Analysis and Optimization of CHR Programs

Jon Sneyers*

Dept. of Computer Science, K.U.Leuven, Belgium
jon@cs.kuleuven.be

Introduction. Constraint Handling Rules (CHR) [2] is a high-level, powerful, yet relatively simple *"no box"* CLP language, embedded in a host language, commonly Prolog. It is based on multi-headed committed-choice rules. Recent implementations of CHR consist of a compiler which translates a CHR program to host language code, and a run-time system implementing the constraint store. Originally, CHR was designed for rapid prototyping of user-defined constraint solvers. In the early years of CHR limited attention went to optimized compilation. As a consequence, the reference implementation of CHR [4] comprises a general compilation schema, with only a small number of optimizations. Currently, CHR is increasingly used as a general-purpose programming language in a wide range of applications. Therefore, performance becomes more important, and recently, more advanced compilation optimizations have been proposed [3].

Several implementations of CHR exist [10]. The K.U.Leuven CHR system [11] includes a state-of-the-art CHR compiler for popular Prolog systems like SWI-Prolog and XSB. The formulation of the refined operational semantics of CHR [1], and subsequently the formulation of the call-based refined operational semantics [5], have captured the essentials of current implementations on a formal level. Optimizations can now be defined formally and their correctness w.r.t. the operational semantics can be proved.

The main goals of my research are to improve the practical usability of CHR, to allow a more declarative use of CHR, and to compile CHR programs to more efficient host language code. These goals are achieved by developping, implementing, and evaluating analyses and optimizations for CHR programs.

Current Results. In [8], a new optimization called *Guard Simplification* is introduced, which uses reasoning about the refined operational semantics to eliminate parts of rule guards that are entailed by the conjunction of the negations of guards in earlier subrules. The general compilation schema [4] translates every head constraint occurrence to a Prolog clause. The *Occurrence Subsumption* optimization [7] detects occurrences for which the generated clause is redundant. Both optimizations are implemented in the K.U.Leuven CHR compiler. We also extended CHR to allow optional mode and type declarations, which further improve the optimizations. We developed a stronger optimization called *Guard and Continuation Optimization* [6] which unifies and extends the above two optimizations. It is defined formally using a new call-based refined operational semantics

* This work was partly supported by project G.0144.03 funded by the Research Foundation - Flanders (F.W.O.-Vlaanderen).

M. Gabbrielli and G. Gupta (Eds.): ICLP 2005, LNCS 3668, pp. 450–451, 2005.

for *Occurrence Representations*, a more expressive representation of CHR programs. Correctness results have been proved. Performance improvements of 20 to 40% have been measured [6,7].

Ongoing and Future Work. Future work regarding the guard and continuation optimization includes: enhancing the entailment reasoning knowledge base to increase the strength of optimizations; improving the scalability of our approach (improving compilation times); adding support for declarations of intended patterns of initial queries, allowing more accurate analyses and stronger optimizations. More ideas for future work related to the guard and continuation optimizations are given in [6].

In [9], we show that every algorithm can be implemented in CHR with the best known time and space complexity. However, it remains a challenge to implement classical algorithms in a natural and elegant way. My current work focusses on implementing and comparing shortest path algorithms in CHR. Identifying and eliminating performance bottlenecks in these programs is a great inspiration for new optimizations.

References

1. Gregory J. Duck, Peter J. Stuckey, María García de la Banda, and Christian Holzbaur. The Refined Operational Semantics of Constraint Handling Rules. In *Proceedings of the 20th Intl. Conference on Logic Programming (ICLP'04)*, 2004.
2. Thom Frühwirth. Theory and Practice of Constraint Handling Rules. In *Special Issue on CLP, Journal of Logic Programming*, volume 37 (1–3), October 1998.
3. Christian Holzbaur, María García de la Banda, Peter J. Stuckey, and Gregory J. Duck. Optimizing compilation of Constraint Handling Rules in HAL. *Special Issue of Theory and Practice of Logic Programming on CHR*, 5, 2005. To appear.
4. Christian Holzbaur and Thom Frühwirth. A Prolog Constraint Handling Rules Compiler and Runtime System. *Special Issue Journal of Applied Artificial Intelligence on Constraint Handling Rules*, 14(4), April 2000.
5. Tom Schrijvers, Peter Stuckey, and Gregory Duck. Abstract Interpretation for Constraint Handling Rules. In *Proceedings of the 7th Intl. Conference on Principles and Practice of Declarative Programming (PPDP'05)*, Lisbon, Portugal, July 2005.
6. Jon Sneyers, Tom Schrijvers, and Bart Demoen. Guard and Continuation Optimization for Occurrence Representations of CHR. In *21st International Conference on Logic Programming (ICLP'05)*, Sitges, Spain, October 2005.
7. Jon Sneyers, Tom Schrijvers, and Bart Demoen. Guard Reasoning for CHR Optimization. Technical Report CW 411, K.U.Leuven, Dept. CS, May 2005.
8. Jon Sneyers, Tom Schrijvers, and Bart Demoen. Guard simplification in CHR programs. In *19th Workshop on (Constraint) Logic Programming*, Germany, 2005.
9. Jon Sneyers, Tom Schrijvers, and Bart Demoen. The Computational Power and Complexity of Constraint Handling Rules. In *2nd Workshop on Constraint Handling Rules (CHR'05)*, Sitges, Spain, October 2005. Submitted.
10. CHR homepage. http://www.cs.kuleuven.be/~dtai/projects/CHR.
11. K.U.Leuven CHR system. http://www.cs.kuleuven.ac.be/~toms/Research/CHR.