29 july 2005

Experiences and directions for Abduction and Induction using Constraint Handling Rules

Henning Christiansen

Computer Science, bldg 42.1
Roskilde University
Universitetsvej 1
P.O. Box 260
DK-4000 Roskilde
Denmark
www.ruc.dk/~henning

Computer Science
Roskilde University

Presentation at Workshop on Abduction and Induction, AIAI'05, Edinburgh, Scotland

Motivation and overview

- Results on abduction by means of constraint logic programming (CLP)
  - Indicates inherent relationship between the two
  - Efficient and elegant implementation
- Speculations and experiments with induction
  - Current results: high flexibility
    (efficiency and scaleability problematic)
  - Discuss:
    - Also here "inherent relation"?
    - Inspiration for new CLP-like technology for abduction-induction integration?

Playing with abduction in Prolog & CHR

A Prolog program:

```
p(X):- q(X), a(X).
q(1).
```

Playing with abduction in Prolog & CHR

A Prolog program:

```
p(X):- q(X), a(X).
q(1).
```

A query:

```
?- p(X).
no
```

Playing with abduction in Prolog & CHR

and CHR

A Prolog program:

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

```
p(X):- q(X), a(X).
q(1).
```

A query:

```
?- p(X).
no
```

Playing with abduction in Prolog & CHR

and CHR

A Prolog program:

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

```
p(X):- q(X), a(X).
q(1).
```

A query:

```
?- p(X).
```

~~no~~

Playing with abduction in Prolog & CHR

and CHR

A Prolog program:

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

```
p(X):- q(X), a(X).
q(1).
```

A query:

```
?- p(X).
```

~~no~~

```
X = 1
a(1) ?
```

Playing with abduction in Prolog & CHR

and CHR

A Prolog program:

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

```
p(X):- q(X), a(X).
q(1).
```

```
a(1) ==> a(2).
a(2), a(3) ==> fail.
```

A query:

```
?- p(X).
```

~~no~~

Playing with abduction in Prolog & CHR

and CHR

A Prolog program:

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

```
p(X):- q(X), a(X).
q(1).
```

```
a(1) ==> a(2).
a(2), a(3) ==> fail.
```

A query:

```
?- p(X).
```

~~no~~

```
X = 1
a(1), a(2) ?
```

Playing with abduction in Prolog & CHR

and CHR

A Prolog program:

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

```
p(X):- q(X), a(X).
q(1).
```

```
a(1) ==> a(2).
a(2), a(3) ==> fail.
```

A query:

```
?- p(X).
```

~~no~~

```
X = 1
a(1), a(2) ?
```

```
?- a(7), p(X).
X = 1
a(7), a(1), a(2) ?

?- p(X), a(3).
no
```

Constraint Handling Rules

- Declarative extension to Prolog for writing constraint solvers [Frühwirth, 1993, 1995]
- A white-box approach to CLP
- Available in SICStus Prolog from 1998; now several impl., also in Haskell and Java
- Has gained popularity as general prog. lang.
  - E.g. language processing (CHR Grammars [2002, 2005])
  - Abductive reasoning
  - .... and a lot of other things, bioinfo., ray-tracing, ... search for CHR web pages

CHR, Introduction by example

```
:- use_module(library(chr)).
handler leq.
constraints leq/2.
:- op(500, xfx, leq).
X leq Y , Y leq Z ==> X leq Z.
X leq Y , Y leq X <=> X=Y.
X leq Y <=> X=Y | true.
X leq Y \ X leq Y <=> true.

p(X,Y):- q(X), r(Y,Z), X leq Z.
```

- **Execution model**: Constraint store, replace/add constraints
- **Declarative semantics:** as indicated by arrow symbols
- **Implementation:** Attributed var's; lot of ongoing work on optimization such as indexing, etc.

Abduction with CHR

- [Abdennadher, Ch., 2000] observed analogy

abducibles	~	constraints of CHR
integrity constraints	~	rules of CHR

- Applied in CHR Grammar system [Ch., 2002, 2005]
- Together with Prolog (and DCG) [Ch., Dahl, 2004-5]
  - HYPROLOG system [ICLP, 2005] available soon

    (abduction, assumptions, and auxiliaries, ...)

A few more details ...

Abduction in CHR, contd. (available in HYPROLOG)

If you say "`abducibles a/1.`" you get *explicit negation*

```
a_(X), a(X) ==> fail.
```

If, furthermore, you say "`compaction a/1.`" you get

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

Advantages:

- Easy to use, full flex. of CHR for the ICs,
- Much more efficient that other approaches to abductive logic programming (up to 2000x for *selected* example)
- Integrates with all of Prolog's and CHR's built-in stuff
  (logical as well as dirty ;-)

Disadvantage:

- Negation essentially limited compared with other, metainterpreter-based approaches

Successful application:

- Elegant model for discourse representation and abduction-based discourse analysis for Natural Language
  - "Meaning-in-Context" [Ch., Dahl, CONTEXT'05]

What you have seen until now is documented, implemented, tested, published etc.

What remains ...

exists as fragments, sketches, chuncks of inefficient code, speculations, and discussions

Towards an integration of abd/induction in Prolog+CHR

Part 1: Rules as dynamic entities, i.e., rules-as-constraints

Example of desirable behaviour:

```
?- a, (a, b ==> c), b.
a, b, c, (a, b ==> c) ?
```

Obs: Declarative semantics generalizes immediately

Prototype implementation

Version 0: Propositional case only

Generic abducible pred. "**?**"

i.e., write **a** as **?a** and **a,b ==> c** as **?a,?b ==> ?c**

One metarule for each no. of head atoms:

```
constraints ?/1, (==>)/2.
?A, ?B, (?A, ?B ==> WhatEver) ==> WhatEver.
```

Correct implementation with variables

Ground representation of dynamic rules

```
?a(*x), ?b(*x,*y) ==> write(*x), ?c(*x,*y).
```

handled by meta-rule of form

```
?A, ?B, (?A1, ?B1 ==> Body) ==>
  true & instance((A1,B1,Body),(A,B,LiveBody)) % guard
|
  LiveBody.
```

```
instance(...):- 10 lines of Prolog .
```

Careful impl. provides very dynamic system:

```
?-   (H ==> B),

     H = (?a(*x), ?b(*y)),

     ?a(1), ?b(2),

     B = (?c(*x,*y), More),

     More = ?d(*y).
```

Careful impl. provides very dynamic system:

Abstract and "useless" rule delayed

```
?-   (H ==> B),

     H = (?a(*x), ?b(*y)),

     ?a(1), ?b(2),

     B = (?c(*x,*y), More),

     More = ?d(*y).
```

Careful impl. provides very dynamic system:

Abstract and "useless" rule delayed

Rule compiles partly; can apply but produces delayed body

```
?-   (H ==> B),
     H = (?a(*x), ?b(*y)),
     ?a(1), ?b(2),
     B = (?c(*x,*y), More),
     More = ?d(*y).
```

Careful impl. provides very dynamic system:

```
?-   (H ==> B),

     H = (?a(*x), ?b(*y)

     ?a(1), ?b(2),

     B = (?c(*x,*y), More),

     More = ?d(*y).
```

Abstract and "useless" rule delayed

Rule compiles partly; can apply but produces delayed body

Halfway compiled rule applies; knows argum's for delayed body

Careful impl. provides very dynamic system:

```
?-   (H ==> B),

     H = (?a(*x), ?b(*y)),

     ?a(1), ?b(2),

     B = (?c(*x,*y), More),

     More = ?d(*y).
```

Abstract and "useless" rule delayed

Rule compiles partly; can apply but produces delayed body

Halfway compiled rule applies; knows argum's for delayed body

Body compiled partly; rest is delayed and ?c(1,2) is called.

Careful impl. provides very dynamic system:

```
?-  (H ==> B),

    H = (?a(*x), ?b(*y)),

    ?a(1), ?b(2),

    B = (?c(*x,*y), More),

    More = ?d(*y).
```

Abstract and "useless" rule delayed

Rule compiles partly; can apply but produces delayed body

Halfway compiled rule applies; knows argum's for delayed body

Body compiled partly; rest is delayed and ?c(1,2) is called.

Compilation of rule finishes; ?d(2) is called.

Abd/induction integration, part 2

We have:

- abduction
- dynamically created rules
- ... in a powerful programming environment

so we just need to program how and when rules are created.

A sketch of an example ...

Pseudocode for naive induction strategy

```
?Pred(Arg) ==>
   if (Pred(Arg) is new) then
      if (++Count(Pred) > n) then
        for any Pred' with (Pred(X) in store
                                    implies Pred'(X) in store)
        do
         (?Pred(*x) ==> ?Pred'(*x))
         unless created already.
```

Example, *n* = 2

Example, *n* = 2

```
?- ?swim(sharky), ?swim(coddy), ?swim(flipper),
```

Example, *n* = 2

```
?- ?swim(sharky), ?swim(coddy), ?swim(flipper),
   ?fish(sharky), ?fish(coddy),
```

Example, *n* = 2

```
?- ?swim(sharky), ?swim(coddy), ?swim(flipper),
   ?fish(sharky), ?fish(coddy),

   ?fish(soly).
```

Example, *n* = 2

```
?- ?swim(sharky), ?swim(coddy), ?swim(flipper),
   ?fish(sharky), ?fish(coddy),
                      ?fish(*x) ==> ?swim(*x).
   ?fish(soly).
```

Example, *n* = 2

```
?- ?swim(sharky), ?swim(coddy), ?swim(flipper),
   ?fish(sharky), ?fish(coddy),

   ?fish(soly).
```

```
?fish(*x) ==> ?swim(*x).
```

Rule applies so **?swim(soly)** part of answer

Example, *n* = 2

```
?- ?swim(sharky), ?swim(coddy), ?swim(flipper),
   ?fish(sharky), ?fish(coddy),

   ?fish(soly).
```

```
?fish(*x) ==> ?swim(*x).
```

Rule applies so `?swim(soly)` part of answer

- This example is implementable, no cheating
- CHR has a nice device `:- option(already_in_store)`
- Thus `?p(*x) ==> ?q(*x)` plus `?q(*x) ==> ?p(*x)` is not a problem

Summing up

- Abduction (with no real negation) works in Prolog+CHR
  - elegant, flexible, efficient
- Simple induction can be added to form integration
  - flexible, inefficient, bad scaleability
- Possible extensions
  - explicit negation and exceptions (??)
  - NB: everything *can* be programmed
  - Efficiency and scaleability may be obtained by
    send-new-rules-to-file-and-recompile (??)
- I dare not say anything about weight and statistics

However ...

Ongoing work on abduction using CHR

- **Probabilistic semantics as way to weighted abd.**
  - Inspiration from [Frühwirth, Di Pierro, Wickely, 2002]: Probabilistic Constraint Handling Rules?
  - Add mechanisms to follow most promising alternative (good heuristics for NLP)
  - Learn probabilities by PRISM system (Sato & al.) ?
- **Alternative CHR execution strategy (for NLP)**
  - Splitting state whenever alternatives occur
  - Efficient copying (in C with relative addr. scheme)
  - All states in parallel
  - Assumptions: ICs should eliminate nonsense state; sets of abducibles of "manageable size"

Conclusion & discussion

What did we learn from this exercise?

Open questions

Conclusion & discussion

What did we learn from this exercise?

- Fun to play with CHR and advanced reasoning

Open questions

Conclusion & discussion

What did we learn from this exercise?

- Fun to play with CHR and advanced reasoning
- Clarified rel'ship abduction <-> constraint LP

Open questions

Conclusion & discussion

What did we learn from this exercise?

- Fun to play with CHR and advanced reasoning
- Clarified rel'ship abduction <-> constraint LP
- Induction as well as integration with abduction can be modelled with some-sort-of-logical-semantics

Open questions

Conclusion & discussion

What did we learn from this exercise?

- Fun to play with CHR and advanced reasoning
- Clarified rel'ship abduction <-> constraint LP
- Induction as well as integration with abduction can be modelled with some-sort-of-logical-semantics

Open questions

- Clarify rel'ship induction <-> constraint LP??

Conclusion & discussion

What did we learn from this exercise?

- Fun to play with CHR and advanced reasoning
- Clarified rel'ship abduction <-> constraint LP
- Induction as well as integration with abduction can be modelled with some-sort-of-logical-semantics

Open questions

- Clarify rel'ship induction <-> constraint LP??
- Useful and efficiently implemented models?