CLP(gRel): Explicit Manipulation of (ground) Relational Dependencies in Logic Programming

Bernhard Pfahringer
(email: bernhard@ai.univie.ac.at)

Austrian Research Institute for Artificial Intelligence
Schottengasse 3, A-1010 Vienna, Austria

Area: Automated Reasoning/Constraint Logic Programming
Category: Short Paper

Abstract

In this paper we introduce CLP(gRel), a kind of CLP language allowing for explicit manipulation of relational dependencies between variables. It is a straight-forward generalization of *domain variables*, a successful constraint propagation technique introduced by [5], Domain variables are the special case of arity one. Unification is extended to handle variables ranging over *relations* correctly (and efficiently). Constraints expressed by variables ranging over *relations* can be used to actively prune the search space by detecting combinations of values that are bound to fail early (cf *forward checking* [5]). The benefit of *relations* is demonstrated in the context of a non-toy problem: diagnostic usage of the KARDIO system, a qualitative simulation model of the electrical activity of the heart, is sped up considerably.

1 Introduction

In this paper we show how *domain variables*, a successful constraint propagation technique introduced by [5], can be generalized to capture relational dependencies of logical variables. Unification will be extended to handle variables ranging over *relations* correctly (and efficiently). Such constraints can be used to actively prune the search space by detecting combinations of values that are bound to fail early (cf *forward checking* [5]). The benefit of *relations* will be demonstrated in the context of a non-toy problem. We are using the KARDIO system [1], a qualitative simulation model of the electrical activity of the heart, to exemplify our points. Given a state of the heart (some combination of arrhythmias) the KARDIO model can be used to compute possible ECG patterns and vice versa. The design of the model influences efficiency heavily : simulation (going from arrhythmias to ECG patterns) is fast whereas diagnosis (going from ECG patterns to arrhythmias) is slow. By use of CLP(gRel) the latter can be sped up without changing the structure of the model considerably.

This paper is outlined as follows: Section 2 defines domain variables and CLP(gRel), an instance of the general CLP scheme [9] able to handle (g)round (rel)ational dependencies of variables. Section 3 gives a procedural semantics by means of meta-interpreter. Section 4 introduces the KARDIO model and reports on experimental results. Section 5 discusses the results, compares our solution to other approaches and outlines further research.

2 Variables ranging over relations

Domain variables can be seen as a way of representing disjunction explicitly by means of a data-structure as opposed to encoding disjunction implicitly in two or more clauses for a predicate. So the following predicate

```
?- nonzero(X) :- X <= [under_60, between_60_100, over_100].
```

where we assume `<=/2` to be an operator specifying the set of legal values for a variable, can be explained to be a syntactic variant of:

```
nonzero(under_60).
nonzero(between_60_100).
nonzero(over_100).
```

The difference is that the former does not necessitate the creation of a choice point during search for a prove and allows for explicit reasoning about the disjunction. Now the straightforward generalization discussed in this paper is to allow predicates of arbitrary arity to be rewritten instead of only unary predicates. This will allow for representing a predicate like:

```
gen_ect(quiet, no).
gen_ect(veb, uni).
```

with the following single clause:

```
gen_ect(VEF, Focus) :-
   t(VEF, Focus) in [t(quiet, no), t(veb, uni)].
```

if we assume an operator `in/2` which relates an arbitrary number of variables - represented by a structure of the appropriate arity taking the variables as arguments - to the according set of legal atomic [1] solutions for the variables. - represented as a list of structures of the appropriate arity taking the values as arguments. So in our example the variables `VEF` and `Focus` are constrained to the values of either `quiet` and `no` or `veb` and `uni` respectively. The extended unification algorithm (described below) will ensure that variable `VEF` is only bound to either `quiet` or `veb` and will furthermore instantiate `Focus` accordingly, and vice versa. Speaking in relational database terms, this list could be called the table of rows expressing the relation `gen_ect` with columns labeled `VEF` and `Focus`.

3 Extended Unification: Procedural Semantics

Unification has to be extended to handle variables constrained to a relation. In the following we will call such variables *r-variables*. Unification has to handle the three following cases:

- If a standard variable and a r-variable have to be unified, the standard variable is simply bound to the r-variable.
- If a constant value and a r-variable have to be unified, we have to select those *rows* of the list of legal solutions, that have this constant value as ith argument, if r-variable is the ith variable. Depending on the number N of such rows found, unification either fails - N = 0 - or unification binds the r-variable to the constant value and all other variables of the disjunction to their according values - N = 1 - or unification binds all the r-variables to newly created r-variables be constrained to the new list of legal solutions - N > 1.
- If two r-variables have to be unified, the two lists of legal solutions will be joined. Again, depending on the number N of resultant rows, unification either fails - N = 0 -, or unification will bind all the variables involved in the two disjunctions to constant values - N = 1 -, or unification binds all the involved r-variables to newly created r-variables which are constrained to the join result.

This behavior of unification can be modeled by the following meta-interpreter. This interpreter is only capable of handling unifications of atomic values, standard variables and r-variables. r-variables are represented by terms `attr( Var, N-dis(AllVars,AllRows))`, meaning `Var` is attributed by `N-dis(AllVars,AllRows)`, where `N` specifies that `Var` is the

[1] We assume legal values to be atoms or numbers throughout this paper, but results are valid for ground terms, too. Note that both *domain variables* and CLP(gRel) are constrained to be syntactic variants of ground unit clauses. Nonground terms will be a non-trivial extension, which will be discussed later.

Nth argument of the term `AllVars` representing all the variables involved in this disjunct, and lastly `AllRows` is the list of all legal solutions.

```
unify(X, Y) :- var(X), !, X = Y.
unify(X, Y) :- var(Y), !, X = Y.
unify( attr(V1, N1-dis(Vars1,Rows1)),
       attr(V2, N2-dis(Vars2,Rows2))) :-
   !,
   join(N1, N2, Rows1, Rows2, NewRows),
   joinbind(NewRows, N1, Vars1, N2, Vars2).
unify( attr(V, N-dis(Vars,Rows)), Atom) :-
   !,
   select(Rows, N, Atom, NewRows),
   selectbind(NewRows, N, Vars).
unify(Atom, attr(V, N-dis(Vars,Rows))) :-
   !,
   select(Rows, N, Atom, NewRows),
   selectbind(NewRows, N, Vars).
unify(X, X).
```

Unification being such a basic operation in a logic programming environment, it cannot be delegated to a meta-interpreter when expecting reasonable efficiency. Some experimental Prolog systems already support user-defined extensions to unification either by means of meta-structures [10], [7] or by attaching attributes to variables [8]. None is comparable to an industrial-strength Prolog system. But fortunately we were able to use a modified version of Sicstus Prolog [2] for implementing our experiments. This version of Sicstus Prolog allows for the specification and management of *attributed variables* and unification therefore or for the specification and management of meta-structures by means of an underlying *delay*-mechanism thus extending builtin unification. [2] The user can specify both unification of two *attributed variables* and of an *attributed variable* and an arbitrary term by writing (Prolog-) clauses for two predicates `metametaunify` and `metatermunify`.

4 The KARDIO Model

The KARDIO expert system models the electrical activity of the heart in a qualitative way. We will just briefly sketch the model, an extensive description of KARDIO can be found in [1]. Overly simplified, the heart works electrically as follows: certain generators supply electrical impulses which are in turn conducted and combined through specific pathways. These resultant impulses allow the model to predict possible ECG patterns. The current version of KARDIO relates 943 different combinations of basic arrhythmias to 3096 different ECG patterns yielding a total of 5240 arrhythmia-ECG pairs. Simulation is very efficient, whereas diagnosis is much slower.

[2] Thanks to Christian Holzbaur for implementing this modifications

In the following we will report results of our experiments. We were using the 1) original model, 2) a version using just *domain variables*, [3] and 3) CLP(gRel). The necessary changes to the original model for experiment 2) were rather small - we just redefined the original `member` predicate to make use of *domain variables* where appropriate. For experiment 3) changes were a bit more involved; the complete model was semi-automatically transformed to make use of *r-variables* wherever possible. One example of this transformation has already been presented above - the `gen_ect` predicate. Both transformations were verified by exhaustive computation of all solutions and comparison to the original model for missing or extra arrhythmia-ECG pairs.

The following table shortly summarizes the results. It reports on the runtimes for finding either just one or all arrhythmias possibly explaining a given ECG. Numbers are averaged over all 3096 different ECGs. *Runtime* is the time measured in milliseconds/ECG, and *Speedup* is the ratio of the runtimes:

Approach	Runtime-One	Speedup-One	Runtime-All	Speedup-All
1. Original	671	1.00	1080	1.00
2. Domains	176	4.02	282	2.82
3. CLP(gRel)	56	11.98	96	11.25

We see that although explicit handling of disjunctions involves some overhead in building up and manipulating appropriate data-structures representing the disjuncts, the achieved considerable reduction in fruitless backtracking still leads to a overall speedup of an order of magnitude for either finding a single or for finding all possible solutions.

5 Discussion and Further Research

The following is a comparison of our approach to other possible approaches. Except for [4] we are not aware of any experiments similar to ours. Typically constraint propagation is used to compute some form of local consistency, be it *node-* or *arc-* or *path-(of some length)consistency* [3] plus backtrack search in the reduced solution space. E.g. the `element` predicate in combination with *forward-checking* as it is introduced in [6] essentially ensures arc-consistency. Our approach results in global consistency for given variables and is able to combine partial solutions dynamically thus interweaving backtrack search with consistency checks. This approach is of course not a panacea for all kind of search problems. It is successful for the KARDIO model exactly because there are only a few solutions (this seems to prevent combinatorial explosion of the size of the intermediate structures representing the possible disjuncts) and the interaction of the different constraints/sub-goals is to complex for finding a good ordering (either statically or dynamically) exploitable by chronological backtracking. As the table given in the previous section shows, CLP(gRel)

[3] In an earlier paper we have reported on results implementing *domain variables* in standard Prolog [11]; for the experiments reported in this paper we have used the above mentioned modified Sicstus-Prolog for a straight-forward implementation of *domain variables* by means of *attributed variables*

can sometimes even be used successfully if one is only interested in one single of all possible solutions, but this is certainly not true in general. Thus e.g. the n-queens problem is definitely not a fruitful application area whereas e.g. scene-labeling (waltz-filtering) probably is.

We are currently investigating whether other internal representations of the disjuncts can be managed more efficiently. Furthermore we have already completed a prototype implementation generalizing the ground value-tupels to arbitrary structures allowing for variables and sub-goal calls. Right now this suffers from a severe runtime penality due to excessive copying and we are not sure, if there can be a remedy to this problem. On the other hand this further generalization allows for an even more radical transformation of a search problem to an explicit construction of the search space with no backtracking at all. The transformed application programs additionally bear some resemblance to or-parallel logic program languages, a relationship that has to be researched in more depth. We are also searching for more declarative ways of specifying extended unification involving *attributed variables* which are themselves related to one or more other variables.

6 Acknowledgements

I am indebted to Igor Mozetic for providing the KARDIO model, to Christian Holzbaur for providing the modified Sicstus-Prolog, to both of them for discussions on the topic, and especially to Robert Trappl for creating a very special working environment. This work was supported by the Austrian Federal Ministry of Science and Research.

References

[1] Bratko I., Mozetic I., Lavrac N.: *Kardio - A Study in Deep and Qualitative Knowledge for Expert Systems*, MIT Press, Cambridge, MA, 1989.

[2] Carlsson M., Widen J.: *Sicstus Prolog Users Manual*, Swedish Institute of Computer Science, SICS/R-88/88007C, 1990.

[3] Guesgen H.W., Hertzberg J.: Some Fundamental Properties of Local Constraint Propagation, *Artificial Intelligence*, 36(2)237-247, 1988.

[4] Freuder E.C.: Synthesizing Constraint Expressions, in *Communications of the ACM*, 21(11), 1978.

[5] Hentenryck P.van, Dincbas M.: Domains in Logic Programming, in *Proceedings of the Fifth National Conference on Artificial Intelligence (AAAI-86)*, Morgan Kaufmann, Los Altos, CA, 1986.

[6] Hentenryck P.van: *Constraint Satisfaction in Logic Programming*, MIT Press, Cambridge, MA, 1989.

[7] Holzbaur C.: Specification of Constraint Based Inference Mechanisms through Extended Unification, Institut fuer Med.Kybernetik u. AI, Universitaet Wien, Dissertation, 1990.

[8] Huitouze S.le: A new data structure for implementing extensions to Prolog, in Deransart P., Maluszunski J.(eds.), *Programming Language Implementation and Logic Programming*, Springer, Heidelberg, 136-150, 1990.

[9] Jaffar J., Michaylov S.: Methodology and Implementation of a CLP System, in *Logic Programming - Proceedings of the 4th International Conference - Volume 1*, MIT Press, Cambridge, MA, 1987.

[10] Neumerkel U.: Extensible Unification by Metastructures, *Proc. META90*, 1990.

[11] Pfahringer B.: Constraint Propagation in Qualitative Modeling: Domains Variables Improve Diagnostic Efficiency, in *Proceedings of the Eighth Conference on Artificial Intelligence and Simulation of Behaviour (AISB91)*, Springer, London, UK, 1991.