Cooperation and Coordination of Constraint Solvers

Dissertation

zur Erlangung des akademischen Grades Doktor rerum naturalium
(Dr. rer. nat.)

vorgelegt an der

Technischen Universität Dresden

Fakultät Informatik

eingereicht von

Dipl.-Inform. Petra Hofstedt

geb. am 22. März 1971 in Halle/Saale

Betreuender Hochschullehrer: Prof. Dr.-Ing. habil. Heiko Vogler

Dresden, im März 2001

Promotionskommission:

Vorsitzender:	Prof. Dr. rer. nat. Hermann Härtig	(TU Dresden)
1. Gutachter und Prüfer:		
	Prof. Dr.-Ing. habil. Heiko Vogler	(TU Dresden)
2. Gutachter:	Prof. Dr. rer. nat. habil. Peter Pepper	(TU Berlin)
3. Gutachter:	Prof. Dr. rer. nat. habil. Michael Thielscher	(TU Dresden)
Prüfer:	Prof. Dr. rer. nat. habil. Peter Buchholz	(TU Dresden)
Weiteres Mitglied:	Prof. Dr. rer. nat. Oliver Deussen	(TU Dresden)

Tag der Promotion: 2. Juli 2001

Berichte aus der Informatik

Petra Hofstedt

Cooperation and Coordination of Constraint Solvers

Shaker Verlag
Aachen 2001

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Hofstedt, Petra:
Cooperation and Coordination of Constraint Solvers / Petra Hofstedt.
Aachen : Shaker, 2001
(Berichte aus der Informatik)
Zugl.: Dresden, Techn. Univ., Diss., 2001
ISBN 3-8265-9351-0

ISBN 3-8265-9351-0
ISSN 0945-0807

Shaker Verlag GmbH • Postfach 1290 • 52013 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • eMail: info@shaker.de

Zusammenfassung

Deklarativen Programmiersprachen liegt die Idee zugrunde, daß die Formulierung von Programmen nah an der Problemspezifikation und -domain möglich sein sollte. Programme deklarativer Sprachen bestehen meist aus mathematischen Objekten, d.h. Relationen und Funktionen. Dementsprechend unterscheidet man deklarative Programmiersprachen gewöhnlich in logische, funktionale, funktional-logische und Constraint-Programmiersprachen.

Constraint-Programmierung ist eine relativ junge Entwicklung auf dem Gebiet deklarativer Sprachen. Gleichwohl gibt es inzwischen eine große Menge von Constraint-Programmiersprachen, ebenso wurden viele Constraint-Lösungsmechanismen ausführlich untersucht. In der Constraint-Programmierung werden Probleme mittels Constraints spezifiziert. Ein Constraint, zu deutsch Bedingung oder Einschränkung, ist dabei eine Formel, die Eigenschaften von Objekten oder Beziehungen zwischen diesen ausdrückt. Programme, in denen Probleme mittels Constraints beschrieben sind, können mit sog. Constraint-Lösern behandelt werden. Das sind Algorithmen die auf Probleme spezieller Bereiche, d.h. spezieller Domains, anwendbar sind. Constraint-Löser erlauben meist, Constraints auf Erfüllbarkeit zu testen, d.h. sie zu lösen, sie erlauben, das Enthaltensein von Constraints in anderen zu prüfen und, teilweise sogar, passende Variablenzuweisungen zu finden, so daß die Constraints, die das Problem beschreiben, erfüllt sind.

Obwohl das Paradigma der Constraint-Programmierung effiziente Mechanismen zur Behandlung von Constraints vieler Domains anbietet, kann ein Constraint-Löser gewöhnlich nur Constraints seiner ganz speziellen Domain behandeln. Daher ist es oft wünschenswert und es wäre vorteilhaft, Constraint-Systeme kombinieren und die entsprechenden Constraint-Lösungs-Mechanismen kooperieren lassen zu können, weil damit die Behandlung und Lösung von Problemen möglich wird, die jeweils einzelne Constraint-Löser nicht behandeln können.

In dieser Arbeit stellen wir ein allgemeines Schema für die Kooperation von Constraint-Lösern vor. Die Basis unseres Systems ist dabei eine einheitliche Schnittstelle für Constraint-Löser, die eine fein-granulare for-

male Spezifikation des Informationsaustausches zwischen Constraint-Lösern erlaubt. Darauf aufbauend entwickeln wir einen offenen und sehr flexiblen Kombinationsmechanismus für Constraint-Löser. Wir bezeichnen diesen Kombinationsmechanismus als offen, da er eine einfache Integration von Constraint-Lösern in das Gesamtsystem unabhängig von ihrer Domain und Implementierungssprache erlaubt. Der Kombinationsmechanismus ist flexibel, da er die Definition vieler verschiedener Kooperationsstrategien der integrierten Löser in einfacher Weise ermöglicht.

Wir schlagen einen allgemeinen Rahmen für die Definition des Verhaltens eines Systems kooperierender Constraint-Löser vor und zeigen seine Verwendung für die schrittweise Definition von Reduktionssystemen, die Kooperationsstrategien eines solchen Systems beschreiben. Die Modularität unserer Definitionen von Reduktionsrelationen auf verschiedenen Ebenen erlaubt, entsprechend der aktuellen Situation und Ressourcen eine große Anzahl von Kooperationsstrategien für die Löser zu definieren, so daß unser Gesamtsystem ein allgemeines Framework für kooperierende Constraint-Löser darstellt.

Abstract

Declarative programming languages base on the idea that programs should be as close as possible to the problem specification and domain. Programs of these languages usually consist of directly formulated mathematical objects, i.e. relations and functions. Accordingly, declarative languages are distinguished into logic, functional, functional-logic, and constraint programming languages.

Even if constraint programming is a young development in declarative programming languages, meanwhile, there is a wide range of constraint programming languages, and many domains and associated constraint solving mechanisms have been investigated. Using constraint programming, problems are specified by means of constraints. A constraint is usually a first order formula which describes properties of objects or relations between them. Problems described by constraints are handled by constraint solvers, i.e. specific algorithms for particular domains which mostly allow to check satisfiability of constraints, i.e. to solve constraints, to compute entailed constraints, and even to find variable assignments such that the constraints specifying the problem are satisfied.

Though the paradigm of constraint programming offers efficient mechanisms to handle constraints of various domains, one constraint solver can only handle constraints of a particular constraint system. Thus, it is often desirable and advantageous to combine several constraint systems and the associated constraint solving techniques because this combination makes it possible to solve problems that none of the single solvers can handle alone.

In this work, we propose a general scheme for the cooperation of constraint solvers. The base of our system is a uniform interface for constraint solvers which allows a fine-grain formal specification of information exchange between constraint solvers. On top of this interface, we develop an open and flexible combination mechanism for constraint solvers. The combination is open in the sense that whenever a new constraint system with associated constraint solver (satisfying certain properties) arises, it can be easily incorporated into the overall system independently of its domain and the language in which the constraint solver is implemented. It is flexible because

the definition of different strategies for the cooperation of the single solvers is possible in a simple way.

We propose a general frame for the definition of the behaviour of a system of cooperating constraint solvers. Using this frame, we develop in a stepwise fashion reduction systems which describe strategies of such an overall system of cooperating solvers. The modularity of our definitions of reduction relations at different levels allows to define a wide range of cooperation strategies for the solvers according to the current situation, resources, and requirements such that our overall system forms a general framework for cooperating solvers.

Acknowledgements

It is my special pleasure to thank the many people who have contributed in some way to the work which is presented here.

I am indebted to my supervisor Heiko Vogler, who gave me the opportunity to work on my chosen subject, much guidance and support. He taught me how to improve my work and encouraged me to report the results of my research. Last but not least he provided a very stimulating and convenient work environment. I am very grateful to Peter Pepper whose technical and social support was very helpful and who opened my mind to different new points of view. I like to thank Michael Hanus, Herbert Kuchen, and Klaus Indermark for their hospitality during my visits to the group 'Programming Languages' of the Aachen University of Technology (RWTH Aachen) and for the many discussions on declarative languages, in particular on functional and functional-logic ones, which proved very important for my research. I would like to thank all members of the group 'Foundations of Programming' of the Dresden University of Technology, in particular my office mate Armin Kühnemann, the past and present members of the group 'Programming Languages' of the RWTH Aachen, and of the 'Compiler Construction and Programming Language' group at Technical University of Berlin for many stimulating discussions and encouragement. As well I thank the members of the Postgraduate Programmes (Graduiertenkollegs) 'Werkzeuge zum effektiven Einsatz paralleler und verteilter Rechnersysteme' and 'Spezifikation diskreter Prozesse und Prozeßsysteme durch operationelle Modelle und Logiken' of the Dresden University of Technology, in particular Hans-Ulrich Karl, for their support and the many discussions. I am grateful to Monika Sturm who initially got me interested in declarative languages and encouraged me in my work. I thank Stephan Frank for careful reading of this thesis and the many helpful suggestions.

I am indebted to my parents and my brother in many ways. They always trusted in me and supported me in all decisions I have ever taken, even when they disagreed. Their love, support, and comprehension have made it possible for many good things to happen, including finishing this thesis.

Finally, I would like to thank my friends for standing by me.

Contents

List of Figures