Journal of Applied Logic 7 (2009) 341–362

Contents lists available at ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

Executable specifications for hypothesis-based reasoning with Prolog and Constraint Handling Rules

Henning Christiansen

Research group PLIS: Programming, Logic and Intelligent Systems, Department of Communication, Business and Information Technologies, Roskilde University, P.O. Box 260, DK-4000 Roskilde, Denmark

ARTICLE INFO

Article history:
Received 18 December 2007
Received in revised form 18 December 2007
Accepted 19 October 2008
Available online 6 November 2008

Keywords:
Abduction
Abduction as deduction
Hypothesis-based reasoning
Logic programming

ABSTRACT

Constraint Handling Rules (CHR) is an extension to Prolog which opens up a spectrum of hypothesis-based reasoning in logic programs without additional interpretation overhead. Abduction with integrity constraints is one example of hypothesis-based reasoning which can be implemented directly in Prolog and CHR with a straightforward use of available and efficiently implemented facilities.

The present paper clarifies the semantic foundations for this way of doing abduction in CHR and Prolog as well as other examples of hypothesis-based reasoning that is possible, including assumptive logic programming, hypotheses with priority and scope, and nonmonotonic reasoning.

Examples are presented as executable code so the paper may also serve the additional purpose of a practical guide for developing such programs, and it is demonstrated that the approach provides a seamless integration with existing constraint solvers.

1. Introduction

Abduction in logic programming has proved to be a powerful technique for solving a range of complex problems, and a large number of approaches have been reported, see [26,27,40,41] for overview. Most known systems are based on meta-level architectures which means that abductive logic programs are interpreted by other logic programs rather than being compiled in an efficient way.

We see abduction as one example within a class of more general hypothesis-based reasoning paradigms. The advantages of abduction are its straightforward declarative specification and the fact that a large class of relevant problems fits naturally into the framework: given an observation O, the task is to find an abductive explanation A which is a set of hypotheses which, when added to the current knowledge base, can explain O and does not introduce inconsistencies. The observation O may be observed symptoms of a malfunctioning system, and A a diagnosis for what may have caused O; or O can represent a desired goal to be achieved and A a set of requirements that must be present in order to reach that goal, e.g., a plan of primitive actions to be performed. While abduction concerns the creation of single sets of hypotheses that can explain the top query, there are other and more dynamic ways to apply hypotheses during a computation, as will be shown below.

In the present paper, we consider general hypothesis-based reasoning realized using available and efficiently implemented logic programming technology in a direct way, which eliminates any interpretative overhead. More specifically, we suggest PROLOG as an overall driver engine (and, as knowledge specification language) with Constraint Handling Rules, CHR, complementing the ability to manage hypotheses by declarative specifications and efficient implementation.

E-mail address: henning@ruc.dk.

1570-8683/$ – see front matter
doi:10.1016/j.jal.2008.10.004

Example 1.1. The following program, which specifies and solves an abductive problem, is written in the syntax for CHR in SICStus and SWI PROLOG. Constraint predicates are distinguished from normal PROLOG predicates by declarations. Constraints of CHR play the role of abducibles and integrity constraints are written as CHR rules, e.g., as indicated below with '==>'.

```
:- chr_constraint professor/1, rich/1, has_good_students/1.
professor(X),rich(X) ==> fail.
happy(X):- rich(X).
happy(X):- professor(X), has_good_students(X).
```

Execution of the query `?- professor(peter), happy(peter)` results in a final constraint store (= abducible explanation) consisting of {`professor(peter)`, `has_good_students(peter)`}. The CHR rule excludes an explanation which includes `rich(peter)`, and thereby forcing the PROLOG engine to try the second clause in its eagerness to verify the query.

The approach provides a seamless and efficient integration with all facilities of the underlying PROLOG and CHR system, including existing constraint solvers, in a way that makes it possible to go beyond strictly abductive reasoning. Furthermore, subtle problems with variables in abducible hypotheses in many earlier approaches do not arise here. The main weakness of using CHR and PROLOG for abduction in this way, when comparing with some other systems, is the limited use of negation.

By hypothesis-based reasoning, we refer to a space of problem solving and programming techniques in which logic programs are extended with a global state of hypotheses which interacts with the logic program and can be manipulated implicitly as in abductive reasoning or in more explicit ways, and which may or may not relate in all details to declarative specifications. It will be shown how abduction and other instances of hypothesis-based reasoning can be realized in straightforward ways with CHR and PROLOG, and we intend in this way also to indicate that there is a rich scope for developers to produce their own variants for different purposes.

Constraint Handling Rules were introduced in the early 1990s by Thom Frühwirth (primary reference [32] from 1998 provides background and early history), and is now available as extensions to major PROLOG systems, including SICStus and SWI. The idea of using CHR for abduction was originally suggested by Slim Abdennadher and the present author in 2000 [1]. The combination of PROLOG and CHR for abductive and other kinds of hypothesis-based reasoning has been developed together with Verónica Dahl since 2002 as a central collaborator, e.g., [13–15].

The present paper aims at giving a coherent presentation of the approach, clarifying the semantic foundations (that were left implicit in earlier publications) as well as exposing hypothesis-based reasoning with PROLOG and CHR as a powerful and flexible programming paradigm.

Examples have been checked using SICStus PROLOG [58]; we use in most cases the CHR syntax and facilities provided by SICStus PROLOG version 4 which is intended to be identical to that of SWI PROLOG [59]. Some extensions to CHR that we describe below are implemented using specifics of SICStus PROLOG version 3, which differs in some details and has a larger collection of low-level facilities, so transfer to version 4 may not be trivial in all cases.

2. Syntax and semantics of CHR and its extensions

Where PROLOG represents a top-down, backward chaining computational paradigm, Constraint Handling Rules, CHR, extends with bottom-up, forward chaining computations. Operationally, CHR is defined as rewriting rules over a constraint store, which can be seen as a global resource to be used by a PROLOG program for storing, manipulating and consulting different hypotheses. This resource-oriented understanding of CHR has led to the formulation of a semantics for CHR [4] based on linear logic (that we have not applied here). We introduce firstly CHR, then extend it with disjunctions into CHR$^{\vee}$ [2], and finally we combine it with PROLOG into a language which we refer to in this paper as PROLOG+CHR.

2.1. Constraint handling rules, CHR

CHR inherits the basic nomenclature of PROLOG, and we refer to the notions of constant and function symbols, predicates, variables, atoms,[1] terms and queries and use initial capital letter for variables, etc. Substitutions, grounding and renaming substitutions are defined as usual.

The predicates in a CHR program are called *constraint predicates*, belonging to disjoint sets of *program specific* ones and a fixed set of *built-in* predicates each having a fixed meaning, including $=$ with its standard meaning of syntactic equality, and `true` and `false`; we assume a theory $\mathcal{B}$ for the built-ins, e.g., $\mathcal{B} \models \mathtt{a} = \mathtt{a}$ and $\mathcal{B} \not\models \mathtt{false}$. A *constraint* is an atom with a constraint predicate, which may be further classified as *program specific* or *built-in* according to its predicate; in contexts with no ambiguity, 'constraint' may also be used for 'constraint predicate'. A CHR program consists of rules of the following kinds.

[1] There is a slight confusion here. Some PROLOG sources use 'atom' to refer to constants; we use 'atom' in the same way as the literature on mathematical logic, i.e., $predicate(term_1, \ldots, term_n)$, $n \geqslant 0$.