Abduction in Logic Programming

Marc Denecker[1] and Antonis Kakas[2]

[1] Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium.
Marc.Denecker@cs.kuleuven.ac.be
http://www.cs.kuleuven.ac.be/∼marcd
[2] Department of Computer Science, University of Cyprus,
75 Kallipoleos St., Nicosia, Cyprus.
antonis@ucy.ac.cy
http://www.cs.ucy.ac.cy/∼antonis

Abstract. Abduction in Logic Programming started in the late 80s, early 90s, in an attempt to extend logic programming into a framework suitable for a variety of problems in Artificial Intelligence and other areas of Computer Science. This paper aims to chart out the main developments of the field over the last ten years and to take a critical view of these developments from several perspectives: logical, epistemological, computational and suitability to application. The paper attempts to expose some of the challenges and prospects for the further development of the field.

1 Introduction

Over the last two decades, abduction has been embraced in AI as a non-monotonic reasoning paradigm to address some of the limitations of deductive reasoning in classical logic. The role of abduction has been demonstrated in a variety of applications. It has been proposed as a reasoning paradigm in AI for diagnosis [8,90], natural language understanding [8,39,4,93], default reasoning [81,29,25,50], planning [28,110,71,59], knowledge assimilation and belief revision [54,76], multi-agent systems [7,64,102] and other problems.

In the context of logic programming, the study of abductive inference started at the end of the eighties as an outcome of different attempts to use logic programming for solving AI-problems. Facing the limitations of standard logic programming for solving these problems, different researchers proposed to extend logic programming with abduction. Eshghi [28] introduced abduction in logic programming in order to solve planning problems in the Event Calculus [65]. In this approach, abduction solves a planning goal by *explaining* it by an ordered sets of events -a plan- that entails the planning goal. This approach was further explored by Shanahan [110], Missiaen et al. [72,71], Denecker [21], Jung [48] and recently in [59,60]. Kakas and Mancarella showed the application of abduction in logic programming for deductive database updating and knowledge assimilation [53,55]. The application of abduction to diagnosis has been studied in [10,11]

A.C. Kakas, F. Sadri (Eds.): Computat. Logic (Kowalski Festschrift), LNAI 2407, pp. 402–436, 2002.

within an abductive logic programming framework whose semantics was defined by a suitable extension of the completion semantics of LP.

In parallel to these studies of abduction as an inferential method, Eshghi and Kowalski [29] and later Kakas and Mancarella in [52,54] and Dung in [25], used abduction as a semantical device to describe the non-monotonic semantics of Logic Programming (in a way analogous to Poole in [81]). In [18,14], abductive logic programming was investigated from a knowledge representation point of view and its suitability for representing and reasoning on incomplete information and definitional and assertional knowledge was shown.

For these reasons, Abductive Logic Programming[1] (ALP) [50,51] was recognized as a promising computational paradigm that could resolve many limitations of logic programming with respect to higher level knowledge representation and reasoning tasks. ALP has manifested itself as a framework for declarative problem solving suitable for a broad collection of problems.

Consequently, at the start of the 90s, a number of abductive systems were developed. In [54], the abductive procedure of [29] for computing negation as failure through abduction was extended to the case of general abductive predicates. Another early abductive procedure was developed in [10] using the completion. [17] proposed SLDNFA, an extension of SLDNF with abduction allowing non-ground abductive hypotheses. [21] proposed an extension of SLDNFA with a constraint solver for linear order and demonstrated that this system could be applied correctly for partial order planning in the context of event calculus. Later, the idea of integrating abduction and constraint solving was developed more generally in the ACLP framework [56,61,60]; this procedure is the result of incorporating CLP constraint solving in the abductive procedure of [54]. In [37] an abductive procedure that can be regarded as a hybrid of SLDNFA and the procedure of Console et al has been defined based on explicit rewrite rules with the completion and equality. This has later [66] incorporated constraint solving in a similar way to the ACLP procedure. A bottom up procedure, later combined with some top down refinement, was given in [106] and [42]; the latter system was an implementation using the Model Generator MGTP developed on the multiprocessor machine developed at ICOT. Another recent abductive procedure in LP is that of AbDual [1] which exploits tabling techniques from XSB.

Despite these efforts and the many potential applications for abduction, it has taken considerable time and effort to develop computationally effective systems based on abduction for practical problems. The field has faced (and to some extend continues to do so) a number of challenges at the logical, methodological and implementational level. In the recent past, important progress has been made on all these levels. The aim of this chapter is to give a comprehensive

[1] The July/August 2000 volume (Vol. 44) of the journal of Logic Programming is a special issue on Abductive Logic Programming. This contains several papers that open new perspectives on the relationship between abduction and other computational paradigms.

overview of the state of the art of Abductive Logic Programming, to point to problems and challenges and to sketch recent progress.

Bob Kowalski has been one of the founders of Abductive Logic Programming. Recently, he has, together with others, proposed [64] that ALP can be used as a framework in which we can integrate an agent's knowledge on how to reduce its goal to subgoals and thus plan how to achieve this goal, described in the program part of an abductive theory, together with the agent's obligations, prohibitions and other elements that determine its reactive behaviour, described in the integrity constraints part of its abductive theory. In this suggestion ALP plays a central role in capturing the behaviour of an autonomous agent that feeds from and reacts to its environment.

This together with his view of its role in the way that Logic Programming should evolve more generally as a programming language of the future is described elegantly in his short position statement on the future of Logic Programming in this volume.

The rest of the paper is organized as follows. Section 2 briefly reviews the study of abduction in AI and philosophy and situates Abductive Logic Programming within this broad context. Section 3 gives the formal definition of abduction, and reviews the different formal semantics that have been proposed in the literature. Section 4 reviews the different ALP frameworks that have been developed so far analyzing their potential scope to applications and their links to other extensions of LP. The paper ends with a discussion of future challenges and prospects of development for the field of ALP.

2 What Is Abduction?

2.1 What Is an Explanation?

The term abduction was introduced by the logician and philosopher C.S. Pierce (1839-1914) who defined it as the inference process of forming a hypothesis that explains given observed phenomena [77]. Often abduction has been defined broadly as any form of "inference to the best explanation" [47] where *best* refers to the fact that the generated hypothesis is subjected to some optimality criterion. This very broad definition covers a wide range of different phenomena involving some form of hypothetical reasoning. Studies of "abduction" range from philosophical treatments of human scientific discovery down to formal and computational approaches in different formal logics.

In the context of formal logic, abduction is often defined as follows. Given a logical theory T representing the expert knowledge and a formula Q representing an observation on the problem domain, abductive inference searches for an explanation formula $\mathcal{E}$ such that:

- $\mathcal{E}$ is satisfiable[2] w.r.t. T and
- it holds that[3] $T \models \mathcal{E} \rightarrow Q$

[2] If $\mathcal{E}$ contains free variables, $\exists(\mathcal{E})$ should be satisfiable w.r.t. T.
[3] Or, more general, if Q and $\mathcal{E}$ contain free variables: $T \models \forall(\mathcal{E} \rightarrow Q)$.

In general, $\mathcal{E}$ will be subjected to further restrictions such as the aforementioned minimality criteria and criteria restricting the form of the explanation formula (e.g. by restricting the predicates that may appear in it). This view defines an abductive explanation of an observation as a formula which *logically entails* the observation. However, some have argued, sometimes with good reasons, that it is more natural to view an explanation as a *cause* for the observation [47]. A well-known example is as follows [92]: the disease *paresis* is caused by a latent untreated form of syphilis. The probability that latent untreated syphilis leads to paresis is only 25%. Note that in this context, the direction of entailment and causality are opposite: syphilis is the cause of paresis but does not entail it, while paresis entails syphilis but does not cause it. Yet a doctor can *explain* paresis by the hypothesis of syphilis while paresis cannot account for an *explanation* for syphilis.

In practice, examples where causation and entailment do not correspond are rare[4]. It turns out that in many applications of abduction in AI, the theory T describes explicit *causality information*. This is notably the case in model-based diagnosis and in temporal reasoning, where theories describe effects of actions. By restricting the explanation formulas to the predicates describing primitive causes in the domain, an explanation formula which entails an observation gives a cause for the observation. Hence, for this class of theories, the logical entailment view implements the causality view on abductive inference.

2.2 Relationship to Other Reasoning Paradigms

As mentioned in the previous section, the definition of abduction is very broad and covers a wide range of hypothetical reasoning inference that could otherwise be formally distinguished. Not surprisingly, there are many different views on what is abduction and how to implement it. Many philosophers and logicians have argued that abduction is a generalization of *induction* [34]. Induction can be defined as inference of general rules that explain certain data. A simple example illustrating inductive inference is the following derivation:

$$\frac{\begin{array}{l} human(socrates) \\ mortal(socrates) \end{array}}{\forall x.human(x) \rightarrow mortal(x)}$$

Hence, induction can also be seen as a form of *inference to the best explanation*.

The term abduction as used in this paper, refers to a form of reasoning that can be clearly distinguished from *inductive inference*. In most current applications of abduction the goal is to infer *extentional knowledge*, knowledge that is specific to the particular state or scenario of the world. In applications of induction, the goal is to infer *intentional knowledge*, knowledge that universally holds

[4] See [91] where the relation between causal and evidential modeling and reasoning is studied and linked to abduction.

in many different states of affairs and not only in the current state of the world. For example, an abductive solution for the problem that a certain car does not start this morning is the explanation that its battery is empty. This explanation is extentional. On the other hand, an inductive inference is to derive from a set of examples the rule that if the battery is empty then the car will not start. This is *intentional knowledge*. As a consequence of this distinction, abductive answers and inductive answers have a very different format. In particular, inductive answers are mostly general rules that do not refer to a particular scenario while abductive answers are usually simpler formulas, often sets of ground atoms, that describe the causes of the observation in the current scenario according to a given general theory describing the problem domain. This distinction in the form of the answer induces in turn strong differences in the underlying inference procedures.

Abduction as a form of inference of extentional hypotheses explaining observed phenomena, is a versatile and informative way of reasoning on incomplete or uncertain knowledge. Incomplete knowledge does not entirely fix the state of affairs of the domain of discourse while uncertain knowledge is *defeasible* in the sense that its truth in the domain of discourse is not entirely certain. In the presence of uncertain knowledge, [81] demonstrated how abductive inference can be used for default reasoning. In the presence of incomplete knowledge, abduction returns an explanation formula corresponding to a (non-empty) collection of possible states of affairs in which the observation would be true or would be caused; on the other hand deduction is the reasoning paradigm to determine whether a statement is true in all possible states of affairs. As such, abduction is strongly related to *model generation* and *satisfiability checking* and can be seen as a refinement of these forms of reasoning. By definition, the existence of an abductive answer proves the satisfiability of the observation. But abduction returns more informative answers; answers which describe the properties of a class of possible states of affairs in which the observation is valid.

2.3 Abduction and Declarative Knowledge Representation

An important role of logic in AI is that of providing a framework for *declarative problem solving*. In this, a human expert specifies his knowledge of the problem domain by a descriptive logic theory and uses logical inference systems to solve computational tasks arising in this domain. Although in the early days of logic-based AI, *deduction* was considered as the unique fundamental problem solving paradigm of declarative logic [70] in the current state of the art deduction has lost this unique place as the central inferential paradigm of logic in AI. Indeed, we argue that a declarative problem solving methodology will often lead to abductive problems. Let us illustrate this with an example problem domain, that of university time tabling.

The process of declarative problem solving starts with the *ontology design* phase: during this step, an alphabet of symbols denoting the relevant objects, concepts and relationships in the problem domain must be designed. This alphabet defines the *ontology* of the problem domain. It precedes the *knowledge description* phase during which the expert expresses his knowledge using the

symbols of the alphabet. In the example domain of university timetables we have three important types of objects: *lectures*, *time slots* and *class rooms*. We could represent them using the predicates $lecture/1$, $time\_slot/1$ and $room/1$. Important relevant relationships between them refer to when and where lectures take place; these relationships could be represented by predicates $time\_of\_lecture/2$ and $room\_of\_lecture/2$.

A key observation is that even though at this stage the knowledge specification has not even started, the choice of the alphabet already determines that certain tasks will be abductive. In particular, the task of computing a correct time table will consist of computing tables for $time\_of\_lecture/2$ and $room\_of\_lecture/2$ that satisfy certain logical constraints imposed on correct schedules. This task is not a deductive task: the "correct" tables will not be deducible from the theory. Rather it is an abductive problem — or a model generation problem[5] — a problem of completing the problem description so that the goal (or "observation") that all lectures are scheduled, holds.

The ontology design phase has a strong impact on the specification phase. If the alphabet is complex and does not have a simple correspondence to the objects, concepts, relations and functions of the domain of discourse, this will complicate the knowledge description phase and lead to a more complex, more verbose, less comprehensive and less modular specification. For example, one simple constraint is that each lecture must be scheduled at some time slot and room:

$$\forall l : lecture(l) \rightarrow \exists t, r : time\_slot(t) \wedge time\_of\_lecture(t, l) \wedge room(r) \wedge room\_of\_lecture(r, l)$$

Another constraint is that two lectures cannot take place in the same room at the same time:

$$\forall t, r, l1, l2.\, room\_of\_lecture(r, l1) \wedge room\_of\_lecture(r, l2) \wedge time\_of\_lecture(t, l1) \wedge time\_of\_lecture(t, l2) \rightarrow l1 = l2$$

If we had represented the assignments of rooms and time slots to lectures by balanced binary tree structures, this would have complicated significantly the expression of these requirements which are now expressed directly. Thus, an important aspect of the declarative problem solving methodology, is that the ontology and alphabet is designed in a task independent way such that it naturally matches with the types of objects and relationships that occur in the problem domain.

The above example illustrates that the choice of the alphabet may enforce the use of a specific type of inference to solve a specific computational task, and that, when we follow a declarative approach this will often lead to the problem tasks to be abductive in nature. Vice versa, the a-priori choice of a specific inferential

[5] Recall the close relationship between abduction (as viewed in this paper) and model generation, as explained in the previous section.

system such as Prolog or a CLP system to solve a specific computational problem has a strong impact on the choice of the alphabet for that problem domain. In the university time tabling problem, a Prolog or CLP solution will not be based on the use of predicates $time\_of\_lecture/2$ and $room\_of\_lecture/2$ but on another, in this case more complex alphabet, typically one in which predicates range over lists or trees of assignments of lectures, time slots and rooms. This alphabet is more complex and is choosen in a task-dependent way, which results in reduced readability and modularity of the problem specification and in a reduced reusability of the specification to solve other types of tasks. The fact that the alphabet in Prolog (and CLP) programming must be chosen in relation to the provided inference system rather than by its match with the human experts conception of the problem domain is one of the fundamental reasons why even pure Prolog programs are rarely perceived as truly *declarative*.

There is clearly a trade-off here. On the one hand, the choice of an alphabet in correspondence with the concepts and relations in the mind of the human expert is a prerequisite to obtain a compact, elegant and readable specification. As a result of this often the computational task of problem solving links tightly to abduction. Putting this more directly we would argue that Declarative Knowledge Representation comes hand in hand with abduction. Abduction then emerges as an important computational paradigm that would be needed for certain problem solving tasks within a declarative representation of the problem domain.

On the other hand, in practice the choice of a representation is not governed merely by issues relating to a natural representation but also and sometimes even more by issues of computational effectiveness. Although the use of a more complex ontology may seriously reduce the elegance of the representation, it may be necessary to be able to use a specific system for solving a problem; in some cases this would mean that the pure declarative problem representation needs to be augmented with procedural, heuristic and strategic information on how to solve effectively the computational problem.

Current research on abduction studies how more intelligent search and inference methods in abductive systems can push this trade-off as far as possible in the direction of more declarative representations.

3 Abductive Logic Programming

This section presents briefly how abduction has been defined in the context of logic programming.

An Abductive Logic Programming theory is defined as a triple (P, A, IC) consisting of a logic program, P, a set of ground abducible atoms A[6] and a set of classical logic formulas IC, called the integrity constraints, such that no atom $p \in A$ occurs in the head of a rule of P.

In the field of Abductive Logic Programming, the definition of abduction is usually specialized in the following way:

[6] In practice, the abducibles are specified by their predicate names.

Definition 1. *Given an abductive logic theory* (P, A, IC)*, an abductive explanation for a query* Q *is a set* $\Delta \subseteq A$ *of ground abducible atoms such that:*

- $P \cup \Delta \models Q$
- $P \cup \Delta \models IC$
- $P \cup \Delta$ *is consistent.*

Some remarks are in order. First, this definition is *generic* both in terms of syntax and semantics. Often, the syntax is that of normal logic programs with negation as failure but some have investigated the use of abduction in the context of extended logic programming [43] or constraint logic programming [56,60,66]. At the level of semantics, the above definition defines the notion of an abductive solution in terms of any given semantics of standard logic programming. Each particular choice of semantics defines its own entailment relation $\models$, its own notion of consistent logic programs and hence its own notion of what an abductive solution is. In practice, the three main semantics of logic programming — completion, stable and well-founded semantics — have been used to define different abductive logic frameworks.

A second remark is that an abductive explanation Δ aims to represent a *nonempty* collection of states of affairs in which the explanandum Q would hold. This explains the third condition that $P \cup \Delta$ should be consistent.

Third, when integrity constraints IC are introduced in the formalism, one must define *how* they constrain the abductive solutions. There are different views on this. Early work on abduction in Theorist in the context of classical logic [81], was based on the *consistency view* on constraints. In this view, any extension of the given theory T with an abductive solution Δ is required to be consistent with the integrity constraints IC: $T \cup IC \cup \Delta$ is consistent. The above definition implements the *entailment view*: the abductive solution Δ together with P should entail the constraints. This view is the one taken in most versions of ALP and is stronger than the consistency view in the sense that a solution according to the entailment view is a solution according to the consistency view but not vice versa.

The difference between both views can be subtle but in practice the different options usually coincide. E.g. it frequently happens that $P \cup \Delta$ has a unique model, in which case both views are equivalent. In practice, many ALP systems [20,61] use the entailment view as this can be easily implemented without the need for any extra specialized procedures for the satisfaction of the integrity constraints since this semantics treats the constraints in the same way as the query.

The above definition aims to define the concept of an abductive solution for a query but does not define *abductive logic programming* as a logic in its own right as a pair of syntax and semantics. However, a notion of *generalized model* can be defined, originally proposed in [52], which suggests the following definition.

Definition 2. M *is a model of an abductive logic framework* (P, A, IC) *iff there exists a set* $\Delta \subseteq A$ *such that* M *is a model of* $P \cup \Delta$ *(according to some LP-semantics) and* M *is a classical model of* IC*, i.e.* $M \models IC$*.*

The entailment relation between abductive logic frameworks and classical logic formulas is then defined in the standard way as follows:

$(P, A, IC) \models F$ *iff for each model* M *of* (P, A, IC), $M \models F$.

Note that this definition is also generic in the choice of the semantics of logic programming. This way, abductive extensions of stable semantics [52], of well-founded semantics [79] and the partial stable model semantics [111] have been defined. Also the completion semantics has been extended [10] to the case of abductive logic programs. The completion semantics of an abductive logic framework (P, A, IC) is defined by the mapping it to its completion. This is the first order logic theory consisting of :

- UN, the set of unique names axioms, or Clark's equality theory.
- IC
- $comp(P, A)$, the set of completed definitions for all non-abducible predicates.

A recent study [16] that attempts to clarify further the representational and epistemological aspects of ALP, has proposed ID-logic as an appropriate logic for ALP. ID-logic is defined as an extension of classical logic with inductive definitions. Each inductive definition consists of a set of rules defining a specific subset of predicates under the well-founded semantics. This logic gives an epistemological view on ALP in which an abductive logic program is a *definition* of the set of the non-abducible predicates and abducible predicates are *open predicates*, i.e. not defined. The integrity constraints in an abductive logic framework are simply classical logic *assertions*. Thus the program P represents the human expert's strong *definitional knowledge* and the theory IC represents the human expert's weaker *assertional knowledge*. Therefore in ID-logic, ALP can be seen as a sort of *description logic* in which the program is a TBOX consisting of *one* simultaneous definition of the non-abducible predicates, and the assertions correspond to the ABOX [115].

4 Abductive Logic Programming Frameworks

The framework defined in the previous section is generic in syntax and semantics. In the past ten years, the framework has been instantiated (and sometimes has been extended) in different ways. In order to show the wider variety of motivations and approaches that are found in Abductive Logic Programming, this section aims to present briefly a number of these alternative frameworks, implemented systems and applications. These different instantiations differ from each other by using different formal syntax or semantics, or sometimes simply because they use a different inference method and hence induce a different procedural semantics.

4.1 Approaches under the Completion Semantics for LP

Abduction through Deduction. One of the first ALP frameworks is that of [10]. The syntax in this framework is that of hierarchical logic programs[7] with a predefined set of abducible predicates. The formal syntax is an extension of Clark's completion semantics [9] in which only the non-abducible predicates are completed. The main aim of this work was to study the relationship between abduction and deduction in the setting of non-monotonic reasoning. In particular, many characterizations of non-monotonic reasoning such as circumscription, predicate completion, explanatory closure implement a sort of *closure principle* allowing to extract implicit negative information out of explicit positive information. What is shown in this work is that for a restricted class of programs, the abductive explanations to a query with respect to a set of (non-recursive) rules can be characterized in a deductive way if we apply the completion semantics as a closure principle.

Formally, given a (hierarchical) abductive logic program P with abducibles A, its completion P_C consists of iff-definitions for the non-abducible predicates. These equivalences allow to rewrite any observation O to an equivalent formula F in the language of abducible predicates such that $P_C \models O \leftrightarrow F$ where $\models$ is classical logic entailment. The formula F, called the *explanation formula*, can be seen as a disjunctive characterization of all abductive solutions of O given P. The restriction to hierarchical programs ensures termination of a procedure to compute the explanation formula. The framework has been extended to handle (a restricted form of)integrity constraints.

The above abductive framework has been used to formalize diagnostic problem solving and classification in nonmonotonic inheritance hierarchies [10,24], and has been extended to characterize updates in deductive databases [12]. The completion semantics is also the basis for the "knowledge compilation" optimization of abductive problem solving described in [11].

The IFF Framework. The IFF framework is also based on the completion semantics. It was initially developed as a unifying framework integrating abduction and view updating [36,37]. The IFF proof procedure is defined by a rewriting system in which an initial goal is rewritten to a disjunction of answers. The main rewrite rules are unfolding, namely backward reasoning with the iff definitions, and propagation, namely forward reasoning with the integrity constraints. IFF produces answers to goals in the form of conjunctions of abducible atoms and denial integrity constraints. An extension of it with special treatment of built-in predicates and constraint logic programming was proposed in [120,66]. Another modification of the IFF proof procedure was developed for applications modeling reasoning of rational agents [64] and management of active rules in databases [96]. The main underlying LP semantics used in this framework is Fitting's three-valued completion semantics but correctness results have been proven also for perfect model semantics and under some restrictions for stable semantics.

[7] A hierarchical program is one without recursion.

Prototype implementations of the three instances of the IFF procedure [37,97] exist and have been applied in many experiments. The original IFF proof procedure has been implemented in Java and was applied within a Voyager extension to the problem of interaction and communication amongst multiple agents, as well as cooperative problem solving. It was also used for information integration from multiple sources [95], to the management of information networks [112], and it has been integrated with PROGOL to learn preconditions of actions in the frameworks of the event and situation calculi. The extension presented in [120,66] has been applied to job-shop scheduling [66] and semantic query optimization [121]. The procedure suggested in [96] was implemented in *April*, and was used in the context of applications for active databases and agents. Recently, it has been used to study the problem of resource allocation in a multi-agent environment [102].

4.2 Approaches under Stable and Well-Founded Semantics

In Logic Programming other semantics have been proposed as refinements of the completion semantics. These include the stable model semantics [38] and the well-founded model semantics [116]. The following ALP frameworks use these semantics for their underlying LP framework.

SLDNFA and ID-logic. SLDNFA [17,20] is an abductive extension of SLDNF-resolution [68], suitable for abductive reasoning in the context of (possibly recursive) abductive logic programs under the completion semantics. It was proven sound and, under certain restrictions, complete with respect to the 3-valued completion and well-founded semantics. This procedure came out of the early attempts to implement AI-planning using abductive reasoning in the event calculus. It was one of the first procedures that correctly handles non-ground abduction, i.e. abduction of atoms with variables. The procedure was also used in one of the first experiments of integration of abduction and constraint solving. [21] describes an extension of SLDNFA with a constraint solver for the theory of total order and applies it for partial order planning and in the context of temporal reasoning with incomplete knowledge.

At the logical level, the work evolved into a study of the role of ALP for knowledge representation and of SLDNFA for abductive and deductive reasoning. A number of subsequent experiments with ALP and SLDNFA demonstrated the role of ALP for knowledge representation of incomplete and temporal knowledge [19,113,114,22]. To explain and clarify the representational and epistemological aspects of ALP, [16] proposed ID-logic, an integration of classical logic with inductive definitions under well-founded semantics.

At the computational level, efforts were done to improve the computational performance and expressivity of the original implementation of the SLDNFA procedure. The SLDNFAC system [117] is developed at the K.U.Leuven and implements abduction in the context of ID-Logic, supporting directly general first order classical axioms in the language and higher order aggregates.

The system integrates constraint solving with the general purpose abductive resolution SLDNFA. It is implemented as a meta-interpreter on top of Sicstus prolog and is available from http://www.cs.kuleuven.ac.be/ dtai/kt/systems-E.shtml.

The SLDNFAC system has been used in the context of prototypical constraint solving problems such as N-queens, logical puzzles, planning problems in the blocks world, etc ... for proving infinite failure of definite logic programs [5], failure of planning goals [13] and for semantic interpretation of temporal information in natural language [119]. An extension of the system has also been used in the context of a scheduling application for the maintenance for power units of power plants. This experiment involves the use of higher order aggregates and is described in detail in section 5.1. Recently, [78] compared SLDNFAC with different other approaches for solving constraint problems including CLP, ACLP and the Smodels system [74] and shows that in many problems the system is competitive.

Bottom up Abduction. This approach was proposed originally in [107] and aims to develop efficient techniques for computing abductive solutions under the generalized stable model semantics [52] by translating the abductive logic program to a standard logic program and applying efficient bottom up stable model generators to this translation. This approach is based on a translation of Abductive logic programs into pure logic programs with stable model semantics [107]. Abductive solutions w.r.t. the original abductive logic program correspond to stable models of its translation[8]. To compute abductive solutions, [107] also proposed a procedure for bottom-up stable model computation based on truth maintenance techniques. It is an extension of the procedure for computing well-founded models of [33,94] and dynamically checks integrity constraints during the computation of stable models and uses them to derive facts. Later this bottom up procedure was integrated with a procedure for top-down expectation [108,46]. This top-down procedure searches for atoms and rules that are relevant for the query (and the integrity constraints) and thus helps to steer the search into the direction of a solution. This procedure has been used for a number of applications in the following two domains.

Legal Reasoning: A dynamic notion of similarity of cases in legal reasoning is implemented using abductive logic programming. The input of this system is legal factors, case bases and the current case and position of user (defendant or plaintiff). The system translates the case bases and the current case into an abductive logic program. Using the top-down proof procedure the system then computes important factors and retrieves a similar case based on the important factors and generates an explanation why the current case is similar to the retrieved case which is preferable to user's position [105]. The system has also been extended

[8] The correctness of this transformation of abductive logic programs to pure logic programs has been shown to be independent of the stable model semantics, and has been extended to handle integrity constraints [111].

so that legal rules and legal cases are combined together for statutory interpretation [103].

Consistency Management in Software Engineering: This system computes a minimal revised logical specification by abductive logic programming. A specification is written in Horn clauses which is translated into an abductive logic program. Given an incompatibility between this specification and new information the system computes by abduction a maximally consistent program that avoids this incompatibility [104].

ACLP: Abductive Constraint Logic Programming. The ACLP framework grew as an attempt to address the problem of providing a high-level declarative programming or modeling environment for problems of Artificial Intelligence which at the same time has an acceptable computational performance. Its roots come from the work on abduction and negation as failure in [29] and the early definitions of Abductive Logic Programming [52,53,50]. Its key elements are (i) the support of abduction as a central inference of the system, to facilitate declarative problem solving, and (ii) the use of Constraint Logic Programming techniques to enhance the efficiency of the computational process of abductive inference as this is applied on the high-level representation of the problem at hand.

In an ACLP abductive theory the program, P, and the integrity constraints, IC, are defined over a CLP language with finite domain constraints. Its semantics is given by a form of Generalized Model semantics which extends (in the obvious way) the definition 1 above when our underlying LP framework is that of CLP. Negation in P is given meaning through abduction and is computed in a homogeneous way as any other abducible. The general computation model of ACLP consists of a cooperative interleaving between hypotheses and constraint generation, via abductive inference, with consistency checking of abducible assumptions and constraint satisfaction of the generated constraints. The integration of abductive reasoning with constraint solving in ACLP is cooperative, in the sense that the constraint solver not only solves the final constraint store generated by the abductive reduction but also affects dynamically this abductive search for a solution. It enables abductive reductions to be pruned early by setting new suitable CLP constraints on the abductive solution that is constructed.

The framework of ACLP has also been integrated with Inductive Logic Programming to allow a form of machine learning under incomplete information [62].

The ACLP system [60,49], developed at the University of Cyprus, implements the ACLP framework of ALP for a restricted sub-language of the full ACLP framework. Currently, the system is implemented as a meta-interpreter on top of the CLP language of ECLiPSe using the CLP constraint solver of ECLiPSe to handle constraints over finite domains (integer and atomic elements). The architecture of the system is quite general and can be implemented in a similar way with other constraint solvers. It can be obtained, together with information on how to use it, from the following web address: http://www.cs.ucy.ac.cy/aclp/. Direct comparison experiments [61]

of ACLP with the underlying CLP system of ECLiPSe have demonstrated the potential of ALP to provide a high-level modeling environment which is modular and flexible under changes of the problem, without compromising significantly the computational efficiency of the underlying CLP framework. ACLP has been applied to several different types of problems. Initial applications have concentrated on the problems of scheduling, time tabling and planning. Other applications include (i) optical music recognition where ACLP was used to implement a system that can handle recognition under incomplete information, (ii) resolving inconsistencies in software requirements where (a simplified form of) ACLP was used to identify the causes of inconsistency and suggest changes that can restore consistency of the specification and (iii) intelligent information integration where ACLP has been used as a basic framework in the development of information mediators for the semantic integration of information over web page sources. Although most of these applications are not of "industrial scale" (with the notable exception of a crew-scheduling [57] application for the small sized company of Cyprus Airways - see also below 5.2) they have been helpful in indicating some general methodological guidelines that can be followed when one is developing abductive applications (see [57]). The air-crew scheduling application produced solutions that were judged to be of good quality, comparable to manually generated solutions by experts of many years on the particular problem, while at the same time it provided a flexible platform on which the company could easily experiment with changes in policy and preferences.

Extended and Preference Abduction. In order to broaden the applicability of ALP in AI and databases, Inoue and Sakama propose two kinds of extensions of ALP: *Extended abduction* [43] and *Preference abduction* [44].

An abductive program in the framework of extended abduction is a pair $\langle K, \mathcal{A} \rangle$ of logic programs possibly including negation as failure and disjunctions. Each instance of element of $\mathcal{A}$ is *abducible*. An explanation of a ground literal G consists of a pair of sets (I, O) of subsets of $\mathcal{A}$ such that $(K \setminus O) \cup I$ is consistent and entails G. An *anti-explanation* of G satisfies the same conditions except that $(K \setminus O) \cup I$ does not entail G. Thus, abduction in this framework extends standard abduction by defining not only explanation but also *anti-explanations*, by allowing solutions in which rules from the program are deleted and by allowing general rules to be abduced or deleted.

Several implementation methods have been proposed for computing extended abduction. [45] proposed a model generation method with term rewriting. In [99,41], transformation methods are proposed that reduce the problem of computing extended abduction to a standard abductive problem. Extended abduction has several potential applications such as abductive theory revision and abduction in non-monotonic theories, view update in deductive databases, theory update, contradiction removal, system repair problems with model checking, and inductive logic programming (see [43,99,41]).

A framework for preference abduction is an abductive logic program $\langle K, \mathcal{A} \rangle$ augmented with a set Ψ of possible priorities between different literals of the program. For a given goal G, preferred abduction computes a set of

abducible atoms I and a subset ψ of Ψ representing some priority relation, such that $K \cup I$ is consistent and $K \cup I \models_\psi G$, which means that G is true in every *preferred answer set* of the *prioritized logic program* $(K \cup I, \psi)$ [98]. Hence, preferred abduction not only abduces atoms but also the priority relationship. A procedure to compute preference abduction has been proposed in [44].

Preference abduction can be used in resolution of the multiple extension problem in non-monotonic reasoning, skeptical abduction, reasoning about rule preference, and preference view update in legal reasoning [44].

ABDUAL: Abduction in extended LP [1] proposes the ABDUAL framework, an abductive framework based on extended logic programs. An abductive logic program in this framework is a tuple $< P, \mathcal{A}, IC >$, where P is an extended logic program (with both explicit and default negation), IC a set of constraints and $\mathcal{A}$ a set of ground objective literals i.e. atoms or explicitly negated atoms. The declarative semantics of this formalism is based on the well-founded semantics for extended programs.

The procedure presented in [1] integrates a tabling mechamism in the abductive inference procedure. The procedure solves an abductive query in two stages. First, the program is transformed by grounding it and adding for each non-abducible ground atom p a rule $not(p) \leftarrow R$ where R expresses that none of the rules for p applies. The resulting program is called the *dual* program. In the second step, abductive solutions are computed by an evaluation method that operates on the dual program.

The ABDUAL system is currently implemented on top of XSB-Prolog [122]. The system is available from http://www.cs. sunysb.edu/~tswift. Work is currently being done in order to migrate some of the tabling mechanisms of ABDUAL, now taken care of the meta-interpreter, into the XSB-engine. Work is also underway on the XSB system so that the co-unfounded set removal operation can be implemented at the engine level.

The ABDUAL system has been applied in medical psychiatric diagnosis [31] as a result of an investigation into the logical representation and automation of DSM-IV (Diagnostic and Statistical Manual of Mental Disorders). The current user interface of the *Diagnostica* system (http://medicinerules.com) uses abduction in a simple but clinically relevant way to allow for hypothetical diagnosis: when there is not enough information about a patient for a conclusive diagnosis, the system allows for hypothesizing possible diagnosis on the basis of the limited information available. This is one of the first applications of abduction that is been commercialized.

ABDUAL has also been employed to detect specification inconsistencies in model-based diagnosis system for power grid failure [6]. Here abduction is used to abduce hypothetical physically possible events that might cause the diagnosis system to come up with a wrong diagnosis violating the specification constraints.

Probabilistic Horn Abduction and Independence Choice Logic. Probabilistic Horn abduction [84], later extended into the independent choice logic [86], is a way to combine logical reasoning and belief networks into a simple

and coherent framework. Its development has been motivated by the Theorist system [88] but it has been extended into a framework for decision and game-theoretic agents that includes logic programs, belief networks, Markov decision processes and the strategic form of a game as special cases. In particular, it has been shown that it is closely related to Bayesian networks [80], where all uncertainty is represented as probabilities.

An independent choice logic theory is made up of two parts:

- a choice space consisting of disjoint sets of ground atoms. The elements of a choice space are called alternatives.
- an acyclic logic program such that no element of an alternative unifies with the head of a clause.

The semantics is model-theoretic. There is a possible world for each choice of one element from each alternative. What is true in a possible world is given by the stable model of the atoms chosen and the logic program. Intuitively the logic program gives the consequences of the choices. This framework is abductive in the sense that the explanations of an observation g provide a concise description of the worlds in which g is true. Belief networks can be defined by having independent probability distributions over the alternatives. Intuitively, we can think of nature making the choice of a value for each alternative. In this case Bayesian conditioning corresponds exactly to the reasoning of the above framework of independent choice logic. This can also be extended to decision theory where an agent can make some choices and nature others [86], and to the game-theoretic case where there are multiple agents who can make choices.

Different implementations of the ICL and its various special cases exist. These include Prolog-style implementations that find explanations top-down [83,89], bottom-up implementations (for the ground case) that use a probabilistic variant of the conflicts used in model-based diagnosis [85], and algorithms based on efficient implementations of belief networks that also exploit the context-specific independent inherent in the rule forms [87]. Initial studies of application of ICL have centered around problems of diagnosis and robot control.

5 Example Applications of Abduction

ALP as a paradigm of declarative problem solving allows us to formalize a wide variety of problems. A survey of the field reveals the potential application of abduction in areas such as databases updates, belief revision, planning, diagnosis, natural language processing, default reasoning, user modeling, legal reasoning, multi-agent systems, scheduling, and software engineering. In this section, two relatively large-scale applications of ALP are presented in some detail in order to illustrate the main features of declarativeness and modularity of an abductive based approach that have been exposed in the previous sections.

5.1 Scheduling of Maintenance

This experiment is based on a real life problem of a Belgian electricity provider. The problem description is as follows. The company has a network of power plants, distributed over different areas and each containing several power producing units. These units need a fixed number of maintenances during the year. The problem is then to schedule these maintenances so that a number of constraints are satisfied and the risk of power shortage (and hence, import from other providers) is as low as possible. The problem was solved using the SLDNFAC system extended with a restricted yet sufficient form of higher-order aggregates. The system accepts first order constraints which are first compiled to rules using the Lloyd-Topor transformation. Below we given an overview of the problem solution. For a more complete description of the solution and the abductive procedure for reasoning on aggregates, we refer the reader to [117].

The fact that a maintenance M lasts from week B till week E, is represented by the predicate $start(M, B, E)$. This is the only abducible predicate in the specification. Other predicates are either defined or are input data and are defined by a table. Some of the main constraints that need to be satisfied are given below[9].

- Maintenances ($maint(M)$) and their duration ($duration(M, D)$) are given by a table. All maintenances must be scheduled, thus for each maintenance there exists an according *start* relation. This is specified via a first order logical formula i.e. an integrity constraint as follows:

$$\forall M : maint(M) \rightarrow \exists B, E, D : week(B) \wedge week(E) \wedge duration(M, D) \wedge E = B + D - 1 \wedge start(M, B, E).$$

- A table of $prohibited(U, Bp, Ep)$ facts specify that maintenances M for unit U are not allowed during the period $[Bp, Ep]$:

$$\forall U, Bp, Ep, M, B, E : prohibited(U, Bp, Ep) \wedge maint\_for\_unit(M, U) \wedge start(M, B, E) \rightarrow (E < Bp \vee Ep < B).$$

- For each week the number of the units in maintenance belonging to a plant P should be less than a maximal number Max. A table of $plant\_max(P, Max)$ atoms defines for each plant the maximal number of units in maintenance simultaneously.

$$\forall P, Max, We : plant(P) \wedge plant\_max(P, Max) \wedge week(We) \rightarrow \exists OnMaint : card(\{U | (unit(U) \wedge unit\_in\_plant(U, P) \wedge in\_maint(U, We))\}, OnMaint) \wedge OnMaint \leq Max.$$

  Note that this constraint uses the cardinality aggregate *card*. The meaning of the above cardinality atom is that the set of units of plant P in maintenance

[9] In this and the following section, variable names start with a capital, as standard in logic programming.

in week We contains $OnMaint$ elements. The predicate in_maint is defined by an auxiliary program rule specifying that a unit U is in maintenance during a certain week W if a maintenance M of this unit is going on during W:

$$in\_maint(U, W) \leftarrow maint\_for\_unit(M, U), start(M, B, E), B \leq W, W \leq E.$$

– Another constraint is that the capacity of the units in maintenance belonging to a certain area should not exceed a given area maximum. To represent this, the summation aggregate is needed. A table of $capacity(U, C)$ describes for each unit its maximum capacity.

$$\begin{aligned}\forall A, Max, We, CM : & area(A) \wedge area\_max(A, Max) \wedge week(We) \wedge \\ & sum(\{(U, C) | (unit(U) \wedge in\_area(U, A) \wedge in\_maint(U, We) \wedge \\ & \qquad capacity(U, C))\}, \lambda(U, Cap)Cap, CM) \\ & \rightarrow 0 \leq CM \wedge CM \leq Max.\end{aligned}$$

In the above constraint, the meaning of the sum aggregate atom is that "the sum of the lambda function over the set expression is CM". It defines CM as the total capacity of area A in maintenance during week We.

The above specification describes some of the necessary properties of a correct schedule. However, not all schedules satisfying these properties are desirable. In particular, schedules that minimise the risk of power shortage are preferable. To this end, the company maintains statistical data about the expected peak load per week. Desirable solutions are those that maximise the *reserve capacity*, that is the difference between the available capacity and the expected peak load. This relation ($reserve(Week, R)$) can then be defined as the difference between available capacity (the sum of capacities of all units not in maintenance during this week) and the estimated peak load:

$$\begin{aligned}reserve(We, R) \leftarrow & peakload(We, Load), total\_capacity(T), \\ & sum(\{(U, Cap) | (unit(U) \wedge in\_maint(U, We) \wedge \\ & \qquad capacity(U, Cap))\}, \\ & \quad \lambda(U, Cap)Cap, InMaint), \\ & R = T - Load - InMaint.\end{aligned}$$

in which $total\_capacity(T)$ means the sum of all capacities of all units.

In the SLDNFAC system, the query for the optimal solution for the scheduling problem is

```
? minimum(set([R],(exists(W) : reserve(W,R)),M), maximize(M).
```

It expresses that an optimal abductive solution is desired in which the minimal reserve for one year is as high as possible.

The actual problem, given by the company, consists of scheduling 56 maintenances for 46 units in one year (52 weeks). The size of the search space is of the order of 52^{46}. The current implementation reduces the goal and the integrity constraints to a large finite domain constraint store without backtracking points. In

the current implementation of SLDNFAC, this reduction phase is completed in less than one minute . Subsequently the CLP solver starts to generate solutions of increasing quality. The current implementation was able to find a solution which is 97% away from the optimal one in 20 minutes.

The same problem was also solved using a CLP system. A comparison between the CLP solution and the ALP solution clearly shows the trade-off between efficiency and flexibility. The pure (optimized) CLP solution will setup its constraint store in several seconds (3 to 4 seconds), and find the same solution as the above specification within 2 minutes (compared to 20 minutes for the SLDNFAC-solver). On the other hand, the CLP solution is a much larger program (400 lines) developed in some weeks of time in which the constraints are hidden within data structures, whereas the above representation in ALP is a simple declarative representation of 11 logical formulae, written down after some hours of discussion.

5.2 Air-Crew Assignment

The second application of abduction that we present is also based on a real-life problem, namely that of crew-assignment for Cyprus Airways. The problem of air crew-assignment is concerned with the assignment of air-crews to each of the flights that an airline company has to cover over some specific period of time. This allocation of crew to flights has to respect all the necessary constraints (validity) and also try to minimize the crew operating cost (quality). The validity of a solution is defined by a large number of complex constraints, which express governmental and international regulations, union rules, company restrictions etc. The quality of the schedule is specified, not only by its cost, but also by the needs and preferences of the particular company or crew at that specific period of time. In addition, an airline is also interested in the problem of re-assignment or of adapting an existing crew assignment to changes in the application environment such as flight delays or cancellations, new flight additions or crew unavailability etc. These changes often affect the quality of an existing solution or even make an existing solution unacceptable.

This problem for (the pilot crew) of Cyprus Airways was solved within ALP using the ACLP system. The problem was represented entirely as an ALP theory $T = (P, A, IC)$. The program part P describes basic data and defines a number of concepts that allow for encoding particular strategies for decomposing the overall goal to subgoals. Different strategies affect efficiency of the problem solving process and the quality of the solutions with respect to the criteria of cost or fairness of assignment. The solution of the problem is captured via an abducible predicate $assigns(Crew, Task)$ (the only member of A) which gives the assignment of crew members to different types of duty tasks (eg. flights, standbys, day-offs, etc.). For details of this and for a more complete description of the problem and its abductive-based solution see [58]. Here we will concentrate more on how the complex validity constraints of the problems are represented in the IC part of the theory.

The problem of air crew-assignment has a large variety of complex constraints that need to be respected. These contain simple constraints such as that a pilot can not be assigned to two overlapping flights but also many other quite complex constraints such as that during any period of 6 days (respectively 14 days) a pilot must have one day off (respectively 2 consecutive days off). Lets us illustrate how some of these would be represented as integrity constraints in IC. The following integrity constraint expresses the requirement that for any pilot there must be at least $MinRest$ hours rest period between any two consecutive duties. $MinRest$ is greater than or equal to 12 and it is calculated according to the previous assignments of the crew. (All variables in the integrity constraints below are universally quantified over the whole formula).

$$\neg assign(Crew, Flight) \leftarrow$$
$$on\_new\_duty(Crew, Flight),$$
$$end\_prev\_duty(Crew, Flight, EndOfDuty),$$
$$time\_difference(EndOfDuty, Flight, RestPeriod),$$
$$MinRest(Crew, MR), RestPeriod < MR.$$

Here $on\_new\_duty(Crew, Flight)$ defines whether the flight, $Flight$, is the beginning of a new duty period for $Crew$ and $end\_prev\_duty(Crew, Flight, EndOfDuty)$ specifies the time of the end of the duty, $EndOfDuty$, for the crew member, $Crew$, which is immediately before the departure time of the flight $Flight$. These are defined in the program P of the theory.

The requirement that each pilot must have at least 2 consecutive days off during any 14 day period is represented by the integrity constraint:

$$consec2\_daysoff(Crew, DeptDate, 14) \leftarrow$$
$$assign(Crew, Flight),$$
$$dept\_date(Flight, DeptDate)$$

where $consec2\_daysoff(Crew, DeptDate, 14)$ means that the $Crew$ has two consecutive days off within a time window of 14 days centered around the date $DeptDate$. This is given in the program P with the help of the definition of $dayoff$ as follows:

$$consec2\_daysoff(Crew, Date, N) \leftarrow$$
$$consec\_days(Date, N, DayA, DayB),$$
$$dayoff(Crew, DayA),$$
$$dayoff(Crew, DayB)$$

$$dayoff(Crew, Date) \leftarrow$$
$$not\ assign(Crew, flight(Id, Date)),$$
$$crew\_at\_base(Date),$$
$$further\_free\_hrs(Crew, Date)$$

$further\_free\_hrs(Crew, Date) \leftarrow$
$next\_date(Date, NDate)$,
$assign(Crew, flight(Id, NDate))$,
$departure(flight(Id, NDate), NDate, DeptTime), DeptTime > 8$

$further\_free\_hrs(Crew, Date) \leftarrow$
$next\_date(Date, NDate)$,
$assign(Crew, flight(Id, NDate))$,
$departure(flight(Id, NDate), NDate, DeptTime)$,
$DeptTime > 6, previous\_date(Date, PDate)$,
$assign(Crew, flight(Id, PDate))$,
$arrival(flight(Id, PDate), PDate, ArrTime), ArrTime < 22$

This expresses the definition of a day-off as a non-working day (0:00 - 24:00), at base, with one of the following additional requirements. Either the crew begins his/her duty after 8am the next morning, or s/he begins work after 6am but finishes before 10pm (22:00) the day before.

During the computation, the satisfaction of this integrity constraint means that whenever a new assumption of assignment of a Crew to a Flight is made we need to ensure that $consec2\_daysoff$ for this Crew member remains satisfied. In some cases this would then dynamically generate extra assignments, of the Crew member to day-offs, to ensure that his/her flight assignments are consistent.

Airlines also have their own requirements on the problem stemming from particular policies of the specific company and crew preferences. The abductive formulation, with its modular representation of the problem, facilitates in many cases a direct representation of these with additional integrity constraints in IC. As an example consider a requirement of Cyprus Airways which states that flight managers should not have more than two duties per week. This can be represented by the following integrity constraint:

$\neg assign(Crew, Flight) \leftarrow$
$rank(Crew, flight\_manager)$,
$on\_new\_duty(Crew, Flight)$,
$num\_of\_duties(Crew, Flight, week\_period, NDuties)$,
$NDuties > 2$.

Here $num\_of\_duties(Crew, Flight, week\_period, NDuties)$ counts the number of duties $NDuties$ that a crew member has within a $week\_period$ centered around the date of the flight $Flight$.

With regards to the problem of re-assignment under some new information, given an existing solution, a new module is added to the crew-assignment system which exploits the natural ability of abduction to reason with a given set of hypotheses, in this case the (partial) existing solution. This module follows three steps: (1) remove from the old solution all hypotheses which are affected by these changes. This step is in fact optional, helping only in the efficiency, since hypotheses which make the existing solution inconsistent will be eventually removed automatically by the re-execution of the abductive goal in step 3 below,

(2) add the new requirements (changes) of the problem. These may be in the form of integrity constraints or simply as new information in the domain of the application and (3) re-execute the (or part of the) abductive goal of the problem with the set of the hypotheses in step (1) as a given initial set of abducible assumptions.

Given the set of flights which are affected by the change(s), the aim is to re-establish the consistency, and preferably also the quality, of the old solution by re-assigning crew to these flights, without having to recalculate a new solution from the beginning but rather by making the fewest possible changes on the old existing solution, within 48 hours from the time of the change.

The re-assignment module in this application is interactive in the sense that the user can select a crew for a particular flight or decide whether to accept a system proposed selection of crew. Having searched for a crew member, the system informs the user about the particular selection, together with a list of other assignments (secondary changes) on this crew in the old solution, that are affected and would also need to be rescheduled. It then gives him/her the option to reject this choice, in which case the system will look for another possibility. When the selection of a crew is done directly by the user, the system will check if this choice is valid and inform the user of the list (if any) of secondary affected flights, that would also need to be rescheduled, resulting from this choice.

Although Cyprus Airways is a small size airline it contains the full complexity of the problem. During the busy months the flight schedule contains over 500 flight legs per month. The ACLP system was able to produce solutions in a few minutes which were judged by the airline's experts on this problem to be of good quality comparable (and with respect to balancing requirement often better) to the manually generated ones. The system was also judged to be useful due to the flexibility that it allowed to experiment easily with changes in policy and preferences of the company. The re-assignment module was able to suggest solutions on how to adapt the existing roster within at most 5 seconds. It was chosen as the most useful module of the system as it could facilitate the operators to develop and adjust a solution to meet the specific needs and preferences that they have at the time.

6 Links of ALP to Other Extensions of LP

In parallel with the development of the above frameworks and systems for ALP it has become clear that there exist strong links between some ALP frameworks and other extensions of Logic Programming.

ALP has tight connections to Answer Set Programming [32]. Recall that the ABDUAL framework [1] is an extension of Answer Set Programming with abduction. Standard ALP (with one negation) is strongly related Stable Logic Programming [69,75], the restriction of Answer Set Programming [32] to pure logic programs. As mentioned in section 4, an abductive logic framework under the generalized stable semantics can be translated in an equivalent logic program under stable semantics. Consequently, current systems for computing

stable models such as SMODELS [75] can be used to compute abduction under the generalized stable semantics. Interestingly, there are significant differences between in computational models that are developed in both areas. Whereas ALP procedures such as SLDNFA, IFF and ACLP are extensions of SLDNF and operate in a top down way on predicate programs, systems like SMODELS are based on bottom up propagation in the propositional grounding of a logic program. More experimentation is needed to assess the strengths and weaknesses of these approaches.

Links have been shown also between ALP and Disjunctive Logic Programming [100,101,124]. The hypothetical reasoning of ALP and the reasoning with disjunctive information of DLP can be interchanged. This allows theories in one framework to be transformed to the other framework and thus to be executed in this other framework. For example, it is possible to transform an ALP theory into a DLP one and then use a system such as the recently developed dlv system [27] to answer abductive queries. Vice versa, [124] showed that abductive proof procedures can be used as for reasoning on DLP programs.

Another type of extension of the LP paradigm is Inductive Logic Programming. Currently, several approaches are under investigation synthesizing ALP and ILP [2,73,123,35]. These approaches aim to develop techniques for knowledge intensive learning with complex background theories. One problem to be faced by ILP techniques is that the training data on which the inductive process operates often contain gaps and inconsistencies. The general idea is that abductive reasoning can feed information into the inductive process by using the background theory for inserting new hypotheses and removing inconsistent data. Stated differently, abductive inference is used to complete the training data with hypotheses about missing or inconsistent data that explain the example or training data using the background theory. This process gives alternative possibilities for assimilating and generalizing this data. In another integration of ALP and ILP, ILP is extended to learn ALP theories from incomplete background data [62]. This allows the framework to perform Multiple Predicate Learning in a natural way.

As we have seen in previous sections several approaches to ALP have recognized the importance of linking this together with Constraint Logic Programming. They have shown that the integration of constraint solving in abductive logic programming enhances the practical utility of ALP. Experiments indicate that the use of constraint solving techniques in abductive reasoning make the abductive computation much more efficient. On the other hand, the integrated paradigm of ALP and CLP can be seen as a high-level constraint programming environment that allows more modular and flexible representations of the problem domain. The potential benefits of this paradigm are largely unexplored at the moment.

7 Challenges and Prospects for ALP

In the past decade, many studies have shown that extending Logic Programming with abduction has many important applications in the context of AI and declarative problem solving. Yet, at this moment the field of ALP faces a number of challenges at the logical, methodological and computational level. In this section we attempt to chart out some of these challenges and point to some promising directions.

7.1 Heterogeneity of ALP

As can be seen in section 4, ALP is a very heterogeneous field. On the one hand, this heterogeneity stems from the fact that logic programming itself shows a complex landscape. On the other hand, it stems from the fact the term *abduction* is defined very broadly and covers a broad class of rather loosely connected reasoning phenomena.

At the conceptual level, abduction is sometimes used to denote concepts at different conceptual levels. For example, in many of the frameworks discussed earlier, abduction is a concept at the *inferential level*: it is a form of logical inference. In other contexts such as in the *abductive semantics* for negation as failure [29], abduction is a concept used at the *semantical level*, as a specific way of formalizing model semantics. This mismatch between different conceptual levels is confusing and a potential hazard for the field.

At the logical level, there are many different formalisms and different semantics. Various forms of abduction have been introduced in different formalisms including pure logic programming, answer set programming and recently a conditional logic programming formalism [30]. The advantage of this is that the field may act as a *forum* for integrating and relating a wide variety of different forms of logical reasoning in otherwise distant areas. A disadvantage is that this heterogeneity may hide a lack of coherence in which efforts of researchers to build effective systems are scattered in a wide variety of incompatible views and approaches. To develop a computational logic, a focused effort at different levels is needed: research on semantics to clarify the declarative meaning, research on knowledge representation to clarify the applications of the logic, research to explore the relation with other logics, and research to investigate how to implement efficient problem solvers. These efforts should link together in a constructive and cross supporting way.

7.2 Epistemological Foundations of ALP

One of the underlying problems of the field is the lack of understanding of the *epistemological foundations* of ALP. Epistemological questions are what kind of knowledge can be represented by an abductive logic framework and vice versa, what does an ALP theory tell us about the problem domain or equivalently, what information about the domain of discourse is expressed by a given ALP theory? Such questions are fundamental to the understanding of any logic. A

clear answer is a prerequisite for developing a well-motivated methodology for declarative problem solving using ALP.

The standard definition of ALP as presented in section 3 does not attempt to answer the above questions. The definition 1 of an abductive solution defines a formal *correctness criterion* for abductive reasoning, but does not address the question of how the ALP formalism should be interpreted. Also the (generic) definition 2 of the formal model semantics of ALP does not provide answers. In fact, here ALP inherits the ambiguity of logic programming at the epistemological level, as demonstrated recently in [15]. Here are some fundamental questions:

- To understand the meaning of an ALP framework, at the very least we need to understand the meaning of its symbols. How is negation in ALP to be understood? The extended completion semantics defined for ALP by Console, Thorasso and Theseider Dupré [10] maps negation as failure literals to classical negation. On the other hand, in the generalized stable semantics [52] and in the ABDUAL framework [1], negation as failure literals are interpreted as modal literals $\neg Kp$ in autoepistemic logic or default logic [38].
- What is the relationship between ALP and classical logic? An ALP framework may contain an arbitrary classical logic theory IC of constraints; in ALP's model semantics, models of an ALP framework satisfy the constraints in IC in the standard way of classical logic. This suggests that ALP is an extension of classical logic. On the other hand, ALP is defined as a study of abductive reasoning while classical logic is normally viewed as the study of deductive reasoning. How are these two views reconciled?

The lack of clear epistemological foundations for ALP is one of the causes of ALP's lack of coherence and is a factor blurring the role and status of ALP at the knowledge representation level in the broader context of logic-based AI. An epistemological study of ALP can contribute significantly to the understanding of the field at the logical and methodological level.

7.3 Computational Challenges

The computational challenges of the paradigm are considerable. The challenge of building abductive systems for solving a broad class of problems formalized by high-level declarative representations, is extremely difficult to realise.

At the theoretical level of complexity, formal results show that in general the problem of computing abduction is hard [26]. In the datalog case, the problem of computing abductive solutions is in general intractable. In the general case of ALP frameworks with function symbols, the existence of an abductive solution is undecidable. On the implementational level, the problem of implementing abductive reasoning can be seen as an extension of the implementation of CLP systems in which we need to reason about constraints of general first order logic.

Current systems such as ACLP, SLDNFAC and IFF are based on the integration of CLP techniques in high level abductive procedures. These systems

operate by reducing the high level constraints in a, in general, nondeterministic process to a constraint store that can be handled efficiently by specialised constraint systems. Recent experiments with the ACLP and SLDNFAC systems have shown that in those cases where the reduction process is deterministic, these procedures can be very performant. However, when the process is nondeterministic, these procedures can start to trash. The reason for this behaviour is that a number of techniques are built in in the current procedures that delay the creation of choice points and perform deterministic computation first. In many applications such as scheduling, these techniques can avoid making choices altogether. In other cases, such as in planning applications, the arsenal of techniques does not suffice to manage these choice points and the current procedures often make *uninformed selections* of choices leading to uncontrolled depth first execution and *trashing*.

The above analysis suggests different ways to improve the computational techniques of ALP. One way is to further improve the techniques to discover deterministic subgoals and delay creation of choice points. A second way is to incorporate techniques for smarter and *better informed* selection of the choice points and choice of alternatives in the choice point. A third way is an improved control to avoid unrestricted depth first reasoning using techniques similar to loop detection and iterative deepening can be used. With respect to the first two problems, different approaches can be followed. One is to further refine the current integration of Constraint Solving in the abductive inference. In the current systems, the CLP solver is a black box that interacts with the abductive solver by returning a solution at the end, or by reporting consistency or inconsistency of the constraint store at different points during the execution. One direction to be examined is how to exploit the information present in the constraint store to steer the search for an abductive solution and make a better informed selection of goals. An alternative direction is to apply techniques from heuristic search in Artificial Intelligence.

An interesting application domain to study the above techniques for abductive reasoning is AI-planning, due to the strong links between abduction and planning and the fact that recently, techniques from constraint solving and heuristic search have been successfully applied in this domain. What we can learn here is how recent developments of constraint and heuristic methods of search in planning could be applied to the more general case of abductive computation.

A complementary approach to address the computational hardness of ALP would be to develop ALP systems in which the user has the facility to incrementally refine her/his model of the problem in a modular way. Starting from a purely declarative problem description, it should be possible to refine the model by adding more and more additional knowledge about the problem, including non-declarative heuristic and operational control knowledge. Again recent work suggests that this is a promising line of development but there is no systematic study of how such a modeling environment would be designed and build in ALP.

A completely different approach is to exploit the kind of techniques used in bottom up abduction [46] (see section 4) based on the computation of stable

models of a ground logic program. Techniques like those used by the smodels system [75] which integrates methods from propositional constraint propagation with bottom up application of semantic fixpoint operators of the 3-valued completion semantics and well-founded semantics. In the current state of the art, it seems that while the latter techniques based on reasoning on propositional theories are more robust, the abductive extensions of SLDNF with CLP may outperform the first ones especially as they can take into account more easily additional problem domain specific information. Therefore, extending the latter procedures along the lines suggested above is a promising research direction.

7.4 Challenges at the Application Level

In the past decade, the potential of the different ALP frameworks have been demonstrated in a wide variety of application domains. However, only a few of the current running applications exceed the level of academic toy examples. Like in many other areas of AI, this potential has not yet been realized in realistic and industrial scale applications. One of the challenges of the domain is to find interesting niche domains with industrial impact in which the current systems can be evaluated and fine-tuned. Experimentation with and evaluation of abductive systems in realistic domains could yield important information at the levels of language constructs, methodology, computational control, integration of heuristic information, etc..

Some prototypical classes of problems that seem good candidates for fine-tuning ALP methods are Scheduling and Planning domains and Knowledge Intensive Learning where machine learning with a rich background knowledge can be performed only if the inductive methods are integrated with abduction [123,73,35].

7.5 A Possible Approach to These Challenges

In this section, we briefly describe our own views on how to approach the above logical and computational challenges.

The underlying logic for ALP is ID-logic [16,23] a logic which is appropriate for ALP in the way that it extends classical logic with inductive definitions of a generalized non-monotone kind. As mentioned earlier in section 3, an abductive logic framework (P, A, IC) has a natural embedding in ID-logic. P represents a definition of the non-abducible predicates while IC represents a set of classical logic assertions. In this view, ALP is the study of abduction in the context of ID-logic. ID-logic was defined in an attempt to cope with the epistemological challenges of logic programming and gives answers to the epistemological questions raised in section 7.2.

At the computational level, we are currently developing a system called the A-system [63,118] integrating features of ACLP and SLDNFAC with special attention to the search in the abductive computation. During the computation, the selection and evaluation of choice points is guided by information obtained

from a constraint store associated to the abductive solution. With this information the high level search can avoid deadend branches before entering them. The result is a more robust and modular system which is capable to solve effectively a wider range of problems than the older systems. The application domain of the experiments with the A-system are currently focused on scheduling and planning applications. The A-system is built on top of Sicstus Prolog (version 3.8.5 or above) and is available at *http://www.cs.kuleuven.ac.be/∼dtai/kt/*.

8 Conclusion

Abductive logic programming grew out of attempts to use logic programming techniques for a broad class of problems from AI and other areas of Computer Science. At present Abductive Logic Programming presents itself as a "conservative extension" of Logic Programming that allows more declarative representations of problems. The main emphasis till now has been on setting up different frameworks for abduction and showing how they provide a general approach to declarative problem solving.

ALP faces a number of challenges, at the logical, methodological and computational level typical for a field in an initial stage of development. We are now beginning to understand the contributions of this field and to develop solutions for the problems that the field faces.

At the logical level, ALP aims to be suitable for declarative knowledge representation, thus facilitating maintenance, reusability and graceful modifiability. Yet, ALP retains from logic programming the possibility of embedding high level strategic information in an abductive program which allows us to speed up and fine tune the computation. In this respect, ALP is able to combine the advantages of declarative specification and programming to a greater extent than standard logic programming.

The field has also started to recognize the full extent of the problem and the complexity of developing effective and useable ALP systems. The overall task of ALP of providing a high-level general purpose modeling environment which at the same time is computationally effective is an extremely difficult one. But we are beginning to learn how to analyze and break this task down to appropriate subproblems that are amenable to study within our current understanding of the field. The hope remains that within the high-level programming environment that ALP could provide, the programmer will be able to solve problems effectively in a translucent way.

Acknowledgements This work was partly supported by the European Union KIT project CLSFA (9621109).

References

1. J. J. Alferes, L. M. Pereira, T. Swift. Well-founded Abduction via Tabled Dual Programs. In Procs. of the 16th International Conference on Logic Programming, Las Cruces, New Mexico, Nov. 29 - Dec. 4, 1999.
2. H. Ade and M. Denecker. Abductive inductive logic programming. In C.S. Mellish, editor, *Proc. of the International Joint Conference on Artificial Intelligence*, pages 1201–1209. Morgan Kaufman, 1995.
3. J. J. Alferes, J. A. Leite, L. M. Pereira, P.Quaresma. Planning as Abductive Updating. In D. Kitchin (ed.), Procs. of AISB'00, 2000.
4. Balsa, J., Dahl, V. and Pereira Lopes, J.G. Datalog Grammars for Abductive Syntactic Error Diagnosis and Repair. In Proc. Natural Language Understanding and Logic Programming Workshop, Lisbon, 1995.
5. M. Bruynooghe, H. Vandecasteele, D.A. de Waal, and Denecker M. Detecting unsolvable queries for definite logic programs. *The Journal of Functional and Logic Programming*, 1999:1–35, November 1999.
6. J.F.Castro, L. M. Pereira, Z.Vale. Power Grid Failure Diagnosis Certification. Technical Report, University of Lisbon.
7. A. Ciampolini, E. Lamma, P. Mello and P. Torroni. Expressing Collaboration and Competition Among Abductive Logic Agents. In *AI*IA Notizie* - Anno XIII(3), Settembre 2000, pag. 19–24.
8. E. Charniak and D. McDermott. *Introduction to Artifical Intelligence.* Addison-Wesley, 1985.
9. K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.
10. L. Console, D. Theseider Dupré, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.
11. L. Console, L. Portinale and Theseider Dupré, D., Using Compiled knowledge to guide and focus abductive diagnosis. IEEE Transactions on Knowledge and Data Engineering, Vol. 8 (5), pp. 690-706, 1996.
12. L. Console, M.L. Sapino and Theseider Dupré, D. The Role of Abduction in Database View Updating. Journal of Intelligent Information Systems, Vol. 4(3), pp. 261-280, 1995.
13. D. de Waal, M. Denecker, M. Bruynooghe, and M. Thielscher. The generation of pre-interpretations for detecting unsolvable planning problems. In *Proceedings of the Workshop on Model-Based Automated Reasoning (15th International Joint Conference on Artificial Intelligence)*, pages 103–112, 1997.
14. M. Denecker. A Terminological Interpretation of (Abductive) Logic Programming. In V.W. Marek, A. Nerode, and M. Truszczynski, editors, *International Conference on Logic Programming and Nonmonotonic Reasoning*, Lecture notes in Artificial Intelligence 928, pages 15–29. Springer, 1995.
15. M. Denecker. On the Epistemological foundations of Logic Programming and its Extensions. In *AAAI Spring Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning*, volume technical report SS-01-01. American Association for Artificial Intelligence, AAAI Press, 2001.
16. M. Denecker. Extending classical logic with inductive definitions. In J.Lloyd et al., editor, *First International Conference on Computational Logic (CL2000)*, volume 1861 of *Lecture notes in Artificial Intelligence*, pages 703–717, London, July 2000. Springer.

17. M. Denecker and D. De Schreye. SLDNFA; an abductive procedure for normal abductive programs. In K.R. Apt, editor, *Proc. of the International Joint Conference and Symposium on Logic Programming*, pages 686–700. MIT Press, 1992.
18. M. Denecker and D. De Schreye. Representing incomplete knowledge in abductive logic programming. In *Proc. of the International Symposium on Logic Programming*, pages 147–163. MIT Press, 1993.
19. M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. *Journal of Logic and Computation*, 5(5):553–578, September 1995.
20. M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.
21. M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *Proc. of the European Conference on Artificial Intelligence*. John Wiley and sons, 1992.
22. M. Denecker, K. Van Belleghem, G. Duchatelet, F. Piessens, and D. De Schreye. Using Event Calculus for Protocol Specification. An Experiment. In M. Maher, editor, *The International Joint Conference and Symposium on Logic Programming*, pages 170–184. MIT Press, 1996.
23. Marc Denecker, Maurice Bruynooghe, and Victor W Marek. Logic programming revisited : logic programs as inductive definitions. *ACM Transactions on Computational Logic*, 2001. accepted.
24. Theseider Dupré, D.. Characterizing and Mechanizing Abductive Reasoning. PhD Thesis, Dip. Informatica, Università di Torino, 1994.
25. P.M. Dung. Negations as hypotheses: an abductive foundation for Logic Programming. In *Proc. of the International Conference on Logic Programming*, 1991.
26. Thomas Eiter, Georg Gottlob, Nicola Leone. Abduction from Logic Programs: Semantics and Complexity. Theoretical Computer Science 189(1-2):129-177 (1997).
27. Thomas Eiter, Wolfgang Faber, Nicola Leone, and Gerald Pfeifer. Declarative problem-solving using the dlv system. In Jack Minker, editor, Logic-Based Artificial Intelligence. Kluwer Academic Publishers, 2000.
28. K. Eshghi. Abductive planning with Event Calculus. In R.A. Kowalski and K.A. Bowen, editors, *Proc. of the International Conference on Logic Programming*. The MIT press, 1988.
29. K. Eshghi and R.A. Kowalski. Abduction compared with negation as failure. In *Proc. of the International Conference on Logic Programming*. MIT-press, 1989.
30. D. Gabbay, L. Giordano, A. Martelli, and M.L. Sapino. Conditional reasoning in logic programming. *Journal of Logic Programming*, 44(1-3):37–74, 2000.
31. J. Gartner, T. Swift, A. Tien, C. V. Damásio, L. M. Pereira. Psychiatric Diagnosis from the Viewpoint of Computational Logic. In G. Wiggins (ed.), Procs. of AISB, 2000.
32. Gelfond, M., Lifschitz, V. Classical negation in logic programs and disjunctive databases. New Generation Computing, pp. 365-387, 1991.
33. Fages, F. A New Fixpoint Semantics for General Logic Programs Compared with the Well-Founded and the Stable Model Semantics. Proc. of ICLP'90, pp. 442 – 458, 1990.
34. P. Flach and A. C. Kakas (Eds.). Abduction and Induction: Essays on their Relation and Integration. Kluwer Academic Press, 2000.
35. P. Flach and A. C. Kakas. Abductive and Inductive Reasoning: Background and Issues. In Peter Flach and Antonis Kakas, editors, *Abduction and Induction: essays on their relation and integration*. Kluwer, 2000.

36. Fung, T.H. Abduction by deduction. Ph.D. Thesis, Imperial College, London, 1996.
37. T.H. Fung, R.A. Kowalski. The iff procedure for abductive logic programming. In *Journal of Logic Programming* 33(2):151–165, Elsevier, 1997.
38. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. of the International Joint Conference and Symposium on Logic Programming*, pages 1070–1080. IEEE, 1988.
39. Hobbs, J.R. An integrated abductive framework for discourse interpretation. Symposium on Automated Abduction,Stanford, 1990.
40. K. Inoue. Hypothetical reasoning in Logic Programs. *Journal of Logic Programming*, 18(3):191–228, 1994.
41. K. Inoue. A simple characterization of extended abduction. In: *Proceedings of the First International Conference on Computational Logic, Lecture Notes in Artificial Intelligence*, 1861, pages 718–732, Springer, 2000.
42. K. Inoue, Y. Ohta, and R. Hasegawa. Bottom-up Abduction by Model Generation. Technical Report TR-816, Institute for New Generation Computer Technology, Japan, 1993.
43. K.Inoue and C. Sakama. Abductive framework for nonmonotonic theory change. In: *Proceedings of IJCAI-95*, pages 204–210, Morgan Kaufmann, 1995.
44. K. Inoue and C. Sakama. Abducing priorities to derive intended conclusions. In: *Proceedings of IJCAI-99*, pages 44–49, Morgan Kaufmann, 1999.
45. K. Inoue and C. Sakama. Computing extended abduction through transaction programs. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):339–367, 1999.
46. Iwayama, N. and Satoh, K. Computing Abduction by Using TMS with Top-Down Expectation. Journal of Logic Programming, Vol. 44 No.1-3, pp. 179 – 206, 2000.
47. J.R. Josephson and S.G. Josephson, editors. *Abductive Inference: Computation, Philosophy, Technology.* New York: Cambridge University Press, 1994.
48. C.G. Jung, K. Fischer, and A. Burt. Multi-agent planning using an abductive event calculus. Technical Report DFKI Report RR-96-04, DFKI, Germany, 1996.
49. A. C. Kakas. ACLP: Integrating Abduction and Constraint Solving. In Proceedings of NMR2000, 2000.
50. Kakas, A. C., Kowalski, R. A., Toni, F., Abductive logic programming. *Journal of Logic and Computation* 2(6) (1993) 719–770
51. 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 5, pages 235-324, D.M. Gabbay, C.J. Hogger and J.A. Robinson eds., Oxford University Press (1998)
52. A.C. Kakas and P. Mancarella. Generalised Stable Models: a Semantics for Abduction. In Proc. 9th European Conference on AI, ECAI90, Stockolm, 1990.
53. A.C. Kakas and P. Mancarella. Database updates through abduction. In *Proc. of the 16th Very large Database Conference*, pages 650–661. Morgan Kaufmann, 1990.
54. A.C. Kakas and P. Mancarella. On the relation of truth maintenance and abduction. In Proc. 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI90, Nagoya, Japan, 1990.
55. A. C. Kakas and P. Mancarella. *Knowledge assimilation and abduction.* International Workshop on Truth Maintenance, Stockholm, ECAI90, Springer Verlag Lecture notes in Computer Science, Vol. 515, pp. 54-71, 1990.
56. Kakas, A. C., Michael, A. Integrating abductive and constraint logic programming. In *Proc. International Logic Programming Conference*, pp. 399-413, 1995.

57. A.C. Kakas and A. Michael. Air-Crew Scheduling through Abduction. Proceedings of IEA/AIE-99, pp. 600–612, 1999.
58. A.C. Kakas and A. Michael. An Abductive-based scheduler for air-crew assingment. Journal of Applied Artificial Intelligence Vol. 15, pp. 333–360, 2001.
59. A.C. Kakas, A. Michael and C. Mourlas. ACLP: a case for non-monotonic reasoning. in Proceedings of NMR98,pp. 46–56, 1998.
60. A.C. Kakas, A. Michael and C. Mourlas. ACLP: Abductive Constraint Logic Programming. Journal of Logic Programming (special issue on Abductive Logic Programming), Vol. 44 (1-3), pp. 129-177, 2000.
61. A.C. Kakas and C. Mourlas. ACLP: Flexible Solutions to Complex Problems. Proceedings of Logic Programming and Non-monotonic Reasoning, LPNMR97, 1997,
62. A.C. Kakas and F. Riguzzi. Abductive Concept Learning. New Generation Computing, Vol. 18, pp. 243-294, 2000.
63. A.C Kakas, Bert Van Nuffelen, and Marc Denecker. A-system : Problem solving through abduction. In *Proceedings of IJCAI'01 - Seventeenth International Joint Conference on Artificial Intelligence*, pp. 591-597, 2001.
64. R.A. Kowalski and F. Sadri, From Logic Programming towards Multi-agent Systems. *Annals of Mathemathics and Artificial Intelligence*, Vol 25, pp. 391-419, 1999.
65. R.A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(4):319–340, 1986.
66. Kowalski, R.A.; Toni, F.; Wetzel, G.; 1998. Executing suspended logic programs. *Fundamenta Informaticae* 34(3):203–224, ISO Press.
67. J. A. Leite, F. C. Pereira, A. Cardoso, L. M.Pereira. Metaphorical Mapping Consistency via Dynamic Logic Programming. In J. Lloyd et al. (eds.), Procs. of First Int. Conf. on Computational Logic (CL 2000), London, UK, pages 1362-1376, LNAI 1861, Springer, 2000.
68. J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
69. V.W. Marek and M. Truszczyński. Stable models and an alternative logic programming paradigm. In K.R. Apt, V. Marek, M. Truszczynski, and D.S. Warren, editors, *The Logic Programming Paradigm: a 25 Years Perspective*, pages pp. 375–398. Springer-Verlag, 1999.
70. J. McCarthy. Situations, actions and causal laws. Technical Report AI-memo 1, Artifical Intelligence Program, Standford University, 1957.
71. Lode R. Missiaen, Marc Denecker, and Maurice Bruynooghe. CHICA, an abductive planning system based on event calculus. *Journal of Logic and Computation*, 5(5):579–602, September 1995.
72. L.R. Missiaen, M. Bruynooghe, and M. Denecker. Abductive planning with event calculus. Internal report, Department of Computer Science, K.U.Leuven, 1992.
73. S. Muggleton. Theory Completion in Learning. In Proceedings of Inductive Logic Programming, ILP00, 2000.
74. I. Niemela and P. Simons. Smodels - an implementation of the stable model and well-founded semantics for normal logic programs. Proceeings of the 4th International Conference on Logic Programming and Non-monotonic Reasoning, pp. 420-429, 1997.
75. I. Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):241–273, 1999.
76. M. Pagnucco. The role of abductive reasoning within the process of belief revision. PhD Thesis, Department of Computer Science, University of Sydney, 1996.

77. C.S. Peirce. *Philosophical Writings of Peirce*. Dover Publications, New York, 1955.
78. Nikolay Pelov, Emmanuel De Mot, and Marc Denecker. Logic programming approaches for representing and solving constraint satisfaction problems : a comparison. In *Proceedings of LPAR'2000 - 7th International Conference on Logic for Programming and Automated Reasoning*, 2000. accepted.
79. L.M. Pereira, J.N. Aparício, and J.J. Alferes. Nonmonotonic reasoning with Well-Founded Semantics. In K. Furukawa, editor, *Proc. of the eight international conference on logic programming*, pages 475–489. the MIT press, 1991.
80. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.
81. D. Poole. A Logical Framework for Default Reasoning. *Artifical Intelligence*, 36:27–47, 1988.
82. D. Poole. A methodology for using a default and abductive reasoning system. *International Journal of Intelligent Systems*, 5(5):521–548, December 1990.
83. D. Poole. Logic programming, abduction and probability: A top-down anytime algorithm for computing prior and posterior probabilities. *New Generation Computing*, 11(3–4):377–400, 1993.
84. D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.
85. D. Poole. Probabilistic conflicts in a search algorithm for estimating posterior probabilities in Bayesian networks. *Artificial Intelligence*, 88:69–100, 1996.
86. D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94:7–56, 1997. special issue on economic principles of multi-agent systems.
87. D. Poole. Probabilistic partial evaluation: Exploiting rule structure in probabilistic inference. In *Proc. 15th International Joint Conf. on Artificial Intelligence (IJCAI-97)*, pages 1284–1291, Nagoya, Japan, 1997.
88. D. Poole, R. Goebel, and R. Aleliunas. Theorist: A logical reasoning system for defaults and diagnosis. In N. Cercone and G. McCalla, editors, *The Knowledge Frontier: Essays in the Representation of Knowledge*, pages 331–352. Springer-Verlag, New York, NY, 1987.
89. David Poole. Learning, bayesian probability, graphical models, and abduction. In Peter Flach and Antonis Kakas, editors, *Abduction and Induction: essays on their relation and integration*. Kluwer, 2000.
90. Poole, D., Goebel, R.G., Aleliunas, Theorist: a logical reasoning system for default and diagnosis. *The Knowledge Fronteer: Essays in the Representation of Knowledge*, Cercone and McCalla eds, Springer Verlag Lecture Notes in Computer Science 331–352, 1987.
91. D. Poole, A. Mackworth, R. G. Goebel, Computational Intelligence: a logical approach Oxford University Press, 1998.
92. Stathis Psillos. Ampliative Reasoning: Induction or Abduction. In *ECAI96 workshop on Abductive and Inductive Reasoning*, 1996.
93. Rochefort, S., Tarau, P. and Dahl, V. Feature Interaction Resolution Through Hypothetical Reasoning. In Proc. 4th World Multiconference on Systemics, Cybernetics and Informatics (SCI2000) and the 6th International Conference on Information Systems Analysis and Synthesis (ISAS2000), Orlando, USA July 23-26, 2000.
94. Saccà, D., Zaniolo, C. Stable Models and Non-Determinism in Logic Programs with Negation. Proc. of PODS'90, pp. 205 – 217, 1990.

95. F.Sadri, F. Toni, I.Xanthakos. A Logic-Agent based System for Semantic Integration. 17th International CODATA Conference- Data and Information for the Coming Knowledge Millennium- CODATA 2000, Theme I-3, Integration of Heterogeneous Databases and Data Warehousing.
96. F. Sadri, F. Toni. Abduction with negation as failure for active databases and agents. *Proc. AI*IA 99, 6th Congress of the Italian Association for Artificial Intelligence*, pages 353–362, Pitagora Editrice Bologna, 1999.
97. F. Sadri, F. Toni. Abduction with Negation as Failure for Active and Reactive Rules, In E. Lamma and P. Mello eds., Proc. AI*IA 99, 6th Congress of the Italian Association for Artificial Intelligence, Springer Verlag LNAI 1792, pages 49-60, 2000.
98. C. Sakama and K. Inoue. Representing priorities in logic programs. In: *Proceedings of the 1996 Joint International Conference and Symposium on Logic Programming*, pages 82–96, MIT Press, 1996.
99. C. Sakama and K. Inoue. Updating extended logic programs through abduction. In: *Proceedings of LPNMR '99, Lecture Notes in Artificial Intelligence*, 1730, pages 147–161, Springer, 1999.
100. C. Sakama and K. Inoue. Abductive logic programming and disjunctive logic programming: their relationship and transferability. *Journal of Logic Programming - Special issue on ALP* 44(1-3):71–96, 2000.
101. C. Sakama and K. Inoue. An alternative approach to the semantics of disjunctive logic programs and deductive databases. *Journal of Automated Reasoning* 13(1):145–172, 1994.
102. F. Sadri, F. Toni and P. Torroni. Dialogues for negotiation: agent varieties and dialogue sequences. In Pre-proc. ATAL'01, special track on negotiation. Seattle, WA, August 2001.
103. Satoh, K. Statutory Interpretation by Case-based Reasoning through Abductive Logic Programming. Journal of Advanced Computational Intelligence, Vol. 1, No.2, pp. 94 – 103, 1997.
104. Satoh, K. Computing Minimal Revised Logic Program by Abduction. Proc. of the International Workshop on the Principles of Software Evolution, IWPSE98, pp. 177 – 182, 1998.
105. Satoh, K. Using Two Level Abduction to Decide Similarity of Cases. Proc. of ECAI'98 pp. 398 – 402, 1998.
106. K. Satoh and N. Iwayama. A Query Evaluation method for Abductive Logic Programming. In K.R. Apt, editor, *Proc. of the International Joint Conference and Symposium on Logic Programming*, 1992.
107. Satoh, K. and Iwayama, N. Computing Abduction by Using the TMS. Proc. of ICLP'91, pp. 505 – 518, 1991.
108. Satoh, K. and Iwayama, N. A Query Evaluation Method for Abductive Logic Programming. Proc. of JICSLP'92, pp. 671 – 685, 1992.
109. Satoh, K. and Iwayama, N., "A Correct Goal-Directed Proof Procedure for a General Logic Program with Integrity Constraints", E. Lamma and P. Mello (eds.), Extensions of Logic Programming, LNAI 660, pp. 24 – 44, Springer-Verlag (1993).
110. M. Shanahan. Prediction is deduction but explanation is abduction. In *Proc. of the IJCAI89*, page 1055, 1989.
111. F. Toni. A semantics for the Kakas-Mancarella procedure for abductive logic programming. *Proc. GULP'95*, M. Alpuente and M. I. Sessa, eds., pages 231-242, 1995.

112. F. Toni. Automated Reasoning for Collective Information Management. *Proc. LocalNets, International Workshop on Community-based Interactive Systems*, in conjunction with AC'99, the Annual Conference of the EC I^3 Programme
113. K. Van Belleghem, M. Denecker, and D. De Schreye. Representing continuous change in the abductive event calculus. In *Proc. of the International Conference on Logic Programming*. MIT-Press, 1994.
114. K. Van Belleghem, M. Denecker, and D. De Schreye. The abductive event calculus as a general framework for temporal databases. In *Proc. of the International Conference on Temporal Logic*, pages 301–316, 1994.
115. K. Van Belleghem, M. Denecker, and D. De Schreye. A strong correspondence between description logics and open logic programming. In Lee Naish, editor, *Proc. of the International Conference on Logic Programming, 1997*, pages 346–360. MIT-press, 1997.
116. A. Van Gelder, K.A. Ross, and J.S. Schlipf.r The Well-Founded Semantics for General Logic Programs. *Journal of the ACM*, 38(3):620–650, 1991.
117. Bert Van Nuffelen and Marc Denecker. Problem solving in ID-logic with aggregates: some experiments. In M. Denecker, A. Kakas, and F. Toni, editors, *8th Int. Workshop on Non-Monotonic Reasoning (NMR2000), session on Abduction*, pages 1–15, Breckenridge, Colorado, USA, April 9-11 2000.
118. B. Van Nuffelen and A.C Kakas, A-System: Programming with Abduction. In *Proceedings of LPNMR2001*, LNAI Vol. 2173, pp. 393-396, Springer Verlag, 2001.
119. Sven Verdoolaege, Marc Denecker, and Frank Van Eynde. Abductive reasoning with temporal information. In Ielka van der Sluis Harry Bunt and Elias Thijsse, editors, *Proceedings of the Fourth International Workshop on Computational Semantics*, pages 351–366, 2001.
120. Wetzel, G. *Abductive and Constraint Logic Programming*. Ph.D. Thesis, Imperial College, London, 1997.
121. G. Wetzel, F. Toni. Semantic Query Optimization through Abduction and Constraint Handling. Proc. of the International Conference on Flexible Query Answering Systems, T. Andreasen, H. L. Larsen and H. Christiansen eds., Springer Verlag LNAI 1495 (1998).
122. The XSB Group. The XSB logic programming system, version 2.0. 1999. Available from `http://www.cs.sunysb.edu/~sbprolog`.
123. A. Yamamoto. Using abduction for induction based on bottom up generalization. In Peter Flach and Antonis Kakas, editors, *Abduction and Induction: essays on their relation and integration*. Kluwer, pp. 267-280,2000.
124. J.H. You, L.Y. Yuan and R. Goebel. An abductive approach to disjunctive logic programming. Journal of Logic Programming (special issue on Abductive Logic Programming), Vol. 44 (1-3), pp. 101-128, 2000.