Abduction in Logic Programming

Antonis Kakas
Department of Computer Science, University of Cyprus,
75 Kallipoleos St., Nicosia, Cyprus.
email: antonis@ucy.ac.cy
Web address: http://www.cs.ucy.ac.cy/~antonis

Marc Denecker
Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium.
email: marcd@cs.kuleuven.ac.be
Web address: http://www.cs.kuleuven.ac.be/~marcd

March 13, 2001

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 nonmonotonic 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 [6, 79], natural language understanding [6, 33, 3, 81], default reasoning [70, 24, 20, 43], planning [23, 97, 61, 51], knowledge assimilation and belief revision [47, 65], multi-agent coordination [5, 55] 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 [23] introduced abduction in logic programming in order to solve planning problems in the Event Calculus [56]. 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 [97], Missiaen et al. [62, 61], Denecker [16], Jung [41] and recently in [51, 52]. Kakas and Mancarella showed the application of abduction in logic programming for deductive database updating and knowledge assimilation [46, 48]. The application of abduction to diagnosis has been studied in [8, 9] 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 [24] and later Kakas and Mancarella in [45, 47] and Dung in [20], used abduction as a semantical device to describe the non-monotonic semantics of Logic Programming (in a way analogous to Poole in [70]). In [13, 11], 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) [43, 44] was recognised 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 [47], the abductive procedure of [24] for computing negation as failure through abduction was extended to the case of general abductive predicates and later in [49, 53, 52] this was extended to incorporate constraint solving of CLP in a new framework called ACLP that integrates features of abductive and constraint logic programming. Another early abductive procedure was developed in [8] using the completion. [12] proposed SLDNFA, an extension of SLDNF with abduction allowing non-ground abductive hypotheses. In [31] an abductive procedure that can be regarded as a hybrid between SLDNFA and the procedure of Console et al has been defined based on explicit rewrite rules with the completion and equality. This has later [103] 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 [92] and [36]; the latter system was an implementation using the Model Generator MGTP developed on the multiprocessor machine developed at ICOT. Another recent abductive procedure in LPis 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, method-

[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.

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

The rest of the paper is organised 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 analysing 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?

Abduction emerged in the context of philosophical studies in logic. There are many views on what is abduction. The aim of this section is to give our personal view as we believe is emerging from recent studies in the field of Artificial Intelligence and Logic Programming.

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 [66]. Often abduction has been defined broadly as any form of "inference to the best explanation" [40] where *best* refers to the fact that the generated hypothesis is subjected to extra quality conditions such as (a form of) minimality or some economic criterion. This definition is extremely general and covers forms of hypothetical reasoning in a wide range of different settings, from human scientific discovery in philosophical treatments of human cognition to formally defined reasoning principles in formal and computational logic.

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] $T \models \mathcal{E} \rightarrow Q$

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

[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: $T \models \forall(\mathcal{E} \rightarrow Q)$.

an abductive explanation of an observation as a formula which *logically entails* the observation. However, some have argued, with good reasons, that it is more natural to view an explanation as a *cause* for the observation [40]. A well-known example is as follows [80]: 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 directionalities 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 a form of hypothetical reasoning, abduction 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 incomplete knowledge, deduction is the reasoning paradigm to determine whether a statement is true in all possible states of affairs; abduction returns an explanation formula corresponding to a collection of possible states of affairs in which the observation would be true or would be caused. 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.

Many philosophers and logicians have argued that abduction is a generalisation of *induction* [28]. Indeed, induction can also be seen as a form of *inference to the best explanation.* But the philosophical definition of abduction is so general and abstract that it covers many forms of inference that could otherwise be formally distinguished. In the context of formal logic and its relation to problem solving, the term *abduction* has been used to denote a form of reasoning that can be distinguished from *inductive reasoning.* In most current applications of

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

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 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 my car does not start this morning is the explanation that the 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. This distinction in the form of the answer is one of the reasons that induce strong differences in the underlying inference procedures.

2.3 Abduction and Declarative Knowledge Representation

In the early days of logic-based AI, *deduction* was considered as the fundamental problem solving paradigm of declarative logic [60]. Standard logic programming belongs to this paradigm and is based on deductive query solving. However, it can be shown easily that in many applications, abduction is a more natural option for *declarative problem solving*. The argument is as follows.

The process of declarative knowledge representation starts with the design of an alphabet. This alphabet defines the *ontology* of the domain of discourse and is designed to represent the objects, concepts and relationships in the problem domain. In the next phase, the human expert can begin with the *knowledge description* phase in which he or she expresses his or her knowledge by formal statements.

It is obvious that the choice of the alphabet is a determining factor for the quality of a specification. 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. For example, to express that "persons are men or women", it is natural to choose the alphabet $person/1, man/1, woman/1$. The rule can be expressed by:

$$\forall x.person(x) \rightarrow man(x) \lor woman(x)$$

If we had represented the set of persons by a balanced binary tree, this would have complicated significantly the expression of the above knowledge.

Consequently, to obtain a declarative knowledge representation, the choice of the alphabet is governed by criteria of isomorphism between this and the concepts used by the human expert. However, the choice of the alphabet determines also what kind of inference is needed to solve a specific computational problem. Once the alphabet is fixed, any computational problem of searching for a possible extension of certain unknown predicates that satisfies certain conditions is by definition an abductive inference problem or a model generation problem[5]. Such computational problems occur very frequently. For example, assume that we want to solve the problem of time tabling lectures at a university. Three important types of objects in this domain are *lectures*, *time slots* and *class rooms*. Lectures take place in class rooms and at some time slots. The natural choice to represent these relations is by predicates, e.g. $time\_of\_lecture/2$ and $room\_of\_lecture/2$. Now, observe that at this stage, although only the alphabet is determined, we know already that solving the time tabling problem requires the computation of a table of $time\_of\_lecture/2$ and $room\_of\_lecture/2$ atoms that satisfies certain data, constraints and conditions imposed on correct schedules. Computing such tables is an abductive task, not a deductive task.

The example illustrates a general phenomenon in the use of logic for problem solving. The choice of the ontology and the alphabet can fix the sort of inference needed to solve the computational problem. If the alphabet is chosen in accordance with the natural structure of the domain of discourse, very often (but not always), abductive inference or model generation is needed. Vice versa, if one insists on using deductive inference or is forced to do so by lack of an alternative, the choice of the alphabet must be done in function of this choice. For example, the university time tabling problem can be solved by deductive query solving (e.g. in Prolog or CLP) but in this case a different and more complex ontology is used (e.g. a list of atoms or tuples). The use of this more complex ontology reduces the readability and modularity and makes the logic specification problem dependent hence reducing its reusability.

Of course, in many cases, the choice of the representation is not governed merely by issues of natural representation but also and even more by issues of computational effectiveness. Clearly, there is a trade-off here. While the use of a more complex ontology may seriously reduce the elegance of the representation, it may greatly augment the expressivity to encode procedural, heuristic and strategic information on how to solve the computational problem. The aim of the research of intelligent search and inference methods is to push this trade-off 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.

[5]Recall the close relationship between abduction and model generation, as explained in the previous section.

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

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

Definition 3.1 *Given an abductive logic theory (P, A, T), 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* in two different ways. One is that it 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 have been used to define different abductive logic frameworks.

Another way in which the definition is generic is the choice of the syntax. In most applications, 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 [37] or constraint logic programming [49].

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 [70], 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 strictly stronger than the consistency view and is the one taken in most versions of ALP.

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 [45], which suggests the following definition.

Definition 3.2 *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:

$$(P, A, IC \models F \text{ iff for each model } M \text{ of } (P, A, C),\ M \models F.$$

Note that also this definition is generic in the choice of the semantics of logic programming. This way, abductive extensions of stable semantics [45] and of well-founded semantics [68] have been defined. Also the completion semantics has been extended [8] to the case of abductive logic programs. The semantics of 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 Clarks equality theory.
- IC
- $comp(P, A)$, the set of completed definitions for all non-abducible predicates.

As argued in [11], in many applications of ALP, the set P in an abductive logic framework (P, A, IC) represents the human expert's strong *definitional knowledge*, i.e. knowledge which fully determines one or a group of predicates in terms of other *open* predicates. Each rule represents a *case* in which the defined predicate in the head will be true; the program P is an exhaustive enumeration of the cases. The set A is the set of open predicates. The set P of rules defines the defined atoms. The open predicates have no definition. The human expert's weaker *assertional knowledge* is represented by the theory IC of *integrity constraints.* To emphasize this way of interpreting ALP in the context of knowledge representation, Abductive Logic Programming is also sometimes called Open Logic Programming [11].

4 Abductive Logic Programming Frameworks

The framework defined in the previous section is generic in the syntax and semantics. In the past ten years, the framework has been instantiated (and sometimes has been extended) in different ways. The aim of this section is to present briefly a number of these alternative frameworks in an attempt to show the wider variety of motivations and approaches that are found in Abductive Logic Programming. These different instantiations differ from each other by using different formal syntax or semantics, or sometimes simply because they use a different inference method, hence induce a different procedural semantics. Although this variety of approaches was useful at the beginning of the field we will argue in section 6 that the convergence of these approaches is an important problem for the field that needs current attention.

4.1 Approaches under the completion semantics for LP

Abduction through Deduction One of the first ALP frameworks is that of [8]. The syntax in this framework is that of hierarchical logic programs[7] with a predefined set of abducible predicates. The formal syntax

[7]A hierarchical program is one without recursion.

is an extension of Clark's completion semantics [7] 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 nonmonotonic reasoning. In particular, many characterisations 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 the abductive explanations to a query with respect to a set of (non-recursive) rules can be characterised in a deductive way if we apply the completion semantics as a closure principle.

Formally, given is 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 $P_C \models O \leftrightarrow F$. The formula F, called the *explanation formula*, can be seen as a disjunctive characterisation of all abductive solutions of O given P. The restriction to hierarchical programs ensures termination of a procedure to compute the explanation formula.

The above abductive framework has been used to formalize diagnostic problem solving and classification in nonmonotonic inheritance hierarchies [8, 19], and has been extended to characterize updates in deductive databases [10]. The completion semantics is also the basis for the "knowledge compilation" optimization of abductive problem solving described in [9].

SLDNFA and Open Logic Programming SLDNFA [12, 15] is an abductive extension of SLDNF-resolution [59], suitable for abductive reasoning in the context of abductive logic programs under the completion semantics. This procedure came out of the early attempts to implement AI-planning using abductive reasoning in the event calculus. Early experiments with the procedure were in the context of temporal reasoning and planning [16].

In a number of subsequent knowledge representation and reasoning experiments with abductive logic programming and SLDNFA [14, 17], the attention gradually shifted to the logical, semantical and representational aspects of the formalism rather than the abductive reasoning. SLDNFA was shown to be useful not only for abductive problems but also for deductive problems. To stress these aspects, [11] developed Open Logic Programming, an LP knowledge representation framework for representing incomplete knowledge. Like the TBOX in description logics, an open logic program represents a set of definitions, while the constraints represent assertions, like the ABOX in description logics. The link with description logics was further investigated in [99].

Recently, the work on OLP has progressed in two directions. At the computational level, a serious effort was done for improving the computational performance and expressivity of the SLDNFA procedure, for example by integrating it with CLP-solvers and higher order aggregates [101]. On the logical level, a problem with OLP is that it is based on the weak completion semantics which does not correctly deal with positive inductive definitions such as transitive closure. For this reason, OLP has evolved into ID-logic[18], an integration of classical logic with inductive definitions under well-founded semantics.

SLDNFA has been applied in experiments with planning and temporal reasoning [16], scheduling and Constraint Satisfaction[67, 101].

The IFF Framework The IFF framework is also based on the completion semantics. It was initially developed as a unifying framework for integrating abduction and view updates [30, 31]. An extension of it [102, 57] was developed to deal with built-in predicates as in constraint logic programming, aimed at applications such as job-shop scheduling, and semantic query optimisation [57, 103]. Later, a modification of the IFF proof procedure incorporating negation as failure was proposed [84], arising from experimenting with the use of the proof procedure to model the reasoning underlying agents [55] and the specification and management of active rules in databases.

The main underlying LP semantics used in this framework is Fitting's three-valued completion semantics. In this framework it is also possible [57], for if and only if definitions to be defined explicitly by the user. The IFF proof procedures apply a number of rewrite rules to goals, to obtain a disjunction of answers to some initial goal. The main rewrite rules are unfolding, namely backward reasoning with the iff definitions, and propagation, namely forward reasoning with the integrity constraints. Answers to goals are conjunctions of abducible atoms, and, possibly denial integrity constraints.

Recently, this framework has been applied to problems of multi-agents systems, information integration [83] and management of information networks [98].

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 [32] and the well-founded model semantics [100]. The following ALP frameworks use these semantics for their underlying LP framework.

Bottom up Abduction This work aimed to develop efficient abductive inference procedures for ALP under Kakas and Mancarella's generalized stable model semantics [45]. The approach was based on a translation of Abductive logic programs into pure logic programs with stable model semantics [93]. Abductive solutions w.r.t. the original abductive logic program correspond to stable models of its translation. [93] developed a system for bottom-up stable model computation and used the resulting system to compute abductive solutions. This system is an extension of the procedure for computing well-founded models of [27, 82] and dynamically checks integrity constraints during the computation of stable models and uses them to derive facts.

A purely bottom-up approach has the problem that the system may make many choices for atoms which may not be leading to the query or may simply be irrelevant for the query. To avoid these drawbacks, later the bottom up system was integrated with a procedure for top-down expectation [94, 95] in the model generator. 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 framework has been applied to problems of Legal Reasoning [91, 89] particularly for the computation of similarity of cases and to problems of Consistency Management in Software Engineering [90].

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 [24] and the early definitions of Abductive Logic Programming [45, 46, 43]. 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 solving 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 Generalised Model semantics which extends (in the obvious way) the definition 3.1 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.

ACLP has been applied to several different types of problems including scheduling, planning, time tabling and information integration. The framework of ACLP has also been integrated [54] with Inductive Logic Programming to allow a form of machine learning under incomplete information.

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* [37] and *Preference abduction* [38].

Extended abduction is an extension of an abductive inference method that does not only infer new abductive atoms but may also remove abductive atoms given in the abductive program.

Several methods were proposed for computing extended abduction. [39] proposed a model generation method with term rewriting. In [86, 35], 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 nonmonotonic theories, view update in deductive databases, theory update, contradiction removal, system repair problems with model checking, and inductive logic programming (see [37, 86, 35]).

In preference abduction, priorities between different literals of the program are given. These priorities induce a preference order on the models of the abductive program and on the abductive solutions to queries. The goal of preference abduction is to compute most preferred solutions. A procedure to compute preference abduction has been defined in [38].

Preference abduction can be used in resolution of the multiple extension problem in nonmonotonic reasoning, skeptical abduction, reasoning about rule preference, and preference view update in legal reasoning [38].

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.

In the ABDUAL system, a tabling mechanism has been integrated with the abductive reasoning. The systems executes an abductive query in two stages. First, the program is transformed by adding for each rule R of the finite ground program a new rule for the falsity of R. The resulting program is called the *dual* program. The evaluation method uses tabling, and operates on the dual program.

The ABDUAL framework has been applied in medical diagnosis [25], reasoning about actions [2], solving inconsistencies in metaphorical reasoning [58] and diagnosis of power grid failure [4].

Probabilistic Horn Abduction and Independence Choice Logic Probabilistic Horn abduction [73], which later evolved into the independent choice logic [75], 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 [77] 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 [69], 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 [75], 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 [72, 78], bottom-up implementations (for the ground case) that use a probabilistic variant of the conflicts used in model-based diagnosis [74], and algorithms based on efficient implementations of belief networks that also exploit the context-specific independent inherent in the rule forms [76]. Initial studies of application of ICL have centered around problems of diagnosis and robot control.

4.3 ALP Systems and their Applications

At the implementational level, several general purpose abductive logic programming systems are in development and have been used in recent experiments and initial applications.

- The ACLP system [52, 42], developed at the University of Cyprus, implements the ACLP framework of ALP for a restricted sub-language of the full ACLP framework. Currently, the ACLP 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 [53] 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 [50] application for the small sized company of Cyprus Airways) they have been helpful in indicating some general methodological guidelines that can be followed when one is developing abductive applications. 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.

- The SLDNFAC system 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. 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 [101] has been used in the context of prototypical constraint solving problems such as N-queens, logical puzzles, planning problems in the blocks world, etc . . . An extension of the system has also been used for a small industrial application, namely the scheduling of maintenance for units of power plants. Due to the use of higher order aggregates (e.g. summation and cardinality), the problem was represented by a very short and concise representation. The extended SLDNFAC system with higher order predicates solved this problem by generating a finite domain constraint store which was subsequently solved by the underlying CLP-solver. Comparisons with a pure CLP-approach showed the benefit of this approach: a combination of reasonable efficiency with much less programming effort due to the modular concise declarative knowledge representation in ID-logic. Recently, [67] compared SLDNFAC with different other approaches for solving constraint problems including CLP, ACLP and the Smodel system [64].

- The bottom up abductive procedure of [93] has been implemented using truth maintenance techniques together with top down expectation methods and 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 [91]. The system has also been extended so that legal rules and legal cases are combined together for statutory interpretation [89].

  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 avoid this incompatibility [90].

- The ABDUAL system based on the abductive procedure of [1] is currently implemented on top of XSB-Prolog integrating tabulation and abduction. It consists of a preprocessor for generating the dual program, plus a meta-interpreter for the tabled evaluation of abductive goals. A preliminary metainterpreter for Abdual, written using the XSB system [104], 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 by the meta-interpreter, into the XSB-engine. Work is also underway on the XSB system so that the counfounded set removal operation can be implemented at the engine level.

The ABDUAL system has been applied in medical psychiatric diagnosis [25] 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 commercialised.

ABDUAL has also been employed to detect specification inconsistencies in model-based diagnosis system for power grid failure [4]. 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.

- Two prototype implementations of the IFF procedure [31, 85] exist, one in Java, of the original IFF proof procedure, the other one in April, of the modification of the proof procedure suggested in [84]. The first prototype has been embedded within a Voyager extension that allows interaction and communication amongst multiple agents, as well as cooperative problem solving. This prototype has been applied to information integration from multiple sources [83], to the management of information networks [98], and it has been integrated with PROGOL to learn preconditions of actions in the frameworks of the event and situation calculi.

  Another prototype implementation of the extension of the proof procedure in [102] has been applied to job-shop scheduling [57] and semantic query optimisation [103].

5 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 ALP and other extensions of Logic Programming. In some cases, these links were found to enhance the capabilities of the ALP frameworks either in their computational effectiveness or in their suitability for application. We present here briefly the main links of ALP with other LP extensions.

Answer Set Programming(ASP) . Strong connections have been established between abductive logic programming under generalised stable model semantics and ASP [26]. At the level of semantics these two frameworks are for a large class of theories (ASP admits only a special type of integrity constraints) equivalent. They have though signifigant differences in the computational models that they use. ASP works with a propositional grounding of the theory together with a global satisfaction computation whereas, as we have seen in the previous section, ALP computes directly on the non-propositional abductive theory. It is expected that an integration of these two approaches would enhance the capabilities of both frameworks.

Inductive Logic Programming Abductive and inductive reasoning can be combined together to enhance each other. Abductive reasoning can feed into the inductive process to help prepare the data on which to generalise. Currently, the synthesis of ALP and ILP has been under investigation [63, 105, 29] to allow a form of knowledge intensive learning with complex background theories. In this, abductive inference is used to explain, through the given theory, the example or training data thus giving alternative possibilities for assimilating and generalising this data. Also the framework of ILP can be extended to learn from incomplete background data where the resulting theories are ALP theories. This allows the framework to perform Multiple Predicate Learning in a natural way.

Constraint Logic Programming The integration of constraint solving in abductive logic programming is important both at the language level and the implementation level. At the language level, it enhances the expressivity of the language by allowing arithmetical expressions. At the implementation level, experiments have shown that the use of constraint solving techniques in abductive reasoning make the abductive computation much more efficient. The integrated paradigm of ALP and CLP can be seen as a high-level constraint programming environment that allows modular and flexible representations of the problem domain. The potential benefits of this paradigm are largely unexplored at the moment.

Disjunctive Logic Programming A correspondence between ALP and this paradigm has been shown [87, 88, 106]. The hypothetical reasoning of ALP and the reasoning with indefinite 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 [22] to answer abductive queries. In addition, it has been shown [106] that abductive proof procedures can be used as a basis to develop a computation of DLP programs.

6 Challenges and Prospects for ALP

Over the last two decades many studies have shown how abductive reasoning can be used to address a variety of problems. Many frameworks have been developed for this paradigm. Despite these initial efforts, the field has faced and to some extend continues to do so a number of challenges at the logical, methodological and computational level in trying to achieve its long-term goal of providing an effective declarative problem solving framework suitable for this variety of problems. In this section we attempt to chart out some of these challenges.

6.1 Heterogenity of ALP

The term *abduction* is a very general term and denotes a broad class of reasoning phenomena. This generality carries over to ALP. As can be seen in section 4, ALP is a heterogeneous field. Not only there are quite different formalisms and different semantics but abduction is sometimes used to denote concepts at totally 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, abduction is a concept at the *semantical level*. For example, in the *abductive semantics* for negation as failure [24], abduction is used as a semantical technique in a specific style of defining formal semantics.

But to develop a computational logic, a focused effort at different levels is needed: semantical research to clarify the declarative meaning, knowledge representation research to clarify the applications of the logic, logical research to explore the relation with other logics, and implementation research to investigate how to implement efficient problem solvers. These efforts should link together in a constructive and cross supporting way. The current heterogeneity in the field makes this combination of efforts extremely difficult. More conceptual homogeneity is needed. It is important that the field to some extend focusses on its goals, its terminology, its logic(s) and semantics, and its procedures. More homogeneity might be obtained by further elaborating the logical foundations of ALP (see next section) and selecting a specific class of applications.

6.2 Epistemological foundations of ALP

One of the underlying problems of the field is the lack of epistemological foundations resulting in the heterogeneity exhibited by the field. The *epistemological study* of a logic is concerned with the question of what knowledge can be expressed in a logic and lays the foundation for the systematic development of a knowledge representation methodology for the logic. Hence an epistemological study of ALP can contribute significantly to the understanding of the field at the logical and methodological level.

What knowledge is represented by an abductive logic framework? With exception of the work on OLP and ID-logic, this question has received very little attention in ALP. The definition 3.1 of an abductive solution aims to define the formal *correctness criterion* for abductive reasoning, but does not address the question of how the ALP formalism should be interpreted. Also the (generic) definition 3.2 of the formal model semantics of ALP does not provide clear answers. For example, how is negation in ALP to be understood? The extended completion semantics defined for ALP by Console, Thorasso and Theseider Duprez [8] maps negation as failure literals to classical negation. On the other hand, in [45] Kakas and Mancarella define a generalisation of stable semantics, in which negation as failure literals could also be interpreted as modal literals $\neg Kp$ in autoepistemic logic or defaults [32]. Another open question is the relationship to classical logic. An ALP framework may contain an arbitrary FOL theory T. Does this mean that ALP is an extension of classical logic? A model of an ALP framework is a classical model of T; this suggests that at the epistemological level, ALP is indeed an extension of FOL. On the other hand, ALP is defined as a study of abductive reasoning while FOL is a study of deductive reasoning. How could the first one be an extension of the second one?

The epistemological confusion of ALP complicates the development of a systematic knowledge representation methodology and introduces a lack of coherence in the field. Moreover, it continues to blur the contributions of ALP at the knowledge representation level to the logic-based AI community.

One possible answer to these questions can be found in the earlier mentioned theory of ID-logic [18]. An abductive logic framework (P, A, T) has a natural embedding in ID-logic. P represents a definition of the non-abducible predicates while T represents a set of classical logic assertions. This view on ALP induces a knowledge representation methodology similar and generalising that of description logics (e.g. Krypton), and is based on the representation of *definitional knowledge* by logic program rules and *assertional knowledge* by constraints. In this view ALP is the study of abduction and model generation in the context of definitional and assertional information.

6.3 Computational and Application Challenges

The computational challenges for the paradigm are considerable. The aim of different abductive frameworks to provide a framework for solving a broad class of problems formalised by high-level declarative representations, is extremely difficult to realise. There are formal complexity results [21] that show that in general the problem of computing abduction is very hard. For example, in the general case of ALP frameworks with function symbols, the existence of an abductive solution is an undecidable problem and in the datalog case, the problem of computing abductive solutions is intractable.

On another more practical level this challenge for ALP can be compared to that of CLP but where now we need to understand how we can satisfy effectively a constraint theory that is given at a high logical level, many times beyond the confined domain of arithmetic etc. In some cases, as recent experiments have shown, it is possible to reduce effectively high level ALP representations to lower level constraints that can then be handled by efficient specialised systems. But not all problem can be handled in this way and hence it is instructive for the field, at least at this stage of its development, to focus on particular classes of application problems (see below at the end of this subsection) and to develop particular computational techniques that would be appropriate for these applications.

There are different ways to approach the problem of the computational complexity of ALP. One way is to refine further the current ways to integrate techniques from Constraint Solving in abductive inference. As mentioned earlier, recent experiments have shown that in some class of problems, high level ALP representations can be reduced efficiently to lower level constraints that can then be handled by efficient specialised systems. Such hybrid computational models for ALP need further study. The interface of abductive reasoning with specialised solvers can be a black box one but it is more interesting to examine the case where the integration of such solvers affects dynamically the abductive computation. For example, the generation of a finite domain constraint store alongside with the abductive reduction could be used in many different ways to affect this reduction and steer the search for an abductive solution. An important challenge for ALP is then to study how techniques from constraint solving (and constraint programming more generally) can be incorporated or exploited within the computation of ALP. The search problem of abduction can also benefit from general techniques of heuristics in Artificial Intelligence. In particular, due to the strong link between abduction and planning it would be useful to study how recent developments of heuristic methods of search in planning could be applied to the more general case of abductive computation.

Another complementary approach to addressing the computational hardness of applying ALP would be to develop ALP systems in such a way that the user has the facility to incrementally refine her/his model of the problem in a modular way. Starting from a purely declarative model it should be possible to develop models that contain 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 an modeling environment would be designed and build in ALP.

Abduction has a wide range of potential applications. But what essential features characterise these applications? What features make the use of abduction more appropriate than other approaches? How do we evaluate the benefits of using abduction in an applications? At the moment there are no clear answers to these questions. One of the reasons for this is the fact that there is no focus on the type of applications or primary problem solving tasks that have been examined using abduction and ALP. In many cases the use of abduction is ad hoc with no methodological guidelines on how to develop the application.

As mentioned above, the field needs to define prototypical classes of problems with which it can fine-tune its methods. Three possibilities are:

- Constraint satisfaction problems where more and more problem specific knowledge needs to be exploited. One approach to this as described above would be to develop ALP as a high-level modeling environment over Constraint Programming that would allow the modular exploitation of the additional knowledge.
- The problem of Planning seen as a specific case of the previous problem.
- Knowledge Intensive Learning where machine learning with a rich background knowledge can be performed only if the inductive methods are

integrated with abduction. Here the integration of ALP with ILP could provide a framework that enhances their separate capabilities.

7 Conclusion

Abductive logic programming grew out of attempts to use logic programming techniques for a broad class of AI-problems. 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 still faces a number of problems, 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 recognise 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 system is an extremely difficult one. But we are beginning to learn how to analyse 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.

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] J. J. Alferes, J. A. Leite, L. M. Pereira, P.Quaresma. Planning as Abductive Updating. In D. Kitchin (ed.), Procs. of AISB'00, 2000.

[3] 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.

[4] J.F.Castro, L. M. Pereira, Z.Vale. Power Grid Failure Diagnosis Certification. Technical Report, University of Lisbon.

[5] 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.

[6] E. Charniak and D. McDermott. *Introduction to Artifical Intelligence.* Addison-Wesley, 1985.

[7] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.

[8] L. Console, D. Theseider Dupre, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. *Journal of Logic and Computation*, 5(5):553–578, September 1995.

[15] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for abductive logic programs. *Journal of Logic Programming*, 34(2):111–167, 1998.

[16] 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.

[17] 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.

[18] 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.

[19] Theseider Dupré, D.. Characterizing and Mechanizing Abductive Reasoning. PhD Thesis, Dip. Informatica, Università di Torino, 1994.

[20] P.M. Dung. Negations as hypotheses: an abductive foundation for Logic Programming. In *Proc. of the International Conference on Logic Programming*, 1991.

[21] Thomas Eiter, Georg Gottlob, Nicola Leone. Abduction from Logic Programs: Semantics and Complexity. Theoretical Computer Science 189(1-2):129-177 (1997).

[22] 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.

[23] 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.

[24] K. Eshghi and R.A. Kowalski. Abduction compared with negation as failure. In *Proc. of the International Conference on Logic Programming.* MIT-press, 1989.

[25] 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.

[26] Gelfond, M., Lifschitz, V. Classical negation in logic programs and disjunctive databases. New Generation Computing, pp. 365-387, 1991.

[27] 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.

[28] P. Flach and A. C. Kakas (Eds.). Abduction and Induction: Essays on their Relation and Integration. Kluwer Academic Press, 2000.

[29] 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.

[30] Fung, T.H. Abduction by deduction. Ph.D. Thesis, Imperial College, London, 1996.

[31] T.H. Fung, R.A. Kowalski. The iff procedure for abductive logic programming. In *Journal of Logic Programming* 33(2):151–165, Elsevier, 1997.

[32] 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.

[33] Hobbs, J.R. An integrated abductive framework for discourse interpretation. Symposium on Automated Abduction,Stanford, 1990.

[34] K. Inoue. Hypothetical reasoning in Logic Programs. *Journal of Logic Programming*, 18(3):191–228, 1994.

[35] 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.

[36] 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.

[37] K.Inoue and C. Sakama. Abductive framework for nonmonotonic theory change. In: *Proceedings of IJCAI-95*, pages 204–210, Morgan Kaufmann, 1995.

[38] K. Inoue and C. Sakama. Abducing priorities to derive intended conclusions. In: *Proceedings of IJCAI-99*, pages 44–49, Morgan Kaufmann, 1999.

[39] K. Inoue and C. Sakama. Computing extended abduction through transaction programs. *Annals of Mathematics and Artificial Intelligence*, 25(3,4):339–367, 1999.

[40] J.R. Josephson and S.G. Josephson, editors. *Abductive Inference: Computation, Philosophy, Technology.* New York: Cambridge University Press, 1994.

[41] 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.

[42] A. C. Kakas. ACLP: Integrating Abduction and Constraint Solving. In Proceedings of NMR2000, 2000.

[43] Kakas, A. C., Kowalski, R. A., Toni, F., Abductive logic programming. *Journal of Logic and Computation* 2(6) (1993) 719–770

[44] 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)

[45] A.C. Kakas and P. Mancarella. Generalised Stable Models: a Semantics for Abduction. In Proc. 9th European Conference on AI, ECAI90, Stockolm, 1990.

[46] 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.

[47] 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.

[48] 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.

[49] Kakas, A. C., Michael, A. Integrating abductive and constraint logic programming. In *Proc. International Logic Programming Conference*, pp. 399-413, 1995.

[50] A.C. Kakas and A. Michael. Air-Crew Scheduling through Abduction. Proceedings of IEA/AIE-99, pp. 600–612, 1999.

[51] A.C. Kakas, A. Michael and C. Mourlas. ACLP: a case for non-monotonic reasoning. in Proceedings of NMR98,pp. 46–56, 1998.

[52] 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.

[53] A.C. Kakas and C. Mourlas. ACLP: Flexible Solutions to Complex Problems. Proceedings of Logic Programming and Non-monotonic Reasoning, LPNMR97, 1997,

[54] A.C. Kakas and F. Riguzzi. Abductive Concept Learning. New Generation Computing, Vol. 18, pp. 243-294, 2000.

[55] Kowalski, R.A.; Sadri, F.; 1999. From Logic Programming to Multi-agent Systems. *Annals of Mathemathics and Artificial Intelligence*

[56] R.A. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(4):319–340, 1986.

[57] Kowalski, R.A.; Toni, F.; Wetzel, G.; 1998. Executing suspended logic programs. *Fundamenta Informaticae* 34(3):203–224, ISO Press.

[58] 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.

[59] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[60] J. McCarthy. Situations, actions and causal laws. Technical Report AI-memo 1, Artifical Intelligence Program, Standford University, 1957.

[61] 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.

[62] L.R. Missiaen, M. Bruynooghe, and M. Denecker. Abductive planning with event calculus. Internal report, Department of Computer Science, K.U.Leuven, 1992.

[63] S. Muggleton. Theory Completion in Learning. In Proceedings of Inductive Logic Programming, ILP00, 2000.

[64] 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.

[65] M. Pagnucco. The role of abductive reasoning within the process of belief revision. PhD Thesis, Department of Computer Science, University of Sydney, 1996.

[66] C.S. Peirce. *Philosophical Writings of Peirce.* Dover Publications, New York, 1955.

[67] 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.

[68] 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.

[69] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, CA, 1988.

[70] D. Poole. A Logical Framework for Default Reasoning. *Artifical Intelligence*, 36:27–47, 1988.

[71] D. Poole. A methodology for using a default and abductive reasoning system. *International Journal of Intelligent Systems*, 5(5):521–548, December 1990.

[72] 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.

[73] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[74] D. Poole. Probabilistic conflicts in a search algorithm for estimating posterior probabilities in Bayesian networks. *Artificial Intelligence*, 88:69–100, 1996.

[75] 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.

[76] 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.

[77] 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.

[78] 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.

[79] 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.

[80] Stathis Psillos. Ampliative Reasoning: Induction or Abduction. In *ECAI96 workshop on Abductive and Inductive Reasoning*, 1996.

[81] 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.

[82] Saccà, D., Zaniolo, C. Stable Models and Non-Determinism in Logic Programs with Negation. Proc. of PODS'90, pp. 205 – 217, 1990.

[83] 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.

[84] 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.

[85] 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.

[86] 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.

[87] 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.

[88] 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.

[89] 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.

[90] 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.

[91] Satoh, K. Using Two Level Abduction to Decide Similarity of Cases. Proc. of ECAI'98 pp. 398 – 402, 1998.

[92] 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.

[93] Satoh, K. and Iwayama, N. Computing Abduction by Using the TMS. Proc. of ICLP'91, pp. 505 – 518, 1991.

[94] Satoh, K. and Iwayama, N. A Query Evaluation Method for Abductive Logic Programming. Proc. of JICSLP'92, pp. 671 – 685, 1992.

[95] Iwayama, N., Satoh, K. Computing Abduction by Using TMS with Top-Down Expectation. Journal of Logic Programming, Vol. 44 No.1-3, pp. 179 – 206, 2000.

[96] 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).

[97] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proc. of the IJCAI89*, page 1055, 1989.

[98] 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

[99] 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.

[100] 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.

[101] 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.

[102] Wetzel, G. *Abductive and Constraint Logic Programming.* Ph.D. Thesis, Imperial College, London, 1997.

[103] 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).

[104] The XSB Group. The XSB logic programming system, version 2.0. 1999. Available from `http://www.cs.sunysb.edu/~sbprolog`.

[105] 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.

[106] 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.