

***A*-System: Problem Solving through Abduction**

Antonis C. Kakas
Dep. of Computer Science
University of Cyprus
B.O.Box 20537
CY-1678 Nicosia, Cyprus
antonis@ucy.ac.cy

Bert Van Nuffelen and Marc Denecker
Dep. of Computer Science
K.U.Leuven
Celestijnenlaan 200A
B-3000 Leuven, Belgium
{bertv, marcd}@cs.kuleuven.ac.be

Abstract

This paper presents a new system, called the *A*-System, performing abductive reasoning within the framework of Abductive Logic Programming. It is based on a hybrid computational model that implements the abductive search in terms of two tightly coupled processes: a reduction process of the high-level logical representation to a lower-level constraint store and a lower-level constraint solving process. A set of initial "proof of principle" experiments demonstrate the versatility of the approach stemming from its declarative representation of problems and the good underlying computational behaviour of the system. The approach offers a general methodology of declarative problem solving in AI where an incremental and modular refinement of the high-level representation with extra domain knowledge can improve and scale the computational performance of the framework.

1 Introduction

Over the last two decades it has become clear that abduction can play a central role in addressing a variety of problems in Artificial Intelligence. These problems include diagnosis [Poole *et al.*, 1987; Console *et al.*, 1996], planning [Missiaen *et al.*, 1995; Kakas *et al.*, 2000; Shanahan, 2000] knowledge assimilation and belief revision, [Inoue and Sakama, 1995; Pagnucco, 1996] multi-agent coordination, [Ciampolini *et al.*, 2000; Kowalski and Sadri, 1999] and knowledge intensive learning [Muggleton, 2000; Mooney, 2000].

The essential feature of this abductive approach to problem solving is the fact that it allows the application problems to be formalized directly in their high-level declarative representation. A close link therefore emerges between declarative problem solving in AI and the logical reasoning of abduction. However, despite this variety of applications for abduction and its potential benefits it has not been easy to develop general systems for abduction that are computationally effective for problems of practical size and complexity.

This paper presents a new system, called the *A*-System, that supports abductive reasoning within the framework of Abductive Logic Programming (ALP) [Kakas *et al.*, 1998; Denecker and Kakas, 2000]. In this framework problems are

represented in a high-level declarative way with logic programming rules and classical first-order sentences (integrity constraints). Our work aims to examine the possibility of developing ALP into a declarative problem solving framework, suitable for a variety of AI problems, that is computationally viable for problems of practical scale.

A principled implementation of the *A*-System is developed based on a hybrid computational model that formalizes the abductive search for a solution in terms of two interleaving processes: the logical reduction of the high-level representation of the problem and lower-level constraint solving. The abductive search is linked tightly to the construction of an associated constraint store.

To validate our approach we have carried out a set of "proof of principle" experiments with an initial implementation of the system that aim to test (a) the general underlying computational behaviour of the *A*-System on different domains and (b) the flexibility of the framework to incorporate additional problem specific knowledge in a modular and computationally enhancing way. In particular, some of these experiments are designed to test the extend to which the high-level representation of problems can be computationally improved via extra domain knowledge added to it. The experiments considered include constraint satisfaction problems and standard AI Planning Systems Competition problems.

The *A*-System has been developed as a follow up of two earlier ALP systems, the ACLP system [Kakas *et al.*, 2000] and SLDNFAC [Denecker and Schreye, 1998], bringing together features from these two systems. The main development in the design of the *A*-System over these previous systems is the fact that the non-determinism in the abductive computation is now made explicit allowing the possibility of implementing this as a form of parameterized heuristic search coupled with a process of deterministic propagation of the state of computation. The *A*-System is available from www.cs.kuleuven.ac.be/~dtai/kt.

2 Declarative Programming with Abduction

Declarative Problem Solving with a high-level representation of the problem at hand is closely related to *abduction*. The reason for this stems from the fact that in a declarative representation where one describes the expert knowledge on the problem domain rather than some method for its solution often, the task of solving the problems consists in filling the

missing information from this representation pertaining to the goal at hand. Typically, in a logical framework of representation the solution consists in finding the extension of some predicate(s) which are incompletely specified in the representation. The high-level theory describing the problem is then extended to a new one such that the problem goal is satisfied.

Computing such extensions of the theory representing our problem is an *abductive task*. Indeed, abduction as a problem solving method assumes that the general data structure for the solution to a problem (or solution carrier) is at the predicate level and hence that a solution is described in the same terms and level as the problem itself.

A framework for problem solving with abduction therefore needs to be expressing enough to allow high-level declarative representations of complex problems with missing or incomplete information. At the same time for such a framework to have a practical value it should provide ways of improving the computational effectiveness of this high-level representation for any particular problem. One such framework that combines representational expressiveness and computational flexibility is that of Abductive Constraint Logic Programming (ACLP) a framework that integrates together Abductive Logic Programming (ALP) with methods from Constraint Logic Programming (CLP). The A-system is developed within this framework. Let us briefly review the underlying framework of ALP.

A recent way to formalize ALP is to view this as a special case of ID-logic [Denecker, 2000], a logic extending classical logic with inductive definitions. Let a language of predicates, $\mathcal{L}$, be given consisting of three disjoint types of predicates: (i) P_D the *defined* predicates, (ii) A the *open or abducible* predicates and (iii) C the *constraint* predicates. Then a theory in ALP is defined as follows.

Definition 2.1 *An abductive theory is a triple (P, A, IC) where:*

- *P is a constraint logic program where only defined predicates appearing in the head of these rules.*
- *A is a set of ground abducible atoms over open predicates.*
- *IC is a set of first order formulae over $\mathcal{L}$, called integrity constraints.*

The representation of an application problem in an abductive theory (P, A, IC) splits, in this view of ALP in terms of ID-logic, into two parts. The set of rules in P represents the expert's strong *definitional knowledge* of the problem, i.e. knowledge which fully determines one or a group of predicates, the *defined* predicates, in terms of other *open* predicates. Each rule represents a *case* in which the defined predicate in the head will be true; the program P is an exhaustive enumeration of the cases. The open or *abducible* predicates have no definition. But some general information about them may be available indirectly through the expert's weaker *assertional knowledge* of the problem. This is represented by the theory IC of *integrity constraints*. The set A enumerates all the possible missing "values" for the open predicates. Abductive solutions are build from A giving a ground (partial) definition of these open predicates.

The constraint predicates are defined, as in CLP, by an underlying constraint theory which is independent of any particular abductive theory (P, A, IC) . We will assume that the constraint theory is a finite domain theory that also includes equality over logical terms. The formal details of this are beyond the scope of this paper.

In ALP the abductive problem task is defined as follows.

Definition 2.2 *Given an abductive theory (P, A, T) and some query Q , consisting of a conjunction of literals over $\mathcal{L}$, an abductive solution (or explanation) for Q is a set $\Delta \subseteq A$ of ground abducible atoms together with an answer substitution θ such that $P \cup \Delta$ is consistent and:*

- $P \cup \Delta \models \forall(\theta(Q))$
- $P \cup \Delta \models IC$.

This definition is generic in that it defines the notion of an abductive solution in terms of any given semantics of standard (constraint) Logic Programming (LP). Each particular choice of semantics defines its own entailment relation $\models$ and hence its own notion of what is an abductive solution. In the context of ID-logic, its inductive definition semantics essentially coincides with the well-founded model semantics [Gelder *et al.*, 1991] for LP when this is a two valued model¹. In the rest of this paper we will be adopting this well-founded model semantics for LP for the development of our A-System.

A computed abductive solution Δ therefore gives a ground definition of the open predicates which in turn through the logic program, $P \cup \Delta$, extends this to the defined predicates.

As an example of an abductive theory we give below a part of the theory representing a planning domain. In this, *drive_vehicle*, is the only open predicate, $<$, $\leq$, $\neq$ are constraint predicates and all others are defined predicates. The predicates *vehicle*, *location* and *time* are simple predicates defining the finite domain of variables, e.g. using CLP notation *vehicle(Truck) : -Truck in 1..10* for a problem where we have ten trucks.

vehicle_at(Truck, Loc, T) : -
init_vehicle_at(Truck, Loc),
 $\neg$ clipped_vehicle_at(Truck, Loc, 0, T)².

vehicle_at(Truck, Loc, T) : -
drive_vehicle(Truck, Loc, E),
 $E < T$,
 $\neg$ clipped_vehicle_at(Truck, Loc, E, T).

clipped_vehicle_at(Truck, Loc1, E, T) : -
drive_vehicle(Truck, Loc2, C),
 $Loc2 \neq Loc1$,
 $E \leq C, C < T$.

prec_vehicle(Truck, Loc, T) : -
vehicle(Truck),
location(Loc),
time(T),
petrol_in(Truck, T).

$\forall Trc, Loc, T.$
 $\underline{drive_vehicle(Trc, Loc, T) \rightarrow prec_vehicle(Trc, Loc, T)}.$

¹The formal details of this are beyond the scope of this paper.

The initial state of the problem is given by a set of facts in the logic program P including statements of the form $init_vehicle_at(Truck, Loc)$, e.g. $init_vehicle_at(truck1, city1.1)$.

3 $\mathcal{A}$ -System and Abductive Search

The computation of an abductive solution in the $\mathcal{A}$ -System can be seen as a process of reduction of the query, Q , to a set of abducible hypotheses on the open predicates together with an associated constraint store $\mathcal{C}_S$ of constraints over the general finite domain constraint theory supported by the particular ALP framework. This is similar to the computation of CLP with two important differences.

First the reduction involves through the hypotheses also a reduction of the integrity constraints IC . Effectively, when a new abductive hypothesis is made the integrity constraints need to be re-evaluated to remain true. This satisfaction of the IC can result in new constraints for the constraint store $\mathcal{C}_S$ and possibly new goals at the same level as the original query. The second difference with CLP is the way that the constraint store $\mathcal{C}_S$ is used in this reduction. As we will see in the next subsection, this is not merely a passive store of constraints to be evaluated at the end of the reduction but it is used actively during the reduction to affect the search. For example, it enables reductions to be pruned early by setting new constraints in $\mathcal{C}_S$ provided that this remains satisfiable.

3.1 Abductive Inference in the $\mathcal{A}$ -System

The logical reduction of a query Q in the $\mathcal{A}$ -System can be defined as a derivation for Q through a rewriting process between states. We give here in brief the formal details of this.

A computational restriction on the form of the integrity constraints, IC , of an abductive theory (P, A, IC) is imposed in the $\mathcal{A}$ -System. These sentences must be (or should be transformed to classically equivalent sentences) of the form: $\forall \neg(l_1, \dots, l_n)$ where each l_i is a literal. Such an integrity constraint will also be called a denial or a negative goal and will be denoted by $\leftarrow l_1, \dots, l_n$. Also to simplify the presentation, we assume that each rule in P is of the form $p(\overline{X}) \leftarrow B[\overline{X}, \overline{X}_i]$ where all variables are (implicitly) universally quantified, referred to as *free* variables, and $B[\overline{X}, \overline{Y}]$ is a conjunction of literals with free variables $\overline{X}$ occurring in the head and $\overline{Y}$ occurring only in the body. Rules are therefore in homogeneous form where no non-variable term can appear in their head e.g. a rule $p(c) \leftarrow q(Y)$ will be written equivalently as $p(X) \leftarrow X = c, q(Y)$.

A state S of the derivation consists of two types of elements: a set of literals (possibly with free variables), called positive goals, and a set of denials, called negative goals, of the form $\forall \overline{X}. \leftarrow F[\overline{X}, \overline{Y}]$ where $F[\overline{X}, \overline{Y}]$ is a conjunction of literals and $\overline{Y}$ are free variables. $\Delta(S)$ denotes the set of abducible atoms in S , i.e. positive goal atoms whose predicate is an open predicate and $\mathcal{C}(S)$ denotes the set of positive constraint atoms in S . A derivation for Q starts with an initial state consisting of the literals of the query Q as positive goals and all the denials in IC as negative goals. The rewriting derivation proceeds by selecting a literal in a goal of S and

applying a suitable inference rule yielding a new state. The main inference rules are given by the following rewrite rules (different elements of S are separated by ";"):

- $p(\overline{t}) \Rightarrow B_i[\overline{t}]$ if p is a defined predicate and $p(\overline{X}) \leftarrow B_i[\overline{X}]$ a rule.
- $p(\overline{t}) \Rightarrow \overline{t} = \overline{s}$ if p is an open predicate and $p(\overline{s}) \in S$.
- $\neg A \Rightarrow \leftarrow A$.
- $\forall \overline{X}. \leftarrow p(\overline{t}), Q \Rightarrow F_1; \dots; F_n$ (p a defined predicate) where $F_i = \forall \overline{X}'. \leftarrow B_i[\overline{t}, \overline{X}_i], Q$ for each rule $p(\overline{Y}) \leftarrow B_i[\overline{Y}, \overline{X}_i], (\overline{X}' \subseteq \overline{X})$.
- $\forall \overline{X}. \leftarrow p(\overline{t}), Q \Rightarrow \forall \overline{X}. \leftarrow p(\overline{t}), Q; F_1; \dots; F_n$ (p an open predicate) where $F_i = \forall \overline{X}'. \leftarrow \overline{t} = \overline{s}, Q$ for each $p(\overline{s}_i)$ an abduced atom in S , $(\overline{X}' \subseteq \overline{X})$.
- $\forall \overline{X}. \leftarrow C, Q \Rightarrow \neg C$, if C is a constraint atom without universally quantified variables.
- $\forall \overline{X}. \leftarrow C, Q \Rightarrow C; \forall \overline{X}. \leftarrow Q$, if C is a constraint atom without universally quantified variables.
- $\forall \overline{X}. \leftarrow \neg A, Q \Rightarrow A$ if A does not contain universally quantified variables.

A successful derivation terminates with a state S such that:

1. S contains positive goals only of the form of abducible atoms or constraint atoms,
2. negative goals in S are denials containing some open atom $p(\overline{t})$ which has already been selected and resolved with each abduced atom $p(\overline{s}) \in S$, and
3. the constraint store $\mathcal{C}(S)$ of S is satisfiable.

Otherwise, the derivation flounders when universally quantified variables appear in the selected literal in a denial. If the derivation does not succeed or flounder then it fails. Note that negation as failure in P is simply re-written (via the 3rd rule) to a denial of the positive atom.

Let S be the final state of a successful derivation. Then any substitution θ that assigns a ground term to each free variable of S and which satisfies the constraint store $\mathcal{C}(S)$ is called a *solution substitution* of S . Such a substitution always exists since $\mathcal{C}(S)$ is satisfiable for a successful derivation.

Theorem 3.1 *Let (P, A, IC) be an abductive theory s.t. $P \models IC$, Q a query, S the final state of a successful derivation for Q and θ a solution substitution of S . Then the pair $\theta(\Delta(S))$ and θ is an abductive solution of Q .*

3.2 Abductive Search

The search for an abductive solution in the $\mathcal{A}$ -System based on the above proof theory, depends very closely on the computed constraint store. Typically, the abducible hypotheses made during the computation are non-ground atoms whose variables are restricted by the constraints in the associated constraint store of the computation. This constraint store plays a central role not only in carrying (together with the abducibles) the solution but also in controlling the abductive search.

In many respects the computation can be viewed as a process of constructing a constraint store from the high-level specification and query of the abductive theory as any decision that we take, e.g. which abducible to introduce or how to satisfy an integrity constraint, extends differently the current constraint store. Hence we can turn things around and use the constraint store to guide the decisions that we make in this abductive search.

The overall pattern of the abductive computation in the *A*-System is as follows:

1. *Deterministic Propagation of +ve and -ve Goals*
2. *Suspend and Evaluate Choices*
 - (a) *Quasi-consistency of -ve Goals*
 - (b) *Evaluate Choices on +ve goals*
3. *Global set of Choices: new +ve Goals, G_{new}*
4. *If G_{new} non-empty then Return to (1)*
5. *Otherwise, Exit and Ground Solution.*

The computation starts with a phase of *deterministic propagation* where all of the current goals that have only one possible rewriting are reduced until no such goals are left in the resulting state of the computation. The purpose of this phase is two-fold: (a) to expose and collect all the choice points in the current state of the computation and (b) to propagate the construction of the constraint store with all new constraints that are necessarily imposed under the choices made so far in the previous iteration steps. The updated constraint store is checked for satisfiability during this phase as it grows. This can have a significant effect on the computation as it enables us to detect early the ensuing failure of the choices made prior to this before committing to other choices.

All the choice points exposed by this deterministic phase are suspended and a new process of their evaluation begins. This explicit handling of the non-determinism in the computation allows the use of a parameterized form of heuristic search where a variety of different types of heuristics can be used. There are essentially two types of choices in the computation. These are (1) a choice of which rule from the program to use in rewriting a positive goal (cf. first and second rewrite rule) and (2) a choice of which way to satisfy a denial (cf. last three rewrite rules).

The choice points of the second type are considered in a phase called *quasi-consistency*³ where we try to satisfy all the denials in the current state together if possible without the introduction of new positive goals. Choices result in a new possible constraint store, that we can compute by a subsidiary phase of deterministic propagation. The "quality" of this ensuing constraint store under various criteria, e.g. tightness of its finite domain variables or minimal change in the current constraint store, forms a possible heuristic criterion. A second criterion concerns the number and complexity of the new positive goals, that a choice would produce. For example, preferring new goals which are deterministic or which introduce new abducibles whose associated denials would

³This name reflects the fact that at this stage of the computation the satisfaction of the integrity constraint denials is in general contingent on a set of new goals.

have a simple form are different possibilities that we are currently exploring here in the implementation of the *A*-System.

After the phase of quasi-consistency the computation examines the other choice points left suspended, namely positive goals which are non-deterministic, and again it can apply a form of heuristic evaluation to choose amongst the different possibilities. Once a global decision is taken on all these choice points the computation either returns to the top to repeat the process, when the choice leads to new positive goals, or otherwise, it terminates successfully by labeling the constraint store and thus grounding the abductive solution.

This *global* approach, of the *A*-System to choice making where it tries to cover together as many choice points as possible results in more informed decisions leading to less backtracking and thus to a better computational performance. It also facilitates the use of heuristics by helping to parameterize them and make them available at the top level.

An important parameter of the computation is the degree to which the constraint store is examined for its satisfiability (and other quality measures) during its construction. One possibility is to do a full check of the whole store at each point where this grows. But this can be costly as a large part of this maybe unnecessary. On the other hand an incomplete check may result in the late (after several choices) recognition of the inconsistency of the constraint store. An intermediate possibility is to localizing the check to the latest addition to the store, i.e to the variables of the current constraints and other variables connected to these by constraints already in the store. This induces a form of dynamic partition of the constraint store into connected components that reflects, to a certain extent, the high-level structure of the problem and query at hand and could help minimize the computation required to check the satisfiability of the store.

4 Experiments with the A-System

A number of different experiments have been carried out to test the underlying general computational behaviour of (an initial implementation of) the *A*-System and how this can be affected by extending the high-level representation of problems with additional information.

The current version of the system is implemented as a meta-program on top of Sicstus Prolog 3.8.5 and uses its finite domain constraint solver. It is developed as an experimentation tool to test different datastructures and strategies. Currently a form of indexing on the abducibles and constraints is used in the datastructures. With respect to the parameters of the abductive search presented in the previous section the current version implements only a very basic form of heuristics. In particular, the methods used for checking the satisfiability of the constraint store during its construction are simple.

A first set of experiments consists of problems whose specification can be reduced to a finite domain constraint store in one phase of reduction. Such problems include standard constraint satisfaction problems, e.g. N-Queens, Graph Coloring and Scheduling problems. The aim here was to confirm that the execution will be deterministic and to compare the execution time with that of solving the problem directly in CLP to see the overhead cost of the reduction of the high-level rep-

resentation of the problem. The tables below show a sample of these experiments for the N-Queens problem and a graph coloring problem. For the graph coloring experiment a set of 4-colorable planar graphs were constructed. The run times are split into the time needed to find an abductive solution and the subsequent time needed to ground this by the CLP labeling. All experiments reported in this paper are done on a Linux machine (800 Mhz) with 512 MB of memory.

N-Queens		
size	abductive solution (s)	clp grounding
30	0.7	10ms
40	1.9	20ms
50	4.1	160ms
60	8.9	160ms

Graph Coloring		
size(nodes)	abductive solution (s)	clp grounding
80	1.9	10ms
100	3.7	20ms
200	21.3	20ms
400	233.3	50ms

Although the reduction of the whole specification is done in one step (no backtracking occurs over the abductive solution) in both experiments the construction time (abductive solution) increases clearly with the size. So the metainterpreter is some orders slower than the Prolog below, which is used in a CLP-program to set up the constraints.

Another set of experiments was carried out on planning problems. These are non-deterministic problems where the abductive computation and search of the $\mathcal{A}$ -System can be tested. We selected from the latest AIPS2000 planning competition two domains: blocks world and logistics. The next table shows the performance of the system on the blocks world domain. For this experiment we used a specification with one action $move(X,Y,T)$, which denotes moving a block X onto Y at time T.

Problem	Planlength	Time(s)
20-0	37	1.7
30-1	57	3.5
45-0	88	10.2
55-1	105	17.2
60-0	114	30.8

The second considered domain is that of logistics. For this domain a specification with functors and the general event calculus is used. Furthermore the specification is two layered: the top layer is a compact high level description of the logistics domain. The lower level actions (from the original AIPS strips description) are derived from the high level ones. The first table below shows the performance of the $\mathcal{A}$ -System on problems where the system has been used to compute only high-level abstract plans. The length of the plans are given by the final time when the goal is achieved (first number in the column) and the number of abducted actions describing the actual plan. A blank entry indicates that the system was unable to find a solution within a given time limit.

The results for the low level plans are presented in a second table. These plans are computed in a separated phase using as input the previous computed high level plans. This step expands the high level plans in number of time points and

actions.

High level solutions

Problem	Length	Time(s)	Max time points
7-0	8/15	6.3	100
8-0	9/19	7.8	100
9-0	8/16	34.1 (*)	100
10-0	7/18	8.5	100
11-0	-	-	-
12-0	9/18	298.4 (*)	100
12-0	9/18	19.9	20

Low level solutions

Problem	Length	Time
7-0	21/40	40ms
8-0	21/47	50ms
9-0	15/39	40ms
10-0	18/46	50ms
12-0	24/45	50ms

Some of the problems took a while to return a solution. Because the finite domain solver is incomplete, the $\mathcal{A}$ -System may go on with a branch although the constraint store is already unsatisfiable (see the marked entries in the above tables). We tried different satisfiability checks but none of them was overall successful. A parameter in such a check is the domain size of the variables (see the last column of the table of the high level plans): by decreasing the maximal number of time points the satisfiability check fails faster when the constraint store is unsatisfiable. As consequence some problems could be solved in a reasonable time. Currently, we are investigating how we can improve this satisfiability check.

Compared to previous experiments with SLDNFAC and ACLP, the $\mathcal{A}$ -System is more reliable and more robust to changes in the order of execution of the query. However because the constraint solver is incomplete it still possible to have one derivation of $\mathcal{A}$ -System failing to find a solution and another one finding it immediately. Another main difference of the $\mathcal{A}$ -System with these previous systems is its separation of search and inference. This allows easy experimentation with several strategies and different degrees of deterministic propagation.

5 Related Work and Conclusions

There are several other abductive systems for ALP that have recently been developed. Different methods have been used e.g. bottom up computation [Iwayama and Satoh, 2000], tabling [Alferes *et al.*, 1999] and rewriting rules with the completion [Fung and Kowalski, 1997; Kowalski *et al.*, 1998]. The $\mathcal{A}$ -System with its two earlier ALP systems, the ACLP system and SLDNFAC, is to our knowledge the first abductive system that has paid particular attention to the computational aspects of abduction using constraint solving extensively to control the abductive search and enhance its computational behaviour.

On a more abstract level our work is related to Answer Set Programming(ASP) [Gelfond and Lifschitz, 1991]. Strong connections have been established [Satoh and Iwayama, 1991] between ALP and ASP. At the level of semantics these two frameworks are for a large class of theories (ASP admits only a special type of integrity constraints) equivalent.

At the level of computation our construction of an abductive solution in the $\mathcal{A}$ -System corresponds to the generation of a model in ASP but there are some significant differences in the respective computational models. It is therefore instructive to develop systematic experiments to compare systems from these two approaches to declarative problem solving.

The abductive search in the $\mathcal{A}$ -System can be parametrized in a variety of ways to reflect the type of cooperation with constraint solver and the heuristics that are used. An important future development of the system is to structure further this parametric space and to study how more techniques from constraint programming and advanced heuristic search can be incorporated in order to improve the general computational behaviour of the framework on a variety of problems. In particular, we can study how recent heuristic methods for planning [Bonet and Geffner, 1999] can be generalized to the abductive computation of the $\mathcal{A}$ -System.

This general improvement of the underlying computational efficiency of the ALP framework, or indeed of any other declarative problem solving framework, is clearly limited as we are aiming to use the framework on a general variety of problems. A complementary line of development of the $\mathcal{A}$ -System and its associated ALP framework concerns the study of how to provide a programming environment where the user has the facility to incrementally refine her/his high-level representation of the problem in a modular way. A general methodology for declarative problem solving with abduction is emerging where the ALP modeling environment provides the possibility for extra problem specific (declarative or control) knowledge to be included that would enhance its computational performance.

References

- [Alferes *et al.*, 1999] J. J. Alferes, L. M. Pereira, and T. Swift. Well-founded abduction via tabled dual programs. In *Proceedings of ICLP-99*, 1999.
- [Bonet and Geffner, 1999] B. Bonet and H. Geffner. Planning as heuristic search: New results. In *Proc. European Conference on Planning (ECP-99)*, 1999.
- [Ciampolini *et al.*, 2000] A. Ciampolini, E. Lamma, P. Mello, and P. Torroni. An implementation for abductive logic agents. In *AI*IA'99, LNAI Vol. 1792*, 2000.
- [Console *et al.*, 1996] L. Console, L. Portinale, and D. Theiseider Dupré. Using compiled knowledge to guide and focus abductive diagnosis. *IEEE Transactions on Knowledge and Data Engineering*, 8 (5):690–706, 1996.
- [Denecker and Kakas, 2000] M. Denecker and A.C. Kakas. *Abductive Logic Programming*. Special issue of JLP, Vol 44 (1-3), 2000.
- [Denecker and Schreye, 1998] M. Denecker and D. De Schreye. Sldnfa: an abductive procedure for abductive logic programs. *Logic Programming*, 34(2):111–167, 1998.
- [Denecker, 2000] M. Denecker. Extending classical logic with inductive definitions. In *Proceedings of CL-2000, 703-717*, 2000.
- [Fung and Kowalski, 1997] T.H. Fung and R.A. Kowalski. The iff procedure for abductive logic programming. *Logic Programming*, 33(2):151–165, 1997.
- [Gelder *et al.*, 1991] A. Van Gelder, K.A. Ross, and J.S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.
- [Gelfond and Lifschitz, 1991] M. Gelfond and V. Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, pp. 365–387, 1991.
- [Inoue and Sakama, 1995] K. Inoue and C. Sakama. Abductive framework for nonmonotonic theory change. In *Proceedings of IJCAI-95*, pages 204–210, 1995.
- [Iwayama and Satoh, 2000] N. Iwayama and K. Satoh. Computing abduction by using tms with top-down expectation. *Logic Programming*, 44(1-3):179–206, 2000.
- [Kakas *et al.*, 1998] A.C. Kakas, R.A. Kowalski, and F. Toni. The role of abduction in logic programming. In D. Gabbay, C. Hogger, and J. Robinson, editors, *Handbook of Logic in AI and Logic Programming*, volume 5, pages 235–324. Oxford University Press, 1998.
- [Kakas *et al.*, 2000] A.C. Kakas, A. Michael, and C. Mourlas. Aclp: Abductive constraint logic programming. *Journal of Logic Programming: Special Issue on Abductive Logic Programming*, 44(1-3):129–177, 2000.
- [Kowalski and Sadri, 1999] R.A. Kowalski and F. Sadri. From logic programming to multi-agent systems. *Annals of Mathematics and Artificial Intelligence*, 1999.
- [Kowalski *et al.*, 1998] R.A. Kowalski, F. Toni, and G. Wetzel. Executing suspended logic programs. *Fundamenta Informaticae*, 34(3):203–224, 1998.
- [Missiaen *et al.*, 1995] L.R. Missiaen, M. Denecker, and M. Bruynooghe. CHICA, an abductive planning system based on event calculus. *Journal of Logic and Computation*, 5(5):579–602, 1995.
- [Mooney, 2000] R.J. Mooney. Integrating abduction and induction in machine learning. In *Abduction and Induction: essays on their relation and integration*, pages 181–191. Kluwer Academic Press, 2000.
- [Muggleton, 2000] S. Muggleton. Theory completion in learning. In *Inductive Logic Programming, ILP-00*, 2000.
- [Pagnucco, 1996] M. Pagnucco. *The role of abductive reasoning within the process of belief revision*. PhD Thesis, Dept. of Computer Science, University of Sydney, 1996.
- [Poole *et al.*, 1987] D. Poole, R. Goebel, and R. Aleliunas. Theorist: A logical reasoning system for defaults and diagnosis. In *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer-Verlag, 1987.
- [Satoh and Iwayama, 1991] K. Satoh and N. Iwayama. Computing abduction by using the tms. In *Proc. of ICLP'91*, pages 505–518, 1991.
- [Shanahan, 2000] M. Shanahan. An abductive event calculus planner. *Logic Programming*, 44(1-3):207–239, 2000.