logic-based specifications into actionable problem solutions ("the holy grail of programming," as some authors say). The history of logic programming witnessed periods of genuine enthusiasm and leaps of knowledge, and also witnessed periods of stagnation. The initial proposal of the language Prolog by Robert Kowalski came with the drawback of the inefficiency of its first implementation of the SLD resolution procedure. As a consequence, the AI community, looking for fast implementations of planning algorithms, moved towards other directions—from the (functionally inspired) PDDL modeling language, which is the de facto standard formalism for planning problem descriptions, to the use of traditional imperative languages.

In the early 1980s, results from unification theory and the implementation of the Warren Abstract Machine by D.H.D. Warren sensibly improved the efficiency of the Prolog implementations, enabling the compilation of programs into efficiently executable bytecode. In those same years, the AI problem-solving community developed the notion of constraint propagation and constraint-based search; these declarative formalisms required a host language in order to be effectively exploited. Researchers in the US (e.g., Joxan Jaffar, Jan-Louis Lassez, Michael Maher, and Peter Stuckey) and in Europe (e.g., Pascal Van Hentenryck), on the wave of the Fifth Generation project, offered effective frameworks and implementations to enable the parametric extension of the Prolog language into a constraint-based framework. Constraint logic programming is a class of logic programming languages whose modeling capabilities are based on the external constraint solver chosen by the programmer (e.g., a solver on finite domains, on rational numbers, or on sets). This extension of Prolog allowed the efficient resolution of large classes of practical problems, and constituted a common research path for the logic programming and constraint programming communities.

However, logic programming and constraint logic programming continued to fall short of expectations in the context of knowledge representation and reasoning. The original declarative semantics that were developed for Prolog and preserved by constraint logic programming were struggling to meet the needs of non-monotonic reasoning. An elegant and effective solution to this problem arose from the work of Michael Gelfond and Vladimir Lifschitz, which was embodied by the stable model semantics for logic programming with negation as failure. It took almost 10 years for researchers to understand how to effectively embed such semantics in a Prolog-style language, leading to a novel paradigm, commonly referred to as answer set programming (ASP). Planning problems are naturally encoded in ASP, and can also be directly encoded as a SAT formula, exploiting the progress in this area. In this millennium, the idea of conflict-driven learning has been introduced in both ASP solvers and SAT solvers, making these systems highly effective for solving large and complex problems. In parallel with the evolution of ASP, the notion of tabling/memoization was introduced by D.S. Warren, and was implemented in Prolog systems (including B-Prolog by Neng-Fa Zhou). Tabling allows Prolog systems to avoid the recomputation of sub-goals, and can also be used for supporting dynamic programming-style optimizations. A good use of tabling has been proven effective for speeding up the search, particularly in solving planning problems.

The language Picat, the subject of this book, is the culminating event of these developments. The language is as declarative as Prolog, but it is more convenient than Prolog in many aspects. It supports the encoding of problems by using constraints, and it enables the search for solutions through the use of constraint and SAT solvers. Picat provides the use of tabling with all of its features. In particular, Picat allows the encoding of planning problems in a programming style that is similar to PDDL, and facilitates their fast resolution thanks to tabling.

I do not know if Picat is already "the holy grail," but surely a generation of students/programmers can benefit a lot from this language. This book provides the perfect compendium to enter the fascinating universe of Picat.

Udine Agostino Dovier
September 2015

Preface

Many complex systems, ranging from social, industrial, economics, financial, educational, to military, require that we obtain high-quality solutions to combinatorial problems. Linear programming and its extensions developed in operations research once formed the primary paradigm for solving combinatorial problems. During the last three decades, a plethora of paradigms have been developed for combinatorial problems, including constraint programming (CP), propositional satisfiability testing (SAT), satisfiability modulo theories (SMT), answer set programming (ASP), tabled logic programming, and heuristic search planners.

Picat is a new logic-based multi-paradigm programming language that integrates logic programming, functional programming, dynamic programming with tabling, and scripting. Picat provides facilities for solving combinatorial search problems, including solver modules that are based on CP, SAT, and MIP (mixed integer programming), and a module for planning that is implemented by the use of tabling. Chapter 1 gives an overview of the Picat language and system.

This book presents Picat as a modeling and solving language for two important classes of combinatorial problems: constraint satisfaction and planning. The constraint satisfaction problem (CSP) is a basic class of combinatorial problems. A CSP consists of a set of variables, each of which is defined over a domain, a set of constraints among the variables, and, optionally, an objective function. A solution to a CSP is a valuation of the variables that satisfies all the constraints and optimizes the objective function, if it exists. Chapters 2 and 3 are devoted to constraint modeling. Chapter 2 introduces the basic built-in constraints in the common API of the three solver modules (cp, sat, and mip), and gives several simple example models that use these constraints. Chapter 3 describes the more sophisticated constraints in the API and gives "real-world" example models for scheduling, resource allocation, and design.

Planning is an important task for building many systems, such as autonomous robots, industrial design software, system security, and military operational systems. Planning is also closely related to model checking. Given an initial state, a set of goal states, and a set of possible actions, the classic planning problem is to find a

plan that transforms the initial state to the goal state. Chapters 5 and 6 are devoted to planning. Chapter 5 introduces depth-unbounded search predicates in the planner module, and gives models for several planning puzzles. Chapter 6 introduces depth-bounded search predicates in the planner module, and shows how domain knowledge and heuristics can be incorporated into planning models to improve the performance. Because planning is solved as a dynamic programming problem with tabling in Picat, a separate chapter (Chap. 4) is included to introduce the tabling feature of Picat.

A combinatorial problem can normally be modeled in different ways and solved by different solvers. The book is wrapped up with a chapter (Chap. 7) that gives several models for solving the Traveling Salesman Problem with different solvers, including CP, SAT, MIP, and tabled planning.

This book is useful for students, including undergraduate-level students, researchers, and practitioners, to learn the modeling techniques for Picat. No prerequisite knowledge about Picat is required, although familiarity with logic or functional programming is a plus. Chapter 1 gives an overview of the data types, built-ins, and language constructs that are needed for the later chapters. For readers who are familiar with Prolog, Haskell, or Python, this chapter also compares Picat with these three languages.

This book does not cover the implementation of Picat. The bibliographical note at the end of each chapter aims to meet the reader's curiosity about how the presented tools are built. Each chapter ends with a set of exercises, which is intended for the reader to practice the presented modeling techniques. The code examples used in the book are available at: http://picat-lang.org/picatbook2015.html.

Neng-Fa Zhou would like to thank his other co-authors for their collaboration on some of the important ideas that underpin the Picat implementation: Roman Barták, Agostino Dovier, Christian Theil Have, Taisuke Sato, and Yi-Dong Shen; his recent students at CUNY who worked on application projects using Picat: Mike Bionchik and Lei Chen; his colleagues at CUNY who have been involved in some way in the Picat project: David Arnow, James Cox, Danny Kopec, and Subash Shankar. The authors would like to thank Ronan Nugent, the Springer editor, for his helpful comments and advice, and the following people who gave us permission to use their examples: Brian Dean, Christopher Jefferson, Tony Hürlimann, and Peter James Stuckey.

Brooklyn, NY, USA — Neng-Fa Zhou
Malmö, Sweden — Håkan Kjellerstrand
New York, NY, USA — Jonathan Fruhman
August 2015

Contents