Rule-based constraint propagation: theory and applications
Brand, S.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 1

Introduction

The notion of the constraint satisfaction problem (CSP) provides a general framework for formulating problems. In this framework, a problem solution corresponds to a variable assignment. The problem context is formalised by stating the range of values the variables can assume, and by specifying for some subsets of the variables which combinations of their values are acceptable — in other words, which constraints must be met.

A large variety of problems can be modelled as CSPs, and in many cases the models are very natural. Consider the graph colouring problem: we wish to colour the vertices of a graph in such a way that connected vertices differ in their colour. In a straightforward formulation as a CSP, each vertex corresponds to a variable ranging over the colours, and two connected vertices give rise to a disequality constraint on the respective variables. The term unification problem can be viewed as a CSP in which the variables range over a term universe and are constrained by equalities. A propositional formula in conjunctive normal form can be seen as a CSP by regarding each clause as a constraint on its Boolean variables. More generally, the CSP framework is applicable to problems in many areas, including Artificial Intelligence (temporal and spatial reasoning, computer vision, planning, computational logic), Operations Research (scheduling, time-tabling, resource allocation), and Bio-informatics (protein structure reconstruction, sequence alignment).

The expressiveness of the CSP framework has consequences for the solving algorithms. Solving CSPs with finite variable domains is NP-complete in general, and one should therefore not expect computationally tractable algorithms. A general method to find solutions of a CSP consists of search, that is, the CSP is split into subproblems which are considered separately. For instance, every assignment of a variable to a domain value induces a subproblem.

In the constraint programming approach to solving CSPs, search is combined with constraint propagation to reduce the search space. The principle of constraint propagation is controlled inference: from the available constraints and domains,
certain new constraints or smaller domains are inferred. In this way, a CSP is transformed by making selected implicit information explicit. Usually, the result of propagation is characterised as a form of local consistency. Constraint propagation is often a very cost-effective method to reduce the problem solving time, insofar as more time is saved in search than spent on propagation.

The question that arises is how constraint propagation can be described and implemented. Conventionally, this is done by generic or specialised constraint propagation algorithms, implemented in an imperative programming style. Generic algorithms, by definition, do not take into account the structure of specific constraints. Constraint-specific algorithms, on the other hand, typically require considerable expertise in the development of constraint propagation algorithm and are hard to understand or verify.

In this thesis, we advocate a rule-based view on constraint propagation.

**Rule-Based Constraint Propagation**

Rule-based programming means the formulation of programs in terms of rules, i.e. premise–conclusion pairs. Such programs are executed by a repeated application of the rules. Hence, rule-based programming is declarative: the program logic is separated from the control of the execution. The interest in rule-based computation goes back at least to the 1970s, when production rule systems were extensively studied in Artificial Intelligence.

We apply the rule-based paradigm to constraint propagation and consequently consider constraint propagation rules. Our notion of a constraint propagation rule is very basic:

\[ A \rightarrow B \]

is a constraint propagation rule if \( A, B \) are sets of constraints. Here are some examples of such rules:

\[
\begin{align*}
  x > y, \ y > z & \rightarrow x > z \quad (r_1) \\
  \text{and}(x, y, z), \ \text{or}(y, z, w), \ w = 1 & \rightarrow y = 1, \ x = z \quad (r_2) \\
  \text{rcc8}(x, y, z), x \in \{\text{disjoint, inside}\}, z \in \{\text{contains, equal}\} & \rightarrow y \neq \text{covers} \quad (r_3)
\end{align*}
\]

The rule \( r_1 \) captures transitivity of the ordering relation \( > \) viewed as a constraint. \( r_2 \) propagates a fact about the constraints and and or which model the respective logical operators. Spatial knowledge is expressed in rule \( r_3 \): \( x, y, z \) are the topological relations of the region pairs \( (A, B), (B, C), (A, C) \), respectively.

The formulation of constraint propagation in terms of rules offers several advantages. Since propagation rules are declarative, the correctness of the constraint propagation step represented by a rule can be verified directly and per rule with the definitions of the involved constraints. Rules represent directed knowledge, and in that way they control inference. The local consistency established by a set of rules can be examined independently of the concrete rule scheduling algorithm.
Contributions and Overview

We argue in this dissertation that a rule-based approach to constraint propagation is useful for both explaining and implementing it. We do so by paying attention to theoretical aspects of rule-based constraint propagation as well as to applications in constraint programs. When discussing the applications, in line with the constraint programming approach to problem solving, we focus especially on modelling declaratively. In detail, we discuss the following topics.

The three chapters following the introductory Chapter 2 on rule-based constraint programming are devoted to problem-independent issues involving constraint propagation rules. In all cases, we discuss in detail a class of constraint propagation rules of particular interest, the membership rules.

Schedulers for constraint propagation rules. In Chapter 3, we consider the problem of computing with constraint propagation rules. We start from the completely general view of constraint propagation as fixpoint computation of functions, and review a corresponding generic iteration algorithm.

We revise this algorithm with a dynamic modification of the set of iterated functions. Specifically, we provide conditions for the removal of functions from this set, so as to improve convergence of the fixpoint computation. The benefit of this technique is multiplied if one deals with sequences of fixpoint computations, as is the case in constraint programming in which constraint propagation is executed repeatedly. A dynamic reduction in the function set then helps convergence in all later computations.

By implementing the revised iteration algorithm for concrete sets of membership rules, we demonstrate the viability of this way of performing constraint propagation. Furthermore, by an empirical evaluation we find that the revised rule scheduler performs very well in comparison with the generic scheduler as well as with the scheduler used in an implementation of CHR, a language specifically designed for rule-based constraint propagation.

This chapter is based on a collaboration with Krzysztof Apt, which appeared as [Apt and Brand, 2003]. A combination with the following chapter will appear as [Brand and Apt, 2005].

Redundancy in constraint propagation rule sets. In Chapter 4, we turn to the question whether each propagation rule in a set of rules is needed for the result of propagation. A natural characterisation of the local consistency established by a set of rules is based on their common fixpoints (where rules are viewed as functions in an abstract setting). Consequently, we formulate the notion of redundancy of a rule with respect to a rule set as follows: removing a redundant rule from the set does not change the common fixpoints. This leads to the notion of a minimal rule set, which contains no redundant rules.
We also investigate rule redundancy empirically, with the help of an implementation of minimisation. In recent years, a number of methods for the automatic generation of classes of constraint propagation rules have been published. While all of these rule generation methods strive to generate rule sets that are minimal in a sense, they fall short on rigour or generality. By processing a number of concrete rule sets generated by such methods, we find that many of the sets are not minimal. We provide here a redundancy notion that is theoretically well-founded, comprehensive, and feasible.

This chapter is an extended version of [Brand, 2003].

**Incremental generation of constraint propagation rules.** In Chapter 5, we approach the problem of generating propagation rules *incrementally*. By this, we mean automatic rule generation as transformations of rule sets into rule sets. One example is the combination of two rules to a new rule: \( c_1 \rightarrow B \) and \( c_2 \rightarrow B \) leads to \( c_1 \lor c_2 \rightarrow B \). The crucial requirement is that the disjunctive constraint \( c_1 \lor c_2 \) must be representable in the underlying language of the considered constraint propagation rules.

We then study incremental rule generation for the specific language of the membership rules. We regard rule sets associated with constraints, and consider the following question: suppose it is known how some given constraints relate to each other, then how do their associated rule sets relate to each other? The relations of constraints we are interested in are incremental constraint definitions, for example, separate constraints versus their conjunction.

For various such incremental constraint definitions, we explain how the associated membership rule sets are incrementally obtained. A natural question concerns the propagation associated with the respective rule sets. We give conditions on the input rule sets that allow us a characterisation of the propagation of the result rule set.

The usability of incremental rule generation for membership rules is demonstrated by an implementation and examples.

*The material in this chapter is based on joint work with Eric Monfroy. It appeared as [Brand and Monfroy, 2003],*

We then consider practical applications that we solve by constraint programming and rule-based constraint propagation.

**Test pattern generation for sequential circuits.** In Chapter 6, we consider a problem from electrical engineering. Since the production process of modern digital circuits is not error-free, it is necessary to verify produced circuits against their specifications. This comparison must be behavioural as the internal circuit structure is inaccessible. Therefore, *test patterns*, sequences
of input data, are used to verify the circuit function by comparing observed and expected output. The test pattern generation problem concerns the generation of such tests for specific circuit faults.

We consider the case of sequential circuits, which have an internal state and for which test generation is thus substantially more complex than for combinational (stateless) circuits. While propositional logic seems natural in this domain, our modelling approach is based on multi-valued logics. The extra values are used for approximating the original problem and for carrying heuristic information.

We develop three different multi-valued logics and compare them by means of standard benchmarks, using our constraint-based implementation. The constraint propagation in our implementation is based on membership rules, and we apply the techniques introduced in the preceding chapters.

*This chapter contains a completely rewritten version of [Brand, 2001b].*

**Modal satisfiability checking.** Chapter 7 presents a constraint-based approach for deciding the satisfiability of modal logic formulas. One approach to solving this problem consists of reformulating it into sequences of propositional satisfiability problems. We extend it by employing a *three-valued* logic in the subproblems instead; the extra value reflects structural information that is lost in the propositional translation. The resulting subproblems are thus non-Boolean, and we view them as CSPs.

We describe a corresponding implementation, which relies on several forms of rule-based constraint propagation. We evaluate the approach and implementation using standard benchmarks. The results show that the three-valued constraint-based approach is competitive with, and in some instances superior to, the purely propositional approach. This shows the interest of a refined modelling made possible by the expressiveness of the CSP-framework.

*This chapter reflects joint work with Rosella Gennari and Maarten de Rijke. It appeared as [Brand et al., 2004], but has been adapted for this dissertation.*

We then return to an application-independent topic.

**Array constraints.** In Chapter 8, we consider constraints that naturally arise when information is arranged in arrays (matrices). We study two forms of constraint propagation for such *array constraints*. Starting from generic template rules that capture the desired local consistency, we derive specialised constraint propagation rules. We then design algorithms that embody the rules and their correctness conditions. The results are systematically developed constraint propagation algorithms for array constraints.
This chapter contains a fully revised and substantially extended version of [Brand, 2001a].

In the next two chapters, we study a domain in which information is naturally structured in array form.

**Qualitative spatial reasoning with relation variables.** In Chapter 9, we discuss an alternative constraint-based approach to qualitative spatial reasoning. In contrast to the standard approach, in which qualitative relations are viewed as constraints, we model relations as variables. These *relation variables* are arranged in an array.

This approach is particularly suitable for qualitative spatial reasoning since space has many aspects (topology, size, direction, etc.). The advantage of this view is that the properties of one spatial aspect as well as the integration of different spatial aspects are expressed as plain constraints on the relation variables. This makes specialised consistency algorithms redundant by a reduction to generic constraint propagation techniques, which we realise by rules.

*This chapter appeared as* [Brand, 2004].

**Qualitative simulation.** Chapter 10 reports our approach to qualitative reasoning involving *change*, based on constraints over relation variables. We use *temporal logic* to describe dynamic system behaviour, which allows concise statements of complex circumstances.

The temporal logic formulas are translated into constraints over the relation variables. We give one translation that simply unfolds the temporal and logical operators, and a second translation that retains the underlying structure by using the array constraints introduced in Chapter 8. This *array translation* leads to particularly compact CSPs and is much more amenable to constraint propagation.

We describe an implementation and discuss case studies.

*This chapter reports joint and on-going work with Krzysztof Apt.*

In Chapter 11, we summarise and contemplate future research issues.