Experiences and directions for Abduction and Induction using Constraint Handling Rules*

POSITION PAPER

Henning Christiansen
Roskilde University, Computer Science Dept.,
P.O. Box 260, DK-4000 Roskilde, Denmark henning@ruc.dk

Abstract

Techniques for doing abduction in a combination of Prolog and Constraint Handling Rules (CHR) are reviewed, and the possible extension to combine with induction is considered. While the indicated implementation for abduction is very efficient, the ideas for induction are at a much more experimental stage. However, experimentation within CHR indicates a logical semantics for the induction mechanisms under consideration and their offset in abductive logic programming.

1 Introduction

Our interest in natural language processing using Constraint Handling Rules (CHR) has lead to interesting observations and results concerning the realization of abduction and induction within CHR.

In this paper, we give an overview of our present results, and indicate new directions.

Our present work has established a very efficient and straightforward implementation of abduction in Prolog by means of CHR, and our thoughts on induction are at the time of writing still at a speculative level.

2 Background

The language of Constraint Handling Rules, CHR, is an extension to Prolog intended as a declarative language for writing constraint solvers for CLP systems; here we give a very compact introduction and refer to [Frühwirth, 1998] for details. CHR is now integrated in several major Prolog implementations and has gained popularity for a variety of applications due to its expressibility and flexibility, which goes far beyond the traditional applications of constraint programming (such as finite domains, arithmetic, etc.).

Constraints of CHR are first-order atoms whose predicates are designated constraint predicates, and a constraint store is a set of such constraints, possible including variables that are understood existentially quantified at the outermost level. A constraint solver is defined in terms of rules which can be of the following two kinds.

Simplification rules: $c_1, \ldots c_n$ <=> *Guard* | $c_{n+1}, \ldots, c_m$

Propagation rules: $c_1, \ldots c_n$ ==> *Guard* | $c_{n+1}, \ldots, c_m$

The c's are atoms that represent constraints, possible with variables, and a simplification rule works by replacing in the constraint store, a possible set of constraints that matches the pattern given by the *head* $c_1, \ldots c_n$ by those corresponding constraints given by the *body* $c_{n+1}, \ldots, c_m$, however only if the condition given by *Guard* holds. A propagation rule executes in a similar way but without removing the head constraints from the store. In addition, rule bodies and guards may include equalities and other standard relations having their usual meaning. The declarative semantics is hinted by the applied arrow symbols (bi-implication, resp., implication formulas, with variables assumed to be universally quantified) and it can be shown that the indicated procedural semantics agrees with this. This is CHR explained in a nutshell.

Abduction in CHR was first proposed by [Abdennadher and Christiansen, 2000]. They proposed a translation of abductive logic programs into CHR, so that the logic program component as well as integrity constraints are written as CHR rules with all predicates consider constraints. We explain a later variation of the principle below in section 3.

Interestingly, the technique can be explained as a transformation of abduction into deduction [Christiansen, 2002; 2005] and it is interesting to compare this with an early paper [Console *et al.*, 1991] (written at a time when CHR did not exist) that pointed out an isorphism between a class of abductive problems and deduction.

3 Abduction in Prolog with a few lines of CHR

The first proposals [Abdennadher and Christiansen, 2000; Christiansen, 2002; 2005] formulated everything in terms pure CHR, and [Christiansen and Dahl, 2004; 2005a] have applied the principles in a combination with Prolog that is explained briefly here.

The basic mechanism is simple: abducible predicates are declared as constraints and when called from a Prolog program, CHR's constraint store serves as a container that collects the abduced atoms. (This is opposed to plain Prolog in which a call to an empty predicate yields failure.)

*This work supported by the CONTROL project, funded by Danish Natural Science Research Council.

For example, the complete hand-coded implementation of an abducible predicate `a/1` is provided by the following three lines.

```
:- use_module(library(chr)).
handler abduction.
constraints a/1.
```

A principle applied in many approaches to abduction is to aim at minimal abductive answers (measures in number of literals) by always trying to unify new abductive atoms with existing ones. As we have argued elsewhere, this principle is not always desirable, but anyhow, it can be implemented by a single CHR rule; the following provides a correct implementation.

```
a(X), a(Y) ==> true | (X=Y ; dif(X,Y)).
```

(We have used SICStus Prolog; facilities explained in [Swedish Institute of Computer Science, 2004].)

Another important aspect of abductive logic programming [Kakas *et al.*, 1998] is the presence of integrity constraints which are logical conditions that limit the possible abductive explanations, in order to exclude nonsense scenarios. Using CHR for abduction, its rules are handy tools for writing integrity constraints. The following rule whose intuitive meaning should be self-explanatory, can be written directly in CHR and will provide the right meaning, declaratively as well as operationally.

```
married(X,Y), married(X,Z)
    ==> Y \= Z | bigamist(X).
```

Compared with other approaches to abduction, a main advantage is that the rules and given facts of an abductive logic program are represented and executed as Prolog code with no meta-interpreter involved to slow things down. Not surprisingly, for the right sort of abductive problems (where deductive steps dominate) we gain significant speed-up with the indicated methods.

The price, then, is a limited use of negation. Where approaches such as [Kakas *et al.*, 2000] can get interesting results by exporting knowledge from negated subgoals whose solution depends on abducibles, this is incompatible with our approach. However, a simple form of so-called explicit negation is supported by our method; see [Christiansen and Dahl, 2004].

4 An application to natural language semantics

As an aside to the overall theme of this position paper, we mention an approach to discourse semantics and analysis called Meaning-in-Context [Christiansen and Dahl, 2005b] which is inspired by the abductive mechanism explained above.

As central for discourse understanding, we consider context (here defined as the set of knowledge learned so-far from the discourse) as a central component. The analysis of any little fragment is analyzed using knowledge from the context, and may contribute with new knowledge to that context, as well as its meaning is expressed in a compact form by means of references to the context. This model is formalized using a possible worlds semantics, which we can map directly into abductive programs formulated as above.

5 Rules in the constraint stores

In yet unpublished work together with V. Dahl, we have extended CHR with rules in the constraint store. The idea came up due to a frustration over the fact that the model of abduction described so far could not express the meaning of general statements of such as "All fishes swim". What we want is, of course, a model so that all known fishes, as well as any future fishes met later in the discourse are assigned the property of being able to swim.

In CHR, there is no counterpart to Prolog's (anyhow dubious) `assert` so that we can extend the program with a new rule. What we propose instead is to extend the notion of constraints from being atomic statements to include also rules such as the following.

```
fish(X) ==> swim(X).
```

An inefficient prototype implementation can be produced by including as set of meta-rules in the original program (for dynamically created rules with head size up to some number). Let us for simplicity assume that prefix operator `?` is applied as a general constraint to capture all abducible predicates, i.e., we write `?fish(X)` instead of `fish(X)`. The following sample metarule is capable of handling a rule in the constraint store with two constraints in the head.

```
?A, ?B, (?A, ?B ==> Body) ==> call(Body).
```

(In practice, we should apply a slightly more complicated version in order to avoid the inherent problems with a non-ground representation. Never mind, the rule above works for interesting examples.)

When a new "dynamic" rule is created it will apply to all existing pairs of constraints that match its head, as well any future such that may arise.

The generation of a rule such as `fish(X) ==> swim(X)` can be done in the body of either another CHR rule or Prolog rule. We just need to declare `==>` as a constraint predicate (the compiler accepts it `;-`), and just "call" a new rule as if it were a goal and it is added to the constraint store.

Interestingly, CHR's declarative semantics generalize immediately with correctness statements for the indicated implementation analogous to those we recognize for CHR.

6 A proposal for induction in CHR

Induction in a logic programming setting means to generate new rules when sufficient evidence for doing so it is present (and it is seen as practical to in(tro)duce that rule). We saw above how it is possible to generate and install new CHR rules when program is running, so (naively stated) to do induction in this model is a matter of writing the program lines that keep track of such evidence.

Referring to the example above, let us assume a program that concerns `fish`es and `swim`ming entities. We can say that we would like to induce a rule p`(X) ==>` q`(X)` whenever p has been observed a sufficient amount of times as facts

$p(a)$ and for all of then, we have also $q(a)$ (and perhaps $q(b)$ for some b without $p(b)$).

If we disregard efficiency, it is straightforward to extend a CHR program as to maintain the necessary statistics each time a constraint is called and check whether the indicated evidence is strong enough for launching a new rule.

The approach is implementable, and will provide a more dynamic paradigm that Inductive Logic Programs, as rules are created dynamically and participate in the continued computational process.

If this proposal should be developed into something of any practical value, it is worth seeking inspiration in methods for data mining, more precisely association rule mining, to which our approach shows obvious similarities.

7 What else do we need?

Abduction in CHR as we have described it can be considered a well-understood, ready to use technique, whereas the approaches to dynamic rule generation as well as evidence-based induction still are at a very preliminary stage.

However, using it for experimentation with possible new mechanisms for induction in logic programming and application thereof (e.g., for advanced discourse analysis) is possible and feasible.

Important aspects of induction such as syntactic bias and preference that are known from other paradigms of induction have been ignored until now and need, of course, to be related to this framework.

Clearly the indicated techniques with a direct implementation in CHR seems very inefficient, and the main contribution of our proposal may be that of modeling a new approaches to induction, which, then may inspire to the development of practically relevant implementation. One obvious advantage of working within CHR at this experimental stage is there is always a logical semantics semantics underneath.

Current research related to abduction in CHR concerns the use of a probabilistic semantics as a systematic way to achieve weights and priorities during evaluation, as well as an alternative way of execution CHR so that backtracking is replaced by a simultaneous evaluation of different possible abductive explanations.

References

[Abdennadher and Christiansen, 2000] Slim Abdennadher and Henning Christiansen. An experimental CLP platform for integrity constraints and abduction. In *Proceedings of FQAS2000, Flexible Query Answering Systems: Advances in Soft Computing series*, pages 141–152. Physica-Verlag (Springer), 2000.

[Christiansen and Dahl, 2004] Henning Christiansen and Veronica Dahl. Assumptions and abduction in Prolog. In Elvira Albert, Michael Hanus, Petra Hofstedt, and Peter Van Roy, editors, *3rd International Workshop on Multiparadigm Constraint Programming Languages, MultiCPL'04; At the 20th International Conference on Logic Programming, ICLP'04 Saint-Malo, France, 6-10 September, 2004*, pages 87–101, 2004.

[Christiansen and Dahl, 2005a] H. Christiansen and V. Dahl. HYPROLOG: a new approach to logic programming with assumptions and abduction. In Maurizio Gabbrielli and Gopal Gupta, editors, *Proceedings of Twenty First International Conference on Logic Programming (ICLP 2005)*, Lecture Notes in Computer Science, 2005. To appear.

[Christiansen and Dahl, 2005b] H. Christiansen and V. Dahl. Meaning in Context. In Anind Dey, Boicho Kokinov, David Leake, and Roy Turner, editors, *Proceedings of Fifth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT-05)*, volume 3554 of *Lecture Notes in Artificial Intelligence*, pages 97–111, 2005.

[Christiansen, 2002] Henning Christiansen. Abductive language interpretation as bottom-up deduction. In Shuly Wintner, editor, *Natural Language Understanding and Logic Programming*, volume 92 of *Datalogiske Skrifter*, pages 33–47, Roskilde, Denmark, July 28 2002.

[Christiansen, 2005] Henning Christiansen. CHR Grammars. *Int'l Journal on Theory and Practice of Logic Programming*, 2005. To appear.

[Console *et al.*, 1991] Luca Console, Daniele Theseider Dupré, and Pietro Torasso. On the relationship between abduction and deduction. *J. Log. Comput.*, 1(5):661–690, 1991.

[Frühwirth, 1998] Thom Frühwirth. Theory and practice of constraint handling rules, special issue on constraint logic programming. *Journal of Logic Programming*, 37(1–3):95–138, October 1998.

[Kakas *et al.*, 1998] A.C. Kakas, R.A. Kowalski, and F. Toni. The role of abduction in logic programming. *Handbook of Logic in Artificial Intelligence and Logic Programming*, vol. 5, Gabbay, D.M, Hogger, C.J., Robinson, J.A., (eds.), Oxford University Press, pages 235–324, 1998.

[Kakas *et al.*, 2000] A.C. Kakas, A. Michael, and C. Mourlas. ACLP: Abductive Constraint Logic Programming. *Journal of Logic Programming*, 44:129–177, 2000.

[Swedish Institute of Computer Science, 2004] Swedish Institute of Computer Science. SICStus Prolog user's manual, Version 3.12. Most recent version available at `http://www.sics.se/isl`, 2004.