Book Review for AMS Reviews

Review of *Constraints Meet Concurrency*
by Jacopo Mauro
Atlantis Press, 2014

This monograph is a reprint of the author's PhD thesis at the University of Bologna, which was supervised by Maurizio Gabrielli and selected by the Italian Chapter of the *European Association for Theoretical Computer Science* (EATCS) as one of the best Italian PhD thesis in 2012. The thesis explores several theoretical and practical topics related to concurrent constraint programming and solving.

Constraint satisfaction problems (CSPs) have been studied in the field of artificial intelligence and beyond for decades, and a large number of tools and algorithms for constraint programming and solving exist today. Indeed, many practical problems can be casted as CSPs, including scheduling, timetabling, resource allocation, and planning problems in manufacturing, education, medicine, and many other sectors. However, solving large CSPs efficiently is a challenge and requires advanced search algorithms that employ clever search heuristics and fast techniques for consistency checking and constraint propagation.

The recent shift towards parallel computing and powerful cloud infrastructures poses both a challenge and an opportunity for CSP solving. On the one hand, constraint programming features must be (and are) integrated in parallel languages such as Go and Scala as well as mainstream languages such as Java and C#. On the other hand, cloud computing increases the scalability of CSP solvers and opens up the possibility of constraint solving as a service. This is exactly when this timely monograph comes onto the scene. It is well-written and sufficiently self-contained to be easily accessible to theoretical computer scientists with knowledge in programming language semantics.

The monograph consists of two parts that follow two introductory and rather superficial chapters that overview basic aspects of constraints (Ch. 2) and concurrency (Ch. 3). The first part focuses on the usage of constraints in concurrency and investigates the expressive power of dialects of Frühwirth's *Constraint Handling Rules* (CHR) language. This Turing-powerful concurrent constraint language is well-researched and has been implemented over several imperative, functional and logic languages. It is equipped with two formal operational semantics: a classic theoretical semantics and an abstract semantics that abstracts from propagation histories (Ch. 4).

The first two CHR-fragments defined by the author (Ch. 5) still include a host language's built-in unification but impose restrictions that render each fragment less expressive than Turing machines. The first fragment prohibits the use of variables in a constraint rule's guard or body if they do not already appear in the rule's head. The second fragment limits the number of atoms in a rule's head to one. In both cases, the existence of an infinite computation wrt. the abstract operational semantics, and thus termination, is decidable. Regarding the first fragment, the proof employs the theory of *well-structured transition systems* (WSTS). In the second case, a highly non-trivial proof by hand is conducted, as reduction techniques involving WSTS or Petri nets are not applicable.

Two Turing-powerful CHR-dialects that result from Koninck's extension of CHR with priorities, whereby higher priority rules are chosen first for execution, are also considered (Ch. 6). One extension fixes priority at compile-time, hence priorities are static. The other extension allows priority annotations involving variables that are instantiated at run-time and, thus, for priorities to be dynamic. The author proves that CHR with priorities is more expressive than standard CHR under the theoretical semantics, because there exists no acceptable language encoding of CHR with priorities into standard CHR. In addition, an encoding of CHR with dynamic priorities into CHR with static priorities demonstrates that dynamic priorities do not increase expressivity beyond CHR with static priorities. In contrast to standard CHR, the expressivity of CHR with priorities is not altered when restricting rules to those having at most one atom in each rule's head.

The monograph's second part presents first steps towards developing an online service-based portfolio CSP solver. This is motivated by the observation that different solvers are better at solving different problem instances, even within the same problem domain. The author prototypes a cloud-based framework called *Constraint in Clouds* (CiC), for efficiently solving large CSPs by employing several solvers in par-

allel, or the same solver with different parameters (Ch. 7). CiC is implemented in Montesi and Guidi's service-oriented programming language *Jolie* and includes learning techniques for forecasting solving time as well as failure-handling techniques. Systematic benchmarking testifies to the validity of the author's ideas for minimizing average constraint solving time, which are inspired by dispatching rules in scheduling, and also to his choice of classifiers for the smart assignment of cloud resources (Ch. 8).

The CiC prototype reveals two significant shortcomings of Jolie, too: the lack of a broadcasting primitive and a request-response pattern that is inefficient wrt. fault-handling. In response, the author develops a broadcasting concept based on a radix-tree data structure together with an algorithm for computing the minimal number of trees required (Ch. 9). He also presents a new approach to model request-response interactions, which enables a more natural treatment of faults and the programming of timeouts (Ch. 10); this approach is formalized in Guidi et al.'s SOCK calculus for service-oriented computing. The monograph concludes with a brief summary of the author's contributions and a rather out-of-place philosophical treatise on scientific discovery (Ch. 11).

Although the title *"Constraints Meet Concurrency"* may raise higher expectations on synergies between the two fields than the monograph covers, I applaud the author for an outstanding dissertation that advances both the theory and practice of CSP solving. The obtained theoretical results on the expressivity of various dialects of the CHR language are very interesting to experts in the field and nicely complement related work. The work on concurrent-constraint solving as a cloud service and on features of service-oriented programming is comparably less mature but a promising and important first step towards increasing the scalability of CSP solvers by utilizing modern, massively parallel computing infrastructures. Therefore, I sincerely hope that this monograph finds its rather broad audience, which ranges from concurrency theoreticians, to CSP solver engineers, to parallel programming language designers.

GERALD LÜTTGEN
Software Technologies Research Group
University of Bamberg
D–96045 Bamberg, Germany
E-mail: gerald.luettgen@swt-bamberg.de