The Role of Abduction in Logic Programming

A.C. Kakas
Department of Computer Science,
University of Cyprus,
75 Kallipoleos Street,
Nicosia P.O. Box 537, Cyprus.
antonis@turing.cs.ucy.ac.cy

R.A. Kowalski, F. Toni
Department of Computing,
Imperial College of Science, Technology and Medicine,
180 Queen's Gate,
London SW7 2BZ, UK.
{rak,ft}@doc.ic.ac.uk

Abstract

This paper is a survey and critical overview of recent work on the extension of Logic Programming to perform Abductive Reasoning (Abductive Logic Programming). It updates the earlier paper "Abductive Logic Programming" [88]. We outline the general framework of Abduction and its applications to Knowledge Assimilation and Default Reasoning; we describe the argumentation-theoretic approach to the use of abduction as an interpretation for Negation as Failure, introduced in the earlier version [88] of this paper; and we present recent work on the generalisation of the argumentation-theoretic approach to provide a framework for default reasoning in general. We also analyse the links between Abduction and Constraint Logic Programming, as well as between Abduction and the extension of Logic Programming obtained by adding a form of explicit negation. Finally we discuss the relation between Abduction and Truth Maintenance.

Contents

1 Introduction

This paper extends and updates our earlier survey and analysis of work on the extension of logic programming to perform abductive reasoning [88]. The purpose of the paper is to provide a critical overview of some of the main research results, in order to develop a common framework for evaluating these results, to identify the main unresolved problems, and to indicate directions for future work. The emphasis is not on technical details but on relationships and common features of different approaches. Some of the main issues we will consider are the contributions that abduction can make to the problems of reasoning with incomplete or negative information, the evolution of knowledge, and the semantics of logic programming and its extensions. We also discuss recent work on the argumentation-theoretic interpretation of abduction, which was introduced in the earlier version of this paper.

The philosopher Pierce first introduced the notion of abduction. In [133] he identified three distinguished forms of reasoning.

Deduction, an analytic process based on the application of general rules to particular cases, with the inference of a result.

Induction, synthetic reasoning which infers the rule from the case and the result.

Abduction, another form of synthetic inference, but of the case from a rule and a result.

Peirce further characterised abduction as the "probational adoption of a hypothesis" as explanation for observed facts (results), according to known laws. "It is however a weak kind of inference, because we cannot say that we believe in the truth of the explanation, but only that it may be true"[133].

Abduction is widely used in common-sense reasoning, for instance in diagnosis, to reason from effect to cause [22, 142]. We consider here an example drawn from [131].

Example 1.1
Consider the following theory T

$$\begin{aligned} grass\text{-}is\text{-}wet &\leftarrow rained\text{-}last\text{-}night \\ grass\text{-}is\text{-}wet &\leftarrow sprinkler\text{-}was\text{-}on \\ shoes\text{-}are\text{-}wet &\leftarrow grass\text{-}is\text{-}wet. \end{aligned}$$

If we observe that our shoes are wet, and we want to know why this is so, $\{rained\text{-}last\text{-}night\}$ is a possible explanation, i.e. a set of hypotheses that together with the explicit knowledge in T implies the given observation. $\{sprinkler\text{-}was\text{-}on\}$ is another alternative explanation.

Abduction consists of computing such explanations for observations. It is a form of non-monotonic reasoning, because explanations which are consistent with one state of a knowledge base may become inconsistent with new information. In the example above the explanation *rained-last-night* may turn out to be false, and the alternative explanation *sprinkler-was-on* may be the true cause for the given observation. The existence of **multiple explanations** is a general characteristic of abductive reasoning, and the selection of "preferred" explanations is an important problem.

1.1 Abduction in logic

Given a set of sentences T (a theory presentation), and a sentence G (observation), to a first approximation, the abductive task can be characterised as the problem of finding a set of sentences Δ (abductive explanation for G) such that:

(1) $T \cup \Delta \models G$,

(2) $T \cup \Delta$ is consistent.

This characterisation of abduction is independent of the language in which T, G and Δ are formulated. The logical implication sign $\models$ in (1) can alternatively be replaced by a deduction operator $\vdash$. The consistency requirement in (2) is not explicit in Peirce's more informal characterisation of abduction, but it is a natural further requirement.

In fact, these two conditions (1) and (2) alone are too weak to capture Peirce's notion. In particular, additional restrictions on Δ are needed to distinguish abductive explanations from inductive generalisations [27]. Moreover, we also need to restrict Δ so that it conveys some reason why the observations hold, e.g. we do not want to explain one effect in terms of another effect, but only in terms of some cause. For both of these reasons, explanations are often restricted to belong to a special pre-specified, domain-specific class of sentences called **abducible**. In this paper we will assume that the class of abducibles is always given.

Additional criteria have also been proposed to restrict the number of candidate explanations:

- Once we restrict the hypotheses to belong to a specified set of sentences, we can further restrict, without loss of generality, the hypotheses to atoms (that "name" these sentences) which are predicates explicitly indicated as abducible, as shown by Poole [145].
- In section 1.2 we will discuss the use of integrity constraints to reduce the number of possible explanations.
- Additional information can help to discriminate between different explanations, by rendering some of them more appropriate or plausible than others. For example Sattar and Goebel [173] use "crucial literals" to discriminate between two mutually incompatible explanations. When the crucial literals are tested, one of the explanations is rejected. More generally Evans and Kakas [56] use the notion of corroboration to select explanations. An explanation fails to be corroborated if some of its logical consequences are not observed. A related technique is presented by Sergot in [175], where information is obtained from the user during the process of query evaluation.
- Moreover various (domain specific) criteria of preference can be specified. They impose a (partial) order on the sets of hypotheses which leads to the discrimination of explanations [13, 22, 61, 77, 143, 148, 180].

Cox and Pietrzykowski [29] identify other desirable properties of abductive explanations. For instance, an explanation should be **basic**, i.e. should not be explainable in terms of other explanations. For instance, in example 1.1 the explanation

$$\{grass\text{-}is\text{-}wet\}$$

for the observation

$$shoes\text{-}are\text{-}wet$$

is not basic, whereas the alternative explanations

$$\{rained\text{-}last\text{-}night\}$$
$$\{sprinkler\text{-}was\text{-}on\}$$

are.

An explanation should also be **minimal**, i.e. not subsumed by another one. For example, in example 1.1 the explanation

$$\{rained\text{-}last\text{-}night,\ sprinkler\text{-}was\text{-}on\}$$

for the observation

$$shoes\text{-}are\text{-}wet$$

is not minimal, while the explanations

$$\{rained\text{-}last\text{-}night\}$$
$$\{sprinkler\text{-}was\text{-}on\}$$

are.

So far we have presented a semantic characterisation of abduction and discussed some heuristics to deal with the multiple explanation problem, but we have not described any proof procedures for computing abduction. Various authors have suggested the use of top-down, goal-oriented computation, based on the use of deduction to drive the generation of abductive hypotheses. Cox and Pietrzykowski [29] construct hypotheses from the "dead ends" of linear resolution proofs. Finger and Genesereth [57] generate "deductive solutions to design problems" using the "residue" left behind in resolution proofs. Poole, Goebel and Aleliunas [150] also use linear resolution to generate hypotheses.

In contrast, the ATMS [102] computes abductive explanations bottom-up. The ATMS can be regarded as a form of hyper-resolution, augmented with subsumption, for propositional logic programs [162]. Lamma and Mello [115] have developed an extension of the ATMS for the non-propositional case. Resolution-based techniques for computing abduction have also been developed by Demolombe and Fariñas del Cerro [31] and Gaifman and Shapiro [64].

Abduction can also be applied to logic programming (LP). A **(general) logic program** is a set of Horn clauses extended by negation as failure [24], i.e. **clauses** of the form:

$$A \leftarrow L_1, \ldots, L_n$$

where each L_i is either an atom A_i or its negation $\sim A_i$ [1], A is an atom and each variable occurring in the clause is implicitly universally quantified. A is called the head and $L_1, \ldots, L_n$ is called the body of the clause. A logic program where each literal L_i in the body of every clause is atomic is said to be **definite**.

Abduction can be computed in LP by extending SLD and SLDNF [23, 53, 54, 91, 94, 34, 181]. Instead of failing in a proof when a selected subgoal fails to unify with the head of any rule, the subgoal can be viewed as a hypothesis. This is similar to viewing abducibles as "askable" conditions which are treated as qualifications to answers to queries [175]. In the same way that it is useful to distinguish a subset of all predicates as "askable", it is useful to distinguish certain predicates as abducible. In fact, it is generally convenient to choose, as abducible predicates, ones which are not conclusions of any clause. As we shall remark at the beginning of section 5, this restriction can be imposed without loss of generality, and has the added advantage of ensuring that all explanations will be basic.

Abductive explanations computed in LP are guaranteed to be minimal, unless the program itself encodes non-minimal explanations. For example, in the propositional logic program

$$\begin{array}{lcl} p & \leftarrow & q \\ p & \leftarrow & q, r \end{array}$$

both the minimal explanation $\{q\}$ and the non-minimal explanation $\{q, r\}$ are computed for the observation p.

The abductive task for the logic-based approach has been proved to be highly intractable: it is NP-hard even if T is a set of acyclic [7] propositional definite clauses [174, 48], and is even harder if T is a set of any propositional clauses [48]. These complexity results hold even if explanations are not required to be minimal. However, the abductive task is tractable for certain more restricted classes of logic programs (see for example [52]).

There are other formalisations of abduction. We mention them for completeness, but in the sequel we will concentrate on the logic-based view previously described.

- Allemand, Tanner, Bylander and Josephson [6] and Reggia [155] present a mathematical characterisation, where abduction is defined over sets of observations and hypotheses, in terms of coverings and parsimony.

- Levesque [117] gives an account of abduction at the "knowledge level". He characterises abduction in terms of a (modal) logic of beliefs, and shows how the logic-based approach to abduction can be understood in terms of a particular kind of belief.

In the previous discussion we have briefly described both **semantics** and **proof procedures** for abduction. The relationship between semantics and proof procedures can be understood as a special case of the relationship between program specifications and programs. A program specification characterises what is the intended result expected from the execution of the program. In the same way semantics can be viewed as an abstract,

[1] In the sequel we will represent negation as failure as $\sim$.

possibly non-constructive definition of what is to be computed by the proof procedure. From this point of view, semantics is not so much concerned with explicating meaning in terms of truth and falsity, as it is with providing an abstract specification which "declaratively" expresses what we want to compute. This specification view of semantics is effectively the one adopted in most recent work on the semantics of LP, which restricts interpretations to Herbrand interpretations. The restriction to Herbrand interpretations means that interpretations are purely syntactic objects, which have no bearing on the correspondence between language and "reality". A purely syntactic view of semantics, based upon the notion of knowledge assimilation described in section 2 below, is developed in [110].

One important alternative way to specify the semantics of a language, which will be used in the sequel, is through the **translation** of sentences expressed in one language into sentences of another language, whose semantics is already well understood. For example if we have a sentence in a typed logic language of the form "there exists an object of type t such that the property p holds" we can translate this into a sentence of the form $\exists x\,(p(x) \wedge t(x))$, where t is a new predicate to represent the type t, whose semantics is then given by the familiar semantics of first-order logic. Similarly the typed logic sentence "for all objects of type t the property p holds" becomes the sentence $\forall x(p(x) \leftarrow t(x))$. Hence instead of developing a new semantics for the typed logic language, we apply the translation and use the existing semantics of first-order logic.

1.2 Integrity Constraints

Abduction as presented so far can be restricted by the use of integrity constraints. Integrity constraints are useful to avoid unintended updates to a database or knowledge base. They can also be used to represent desired properties of a program [116].

The concept of integrity constraints first arose in the field of databases and to a lesser extent in the field of AI knowledge representation. The basic idea is that only certain knowledge base states are considered acceptable, and an integrity constraint is meant to enforce these legal states. When abduction is used to perform updates (see section 2), we can use integrity constraints to reject abductive explanations.

Given a set of integrity constraints, I, of first-order closed formulae, the second condition (2) of the semantic definition of abduction (see section 1.1) can be replaced by:

(2′) $T \cup \Delta$ satisfies I.

As previously mentioned, we also restrict Δ to consist of atoms drawn from predicates explicitly indicated as abducible. Until the discussion in section 5.7, we further restrict Δ to consist of **variable-free atomic sentences**.
In the sequel an **abductive framework** will be given as a triple $\langle T, A, I\rangle$, where T is a theory, A is the set of abducible predicates, i.e. $\Delta \subseteq A$ [2] and I is a set of integrity constraints.

[2]Here and in the rest of this paper we will use the same symbol A to indicate both the set of abducible predicates and the set of all their variable-free instances.

There are several ways to define what it means for a knowledge base KB ($T \cup \Delta$ in our case) to satisfy an integrity constraint ϕ (in our framework $\phi \in I$). The **consistency view** requires that:

$$KB \text{ satisfies } \phi \text{ iff } KB \cup \phi \text{ is consistent.}$$

Alternatively the **theoremhood view** requires that:

$$KB \text{ satisfies } \phi \text{ iff } KB \models \phi.$$

These definitions have been proposed in the case where the theory is a logic program P by Kowalski and Sadri [165] and Lloyd and Topor [118] respectively, where KB is the Clark completion [24] of P.

Another view of integrity constraints [85, 90, 107, 160, 161] regards these as **epistemic** or **metalevel** statements about the content of the database. In this case the integrity constraints are understood as statements at a different level from those in the knowledge base. They specify what must be true about the knowledge base rather than what is true about the world modelled by the knowledge base. When later we consider abduction in LP (see sections 4,5), integrity satisfaction will be understood in a sense which is stronger than consistency, weaker than theoremhood, and arguably similar to the epistemic or metalevel view.

For each such semantics, we have a specification of the integrity checking problem. Although the different views of integrity satisfaction are conceptually very different, the integrity checking procedures based upon these views are not very different in practice (e.g. [30, 165, 118]). They are mainly concerned with avoiding the inefficiency which arises if all the integrity constraints are retested after each update. A common idea of all these procedures is to render integrity checking more efficient by exploiting the assumption that the database before the update satisfies the integrity constraints, and therefore if integrity constraints are violated after the update, this violation should depend upon the update itself. In [165] this assumption is exploited by reasoning forward from the updates. This idea is exploited for the purpose of checking the satisfaction of abductive hypotheses in [54, 93, 94]. Although this procedure was originally formulated for the consistency view of constraint satisfaction, it has proved equally appropriate for the semantics of integrity constraints in abductive logic programming.

1.3 Applications

In this section we briefly describe some of the applications of abduction in AI. In general, abduction is appropriate for reasoning with incomplete information. The generation of abducibles to solve a top-level goal can be viewed as the addition of new information to make incomplete information more complete.

Abduction can be used to generate causal explanations for **fault diagnosis** (see for example [25, 151]). In medical diagnosis, for example, the candidate hypotheses are the possible causes (diseases), and the observations are the symptoms to be explained [146, 155]. Abduction can also be used for model-based diagnosis [51, 159]. In this case the theory describes the "normal" behaviour of the system, and the task is to find a set of hypotheses of the form "some component A is not normal" that explains why the behaviour of the system is not normal.

Abduction can be used to perform **high level vision** [29]. The hypotheses are the objects to be recognised, and the observations are partial descriptions of objects.

Abduction can be used in **natural language understanding** to interpret ambiguous sentences [22, 62, 78, 179]. The abductive explanations correspond to the various possible interpretations of such sentences.

In **planning** problems, plans can be viewed as explanations of the given goal state to be reached [50, 176].

These applications of abduction can all be understood as generating hypotheses which are causes for observations which are effects. An application that does not necessarily have a direct causal interpretation is **knowledge assimilation** [94, 105, 114, 125], described in greater detail below. The assimilation of a new datum can be performed by adding to the theory new hypotheses that are explanations for the datum. Knowledge assimilation can also be viewed as the general context within which abduction takes place. **Database view updates** [17, 91, 28] are an important special case of knowledge assimilation. Update requests are interpreted as observations to be explained. The explanations of the observations are transactions that satisfy the update request.

Another important application which can be understood in terms of a "non-causal" use of abduction is **default reasoning**. Default reasoning concerns the use of general rules to derive information in the absence of contradictions. In the application of abduction to default reasoning, conclusions are viewed as observations to be explained by means of assumptions which hold by default unless a contradiction can be shown [53, 145]. As Poole [145] argues, the use of abduction avoids the need to develop a non-classical, non-monotonic logic for default reasoning. In section 3 we will further discuss the use of abduction for default reasoning in greater detail. Because negation as failure in LP is a form of default reasoning, its interpretation by means of abduction will be discussed in section 4.

Some authors (e.g. Pearl [132]) advocate the use of probability theory as an alternative approach to common sense reasoning in general, and to many of the applications listed above in particular. However, Poole [149] shows how abduction can be used to simulate (discrete) Bayesian networks in probability theory. He proposes the language of probabilistic Horn abduction: in this language an abductive framework is a triple $\langle T, A, I\rangle$, where T is a set of Horn clauses, A is a set of abducibles without definitions in T (without loss of generality, see section 5), and I is a set of integrity constraints in the form of denials of abducibles only. In addition, for each integrity constraint, a probability value is assigned to each abducible, so that the sum of all the values of all the abducibles in each integrity constraint is 1. If the abductive framework satisfies certain assumptions, e.g. T is acyclic [7], the bodies of all the clauses defining each non-abducible atom are mutually exclusive and these clauses are "covering", and abducibles in A are "probabilistically independent", then such a probabilistic Horn abduction theory can be mapped onto a (discrete) Bayesian network and vice versa.

2 Knowledge Assimilation

Abduction takes place in the context of assimilating new knowledge (information, belief or data) into a theory (or knowledge base). There are four possible deductive relationships between the current knowledge base (KB), the new information, and the new KB which arises as a result [105, 110].

1. The new information is already deducible from the current KB. The new KB, as a result, is identical with the current one.
2. The current KB = $KB_1 \cup KB_2$ can be decomposed into two parts. One part KB_1 together with the new information can be used to deduce the other part KB_2. The new KB is KB_1 together with the new information.
3. The new information violates the integrity of the current KB. Integrity can be restored by modifying or rejecting one or more of the assumptions which lead to the contradiction.
4. The new information is independent from the current KB. The new KB is obtained by adding the new information to the current KB.

In case (4) the KB can, alternatively, be augmented by an explanation for the new datum [94, 105, 114]. In [114] the authors have developed a system for knowledge assimilation (KA) based on this use of abduction. They have identified the basic issues associated with such a system and proposed solutions for some of these.

Various motivations can be given for the addition of an abductive explanation instead of the new datum in case (4) of the process of KA. For example, in natural language understanding or in diagnosis, the assimilation of information naturally demands an explanation. In other cases the addition of an explanation as a way of assimilating new data is forced by the particular way in which the knowledge is represented in the theory. This is the case, for instance, for the formulation of temporal reasoning in the Event Calculus [113, 108], as illustrated by the following example.

Example 2.1
The simplified version of the event calculus we consider contains an axiom that expresses the persistence of a property P from the time T_1 that it is initiated by an event E to a later time T_2:

$$\begin{aligned} holds\_at(P, T_2) \leftarrow\ & happens(E, T_1), \\ & T_1 < T_2, \\ & initiates(E, P), \\ & persists(T_1, P, T_2). \end{aligned}$$

New information that a property holds at a particular time point can be assimilated by adding an explanation in terms of the happening of some event that initiates this property at an earlier point of time together with an appropriate assumption that the property persists from one time to the other [50, 89, 176, 186]. This has the additional effect that the new KB will imply that the property holds until it is terminated in the future by the happening of some event [176]. The fact that a property P cannot persist from a time T_1 to a later time T_2 if an event E happens at a time T between T_1 and T_2 such that E terminates P is expressed by the following integrity constraint:

$$\neg[persists(T_1,\ P,\ T_2) \wedge happens(E,\ T) \wedge terminates(E,\ P) \wedge T_1 < T < T_2].$$

Assimilating new information by adding explanations that satisfy the integrity constraints has the further effect of resolving conflicts between the current KB and the new information [89, 176]. For example, suppose that KB contains the facts [3]

$$\begin{array}{l} happens(takes\_book(mary), t_0) \\ initiates(takes\_book(X), has\_book(X)) \\ terminates(gives\_book(X, Y), has\_book(X)) \\ initiates(gives\_book(X, Y), has\_book(Y)) \end{array}$$

Then, given $t_0 < t_1 < t_2$, the persistence axiom predicts $holds\_at(has\_book(mary), t_1)$ by assuming $persists(t_0, has\_book(mary), t_1)$, and $holds\_at(has\_book(mary), t_2)$ by assuming $persists(t_0, has\_book(mary), t_2)$. Both these assumptions are consistent with the integrity constraint. Suppose now that the new information $holds\_at(has\_book(john), t_2)$ is added to KB. This conflicts with the prediction $holds\_at(has\_book(mary), t_2)$. However, the new information can be assimilated by adding to KB the hypotheses $happens(gives\_book(mary, john), t_1)$ and $persists(t_1, has\_book(john), t_2)$ and by retracting the hypothesis $persists(t_0, has\_book(mary), t_2)$. Therefore, the earlier prediction $holds\_at(has\_book(mary), t_2)$ can no longer be derived from the new KB.

Note that in this example the hypothesis $happens(gives\_book(mary, john), t_1)$ can be added to KB since it does not violate the further integrity constraint

$$\neg[happens(E, T) \wedge precondition(E, T, P) \wedge \sim holds\_at(P, T)]$$

expressing that an event E cannot happen at a time T if the preconditions P of E do not hold at time T. In this example, we may assume that KB also contains the fact

$$precondition(gives\_book(X, Y), has\_book(X)).$$

Once a hypothesis has been generated as an explanation for an external datum, it itself needs to be assimilated into the KB. In the simplest situation, the explanation is just added to the KB, i.e. only case (4) applies without further abduction. Case (1) doesn't apply, if abductive explanations are required to be basic. However case (2) may apply, and can be particularly useful for discriminating between alternative explanations for the new information. For instance we may prefer a set of hypotheses which entails information already in the KB, i.e. hypotheses that render the KB as "compact" as possible.

Example 2.2
Suppose the current KB contains

$$\begin{array}{lcl} p & \leftarrow & q \\ p & & \\ r & \leftarrow & q \\ r & \leftarrow & s \end{array}$$

[3] Note that here KB contains a definition for the abducible predicate $happens$. In section 5 we will see that new predicates and clauses can be added to KB so that abducible predicates have no definitions in the transformed KB.

and r is the new datum to be assimilated. The explanation $\{q\}$ is preferable to the explanation $\{s\}$, because q implies both r and p, but s only implies r. Namely, the explanation $\{q\}$ is more relevant.

Notice however that the use of case (2) to remove redundant information can cause problems later. If we need to retract previously inserted information, entailed information which is no longer explicitly in the KB might be lost.

It is interesting to note that case (3) can be used to check the integrity of abductive hypotheses generated in case (4).

Any violation of integrity detected in case (3) can be remedied in several ways [105]. The new input can be retracted as in conventional databases. Alternatively the new input can be upheld and some other assumptions can be withdrawn. This is the case with **view updates**. The task of translating the update request on the view predicates to an equivalent update on the extensional part (as in case (4) of KA) is achieved by finding an abductive explanation for the update in terms of variable-free instances of extensional predicates [91]. Any violation of integrity is dealt with by changing the extensional part of the database.

Example 2.3
Suppose the current KB consists of the clauses

$$\begin{array}{l} sibling(X, Y) \leftarrow parent(Z, X),\ parent(Z, Y) \\ parent(X, Y) \leftarrow father(X, Y) \\ parent(X, Y) \leftarrow mother(X, Y) \\ father(john, mary) \\ mother(jane, mary) \end{array}$$

together with the integrity constraints

$$\begin{array}{l} X = Y \leftarrow father(X, Z),\ father(Y, Z) \\ X = Y \leftarrow mother(X, Z),\ mother(Y, Z) \\ X \neq Y \leftarrow mother(X, Z),\ father(Y, W) \end{array}$$

where $sibling$ and $parent$ are view predicates, $father$ and $mother$ are extensional, and $=, \neq$ are "built-in" predicates such that

$X = X$ and
$s \neq t$ for all distinct variable-free terms s and t.

Suppose the view update

insert $sibling(mary, bob)$

is given. This can be translated into either of the two minimal updates

insert $father(john, bob)$
insert $mother(jane, bob)$

on the extensional part of the KB. Both of these updates satisfy the integrity constraints. However, only the first update satisfies the integrity constraints if we are given the further update

$$\text{insert } \mathit{mother}(\mathit{sue}, \mathit{bob}).$$

The general problem of belief revision has been studied formally in [65, 128, 129, 37]. Gärdenfors proposes a set of axioms for rational belief revision containing such constraints on the new theory as "no change should occur to the theory when trying to delete a fact that is not already present" and "the result of revision should not depend on the syntactic form of the new data". These axioms ensure that there is always a unique way of performing belief revision. However Doyle [37] argues that, for applications in AI, this uniqueness property is too strong. He proposes instead the notion of "economic rationality", in which the revised sets of beliefs are optimal, but not necessarily unique, with respect to a set of preference criteria on the possible beliefs states. This notion has been used to study the evolution of databases by means of updates [86]. It should be noted that the use of abduction to perform belief revision in the view update case also allows results which are not unique, as illustrated in example 2.3. Aravindan and Dung [8] have given an abductive characterisation of rational belief revision and have applied this result to formulate belief revision postulates for the view update problem.

A logic-based theory of the assimilation of new information has also been developed in the Relevance Theory of Sperber and Wilson [178] with special attention to natural language understanding. Gabbay, Kempson and Pitts [63] have investigated how abductive reasoning and relevance theory can be integrated to choose between different abductive interpretations of a natural language discourse.

KA and belief revision are also related to truth maintenance systems. We will discuss truth maintenance and its relationship with abduction in section 8.

3 Default Reasoning viewed as Abduction

Default reasoning concerns the application of general rules to draw conclusions provided the application of the rules does not result in contradictions. Given, for example, the general rules "birds fly" and "penguins are birds that do not fly" and the only fact about Tweety that Tweety is a bird, we can derive the default conclusion that Tweety flies. However, if we are now given the extra information that Tweety is a penguin, we can also conclude that Tweety does not fly. In ordinary, common sense reasoning, the rule that penguins do not fly has priority over the rule that birds fly, and consequently this new conclusion that Tweety does not fly causes the original conclusion to be withdrawn.

One of the most important formalisations of default reasoning is the Default Logic of Reiter [158]. Reiter separates beliefs into two kinds, ordinary sentences used to express "facts" and default rules of inference used to express general rules. A **default rule** is an inference rule of the form

$$\frac{\alpha(X) : \mathrm{M}\beta_1(X), \ldots, \mathrm{M}\beta_n(X)}{\gamma(X)}$$

which expresses, for all variable-free instances t of X [4], that $\gamma(t)$ can be derived if $\alpha(t)$ holds and each of $\beta_i(t)$ is consistent, where $\alpha(X)$, $\beta_i(X)$, $\gamma(X)$ are first-order formulae. Default rules provide a way of extending an underlying incomplete theory. Different applications of the defaults can yield different extensions.

As already mentioned in section 1, Poole, Goebel and Aleliunas [150] and Poole [145] propose an alternative formalisation of default reasoning in terms of abduction. Like Reiter, Poole also distinguishes two kinds of beliefs:

- beliefs that belong to a consistent set of first order sentences $\mathcal{F}$ representing "facts", and
- beliefs that belong to a set of first order formulae D representing defaults.

Perhaps the most important difference between Poole's and Reiter's formalisations is that Poole uses sentences (and formulae) of classical first order logic to express defaults, while Reiter uses rules of inference. Given a Theorist framework $\langle \mathcal{F}, D \rangle$, default reasoning can be thought of as theory formation. A new theory is formed by extending the existing theory $\mathcal{F}$ with a set Δ of sentences which are variable-free instances of formulae in D. The new theory $\mathcal{F} \cup \Delta$ should be consistent. This process of theory formation is a form of abduction, where variable-free instances of defaults in D are the candidate abducibles. Poole [145] shows that the semantics of the theory formation framework $\langle \mathcal{F}, D \rangle$ is equivalent to that of an abductive framework $\langle \mathcal{F}', A, \emptyset \rangle$ (see section 1.2) where the default formulae are all atomic. The set of abducibles A consists of a new predicate

$$p_w(x)$$

for each default formula

$$w(x)$$

in D with free variables x. The new predicate is said to "name" the default. The set $\mathcal{F}'$ is the set $\mathcal{F}$ augmented with a sentence

$$\forall X \, [p_w(X) \rightarrow w(X)]$$

for each default in D.

The theory formation framework and its correspondence with the abductive framework can be illustrated by the flying-birds example.

Example 3.1
In this case, the framework $\langle \mathcal{F}, D \rangle$ is [5]

$$\mathcal{F} \;\; = \{ \;\; \mathit{penguin}(X) \rightarrow \mathit{bird}(X),$$

[4] We use the notation X to indicate a tuple of variables $X_1, \ldots, X_n$ and t to represent a tuple of terms $t_1, \ldots, t_n$.

[5] Here, we use the conventional notation of first-order logic, rather than LP form. We use $\rightarrow$ for the usual implication symbol for first-order logic in contrast with $\leftarrow$ for LP. However, as in LP notation, variables occurring in formulae of $\mathcal{F}$ are assumed to be universally quantified. Formulae of D, on the other hand, should be understood as schemata standing for the set of all their variable-free instances.

$$\begin{aligned} & penguin(X) \rightarrow \neg fly(X), \\ & penguin(tweety), \\ & bird(john)\} \\ D = \{ \; & bird(X) \rightarrow fly(X)\}. \end{aligned} \tag{1}$$

The priority of the rule that penguins do not fly over the rule that birds fly is obtained by regarding the first rule as a fact and the second rule as a default. The atom $fly(john)$ is a default conclusion which holds in $\mathcal{F} \cup \Delta$ with

$$\Delta = \{ bird(john) \rightarrow fly(john) \}.$$

We obtain the same conclusion by naming the default (1) by means of a predicate $birds\text{-}fly(X)$, adding to $\mathcal{F}$ the new "fact"

$$birds\text{-}fly(X) \rightarrow [bird(X) \rightarrow fly(X)] \tag{2}$$

and extending the resulting augmented set of facts $\mathcal{F}'$ with the set of hypotheses $\Delta' = \{ birds\text{-}fly(john) \}$. On the other hand, the conclusion $fly(tweety)$ cannot be derived, because the extension

$$\Delta = \{ bird(tweety) \rightarrow fly(tweety) \}$$

is inconsistent with $\mathcal{F}$, and similarly the extension

$$\Delta' = \{ birds\text{-}fly(tweety) \}$$

is inconsistent with $\mathcal{F}'$.

Poole shows that normal defaults without prerequisites in Reiter's default logic

$$\frac{: \mathrm{M}\beta(X)}{\beta(X)}$$

can be simulated by Theorist (abduction) simply by making the predicates $\beta(X)$ abducible. He shows that the default logic extensions in this case are equivalent to maximal sets of variable-free instances of the default formulae $\beta(X)$ that can consistently be added to the set of facts.

Maximality of abductive hypotheses is a natural requirement for default reasoning, because we want to apply defaults whenever possible. However, maximality is not appropriate for other uses of abductive reasoning. In particular, in diagnosis we are generally interested in explanations which are minimal. Later, in section 5.1 we will distinguish between default and non-default abducibles in the context of abductive logic programming.

In the attempt to use abduction to simulate more general default rules, however, Poole needs to use integrity constraints. The new theory $\mathcal{F} \cup \Delta$ should be consistent with these constraints. Default rules of the form:

$$\frac{\alpha(X) : \mathrm{M}\beta_1(X), \ldots, \mathrm{M}\beta_n(X)}{\gamma(X)}$$

are translated into "facts", which are implications

$$\alpha(X) \wedge M_{\beta_1}(X) \wedge \ldots \wedge M_{\beta_n}(X) \rightarrow \gamma(X)$$

where M_{β_i} is a new predicate, and $M_{\beta_i}(X)$ is a default formula (abducible), for all $i = 1, \ldots, n$. Integrity constraints

$$\neg \beta_i(X) \rightarrow \neg M_{\beta_i}(X)$$

are needed to link the new predicates M_{β_i} appropriately with the predicates β_i, for all $i = 1, \ldots, n$. A further integrity constraint

$$\neg \gamma(X) \rightarrow \neg M_{\beta_i}(X),$$

for any $i = 1, \ldots, n$, is needed to prevent the application of the contrapositive

$$\neg \gamma(X) \wedge M_{\beta_1}(X) \wedge \ldots \wedge M_{\beta_n}(X) \rightarrow \neg \alpha(X)$$

of the implication, in the attempt to make the implication behave like an inference rule. This use of integrity constraints is different from their intended use in abductive frameworks as presented in section 1.2.

Poole's attempted simulation of Reiter's general default rules is not exact. He presents a number of examples where the two formulations differ and argues that Reiter's default logic gives counterintuitive results. In fact, many of these examples can be dealt with correctly in certain extensions of default logic, such as Cumulative Default Logic [121], and it is possible to dispute some of the other examples. But, more importantly, there are still other examples where the Theorist approach arguably gives the wrong result. The most important of these is the now notorious Yale shooting problem of [73, 74]. This can be reduced to the propositional logic program

$$\begin{aligned} & alive\text{-}after\text{-}load\text{-}wait\text{-}shoot \leftarrow alive\text{-}after\text{-}load\text{-}wait, \\ & \qquad\qquad \sim abnormal\text{-}alive\text{-}shoot \\[1ex] & loaded\text{-}after\text{-}load\text{-}wait \leftarrow loaded\text{-}after\text{-}load, \\ & \qquad\qquad \sim abnormal\text{-}loaded\text{-}wait \\ & abnormal\text{-}alive\text{-}shoot \leftarrow loaded\text{-}after\text{-}load\text{-}wait \\ & alive\text{-}after\text{-}load\text{-}wait \\ & loaded\text{-}after\text{-}load. \end{aligned}$$

As argued in [127], these clauses can be simplified further: First, the facts $alive\text{-}after\text{-}load\text{-}wait$ and $loaded\text{-}after\text{-}load$ can be eliminated by resolving them against the corresponding conditions of the first two clauses, giving

$$\begin{aligned} & alive\text{-}after\text{-}load\text{-}wait\text{-}shoot \leftarrow \sim abnormal\text{-}alive\text{-}shoot \\ & loaded\text{-}after\text{-}load\text{-}wait \leftarrow \sim abnormal\text{-}loaded\text{-}wait \\ & abnormal\text{-}alive\text{-}shoot \leftarrow loaded\text{-}after\text{-}load\text{-}wait \end{aligned}$$

Then the atom $loaded\text{-}after\text{-}load\text{-}wait$ can be resolved away from the second and third clauses leaving the two clauses

$alive\text{-}after\text{-}load\text{-}wait\text{-}shoot \leftarrow \sim\ abnormal\text{-}alive\text{-}shoot$
$abnormal\text{-}alive\text{-}shoot \leftarrow \sim\ abnormal\text{-}loaded\text{-}wait$

The resulting clauses have the form

$$p \leftarrow \sim q$$

$$q \leftarrow \sim r.$$

Hanks and McDermott showed, in effect, that the default theory, whose facts consist of

$$\neg q \rightarrow p$$

$$\neg r \rightarrow q$$

and whose defaults are the normal defaults

$$\frac{: \mathrm{M}\,\neg\, q}{\neg\, q} \qquad \frac{: \mathrm{M}\,\neg\, r}{\neg\, r}$$

has two extensions: one in which $\neg\, r$, and therefore q holds; and one in which $\neg\, q$, and therefore p holds. The second extension is intuitively incorrect under the intended interpretation. Hanks and Mc Dermott showed that many other approaches to default reasoning give similarly incorrect results. However, Morris [127] showed that the default theory which has no facts but contains the two non-normal defaults

$$\frac{: \mathrm{M}\,\neg\, q}{p} \qquad \frac{: \mathrm{M}\,\neg\, r}{q}$$

yields only one extension, containing q, which is the correct result. In contrast, all natural representations of the problem in Theorist give incorrect results.

As Eshghi and Kowalski [53], Evans [55] and Apt and Bezem [7] observe, the Yale shooting problem has the form of a logic program, and interpreting negation in the problem as negation as failure yields only the correct result. This is the case for both the semantics and the proof theory of LP. Moreover, [53] and [89] show how to retain the correct result when negation as failure is interpreted as a form of abduction.

On the other hand, the Theorist framework does overcome the problem that some default theories do not have extensions and hence cannot be given any meaning within Reiter's default logic. In the next section we will see that this problem also occurs in LP, but that it can also be overcome by an abductive treatment of negation as failure. We will also see that the resulting abductive interpretation of negation as failure allows us to regard LP as a hybrid which treats defaults as abducibles in Theorist but treats clauses as inference rules in default logic.

The inference rule interpretation of logic programs, makes LP extended with abduction especially suitable for default reasoning. Integrity constraints can be used, not for preventing application of contrapositives, but for representing negative information and exceptions to defaults.

Example 3.2
The default (1) in the flying-birds example 3.1 can be represented by the logic program

$$fly(X) \leftarrow bird(X),\ birds\text{-}fly(X),$$

with the abducible predicate $birds\text{-}fly(X)$. Note that this clause is equivalent to the "fact" (2) obtained by renaming the default (1) in Theorist. The exception can be represented by an integrity constraint:

$$\neg\, fly(X) \leftarrow penguin(X).$$

The resulting logic program, extended by means of abduction and integrity constraints, gives similar results to the Theorist formulation of example 3.1.

In sections 4, 5 and 6 we will see other ways of performing default reasoning in LP. In section 4 we will introduce negation as failure as a form of abductive reasoning. In section 5 we will discuss abductive logic programming with default and non-default abducibles and domain-specific integrity constraints. In section 6 we will consider an extended LP framework that contains clauses with negative conclusions and avoids the use of explicit integrity constraints in many cases. In section 7 we will present an abstract argumentation-based framework for default reasoning which unifies the treatment of abduction, default logic, LP and several other approaches to default reasoning.

4 Negation as Failure as Abduction

We noted in the previous section that default reasoning can be performed by means of abduction in LP by explicitly introducing abducibles into rules. Default reasoning can also be performed with the use of negation as failure (NAF) [24] in general logic programs. NAF provides a natural and powerful mechanism for performing non-monotonic and default reasoning. As we have already mentioned, it provides a simple solution to the Yale shooting problem. The abductive interpretation of NAF that we will present below provides further evidence for the suitability of abduction for default reasoning.

To see how NAF can be used for default reasoning, we return to the flying-birds example.

Example 4.1
The NAF formulation differs from the logic program with abduction presented in the last section (example 3.2) by employing a negative condition

$$\sim\ abnormal\text{-}bird(X)$$

instead of a positive abducible condition

$$birds\text{-}fly(X)$$

and by employing a positive conclusion

$$abnormal\text{-}bird(X)$$

in an ordinary program clause, instead of a negative conclusion

$$\neg fly(X)$$

in an integrity constraint. The two predicates *abnormal-bird* and *birds-fly* are opposite to one another. Thus in the NAF formulation the default is expressed by the clause

$$fly(X) \leftarrow bird(X), \sim abnormal\text{-}bird(X)$$

and the exception by the clause

$$abnormal\text{-}bird(X) \leftarrow penguin(X).$$

In this example, both the abductive formulation with an integrity constraint and the NAF formulation give the same result. We will see later in section 5.5 that there exists a systematic transformation which replaces positive abducibles by NAF and integrity constraints by ordinary clauses. This example can be regarded as an instance of that transformation.

4.1 Logic programs as abductive frameworks

The similarity between abduction and NAF can be used to give an abductive interpretation of NAF. This interpretation was presented in [53] and [54], where negative literals are interpreted as abductive hypotheses that can be assumed to hold provided that, together with the program, they satisfy a canonical set of integrity constraints. A general logic program P is thereby transformed into an abductive framework $\langle P^*, A^*, I^*\rangle$ (see section 1) in the following way.

- A new predicate symbol p^* (the opposite of p) is introduced for each p in P, and A^* is the set of all these predicates.
- P^* is P where each negative literal $\sim p(t)$ has been replaced by $p^*(t)$.
- I^* is a set of all integrity constraints of the form [6]:

$$\forall X \neg [p(X) \wedge p^*(X)] \text{ and}$$
$$\forall X [p(X) \vee p^*(X)].$$

[6]In the original paper the disjunctive integrity constraints were written in the form

$$\mathrm{Demo}(P^* \cup \Delta, p(\mathrm{t})) \vee \mathrm{Demo}(P^* \cup \Delta, p^*(t)),$$

where t is any variable-free term. This formulation makes explicit a particular (meta-level) interpretation of the disjunctive integrity constraint. The simpler form

$$\forall X [p(X) \vee p^*(X)]$$

is neutral with respect to the interpretation of integrity constraints and allows the meta-level interpretation as a special case.

The semantics of the abductive framework $\langle P^*, A^*, I^*\rangle$, in terms of **extensions** [7] $P^* \cup \Delta$ of P^*, where $\Delta \subseteq A^*$, gives a semantics for the original program P. A conclusion Q holds with respect to P if and only if the query Q^*, obtained by replacing each negative literal $\sim p(t)$ in Q by $p^*(t)$, has an abductive explanation in the framework $\langle P^*, A^*, I^*\rangle$. This transformation of P into $\langle P^*, A^*, I^*\rangle$ is an example of the method, described at the end of section 1.1, of giving a semantics to a language by translating it into another language whose semantics is already known.

The integrity constraints in I^* play a crucial role in capturing the meaning of NAF. The denials express that the newly introduced symbols p^* are the negations of the corresponding p. They prevent an assumption $p^*(t)$ if $p(t)$ holds. On the other hand the disjunctive integrity constraints force a hypothesis $p^*(t)$ whenever $p(t)$ does not hold.

Hence we define the meaning of the integrity constraints I^* as follows: An extension $P^* \cup \Delta$ (which is a Horn theory) of P^* **satisfies** I^* if and only if for every variable-free atom p,

$$P^* \cup \Delta \not\models p \wedge p^*, \text{ and}$$
$$P^* \cup \Delta \models p \quad \text{or} \quad P^* \cup \Delta \models p^*.$$

Eshghi and Kowalski [54] show that there is a one to one correspondence between stable models [68] of P and abductive extensions of P^*. We recall the definition of stable model:

Let P be a general logic program, and assume that all the clauses in P are variable-free [8]. For any set M of variable-free atoms, let P_M be the Horn program obtained by deleting from P:

i) each rule that contains a negative literal $\sim A$, with $A \in M$,

ii) all negative literals in the remaining rules.

If the minimal (Herbrand) model of P_M coincides with M, then M is a **stable model** for P.

The correspondence between the stable model semantics of a program P and abductive extensions of P^* is given by:

- For any stable model M of P, the extension $P^* \cup \Delta$ satisfies I^*, where $\Delta = \{p^* \mid p \text{ is a variable-free atom}, p \notin M\}$.
- For any Δ such that $P^* \cup \Delta$ satisfies I^*, there is a stable model M of P, where $M = \{p \mid p \text{ is a variable-free atom}, p^* \notin \Delta\}$.

Notice that the disjunctive integrity constraints in the abductive framework correspond to a totality requirement that every atom must be either true or false in the stable model

[7]This use of the term "extension" is different from other uses. For example, in default logic an extension is formally defined to be the deductive closure of a theory "extended" by means of the conclusions of default rules. In this paper we also use the term "extension" informally (as in example 3.1) to refer to Δ alone.

[8]If P is not variable-free, then it is replaced by the set of all its variable-free instances.

semantics. Several authors have argued that this totality requirement is too strong, because it prevents us from giving a semantics to some programs, for example $p \leftarrow \sim p$. We would like to be able to assign a semantics to every program in order to have modularity, as otherwise one part of the program can affect the meaning of another unrelated part. We will see below that the disjunctive integrity constraint also causes problems for the implementation of the abductive framework for NAF.

Notice that the semantics of NAF in terms of abductive extensions is syntactic rather than model-theoretic. It is a semantics in the sense that it is a non-constructive specification. Similarly, the stable model semantics, as is clear from its correspondence with abductive extensions, is a semantics in the sense that it is a non-constructive specification of what should be computed. The computation itself is performed by means of a proof procedure.

4.2 An abductive proof procedure for LP

In addition to having a clear and simple semantics for abduction, it is also important to have an effective method for computing abductive explanations. Any such method will be very useful in practice in view of the many diverse applications of abductive reasoning, including default reasoning. The Theorist framework of [145, 150] provides such an implementation of abduction by means of a resolution based proof procedure.

In their study of NAF through abduction Eshghi and Kowalski [54] have defined an abductive proof procedure for NAF in logic programming. We will describe this procedure in some detail as it also serves as the basis for computing abductive explanations more generally within logic programming with other abducibles and integrity constraints (see section 5). In this section we will refer to the version of the abductive proof procedure presented in [39]. [9]

The abductive proof procedure interleaves two types of computation. The first type, referred to as the **abductive phase**, is standard SLD- resolution, which generates (negative) hypotheses and adds them to the set of abducibles being generated, while the second type, referred to as the **consistency phase** [10], incrementally checks that the hypotheses satisfy the integrity constraints I^* for NAF. Integrity checking of a hypothesis $p^*(t)$ reasons forward one step using a denial integrity constraint to derive the new denial $\neg\, p(t)$, which is then interpreted as the goal $\leftarrow\ p(t)$. Thereafter it reasons backward in SLD-fashion in all possible ways. Integrity checking succeeds if all the branches of the resulting search space fail finitely, in other words, if the contrary of $p^*(t)$, namely $p(t)$, finitely fails to hold. Whenever the potential failure of a branch of the consistency phase search space is due to the failure of a selected abducible, say $q^*(s)$, a new abductive phase of SLD-resolution is triggered for the goal $\leftarrow\ q(s)$, to ensure that the disjunctive integrity constraint $q^*(s) \vee q(s)$ is not violated by the failure of both $q^*(s)$ and $q(s)$. This attempt to show $q(s)$ can require in turn the addition of further abductive assumptions to the set of hypotheses which is being generated.

[9] As noticed by Dung [39], the procedure presented in [54] contains a mistake, which is not present, however, in the earlier unpublished version of the paper.

[10] We use the term "consistency phase" for historical reasons. However, in view of the precise definition of integrity constraint satisfaction, some other term might be more appropriate.

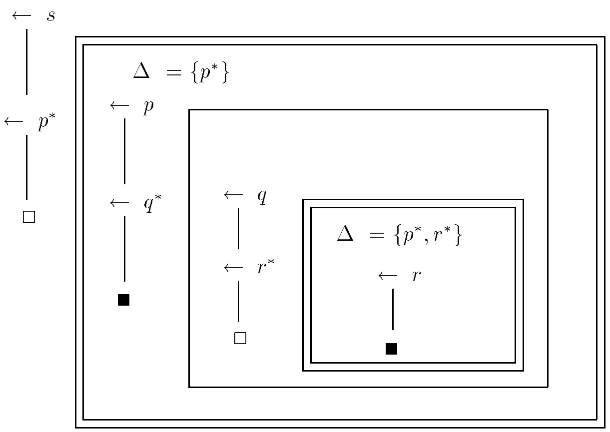

Figure 1: computation for example 4.2

To illustrate the procedure consider the following logic program, which is a minor elaboration of the propositional form of the Yale shooting problem discussed in section 3.

Example 4.2

$$
\begin{array}{lcl}
s & \leftarrow & \sim p \\
p & \leftarrow & \sim q \\
q & \leftarrow & \sim r
\end{array}
$$

The query $\leftarrow\ s$ succeeds with answer $\Delta = \{p^*,\, r^*\}$. The computation is shown in figure 1. Parts of the search space enclosed by a double box show the incremental integrity checking of the latest abducible added to the explanation Δ. For example, the outer double box shows the integrity check for the abducible p^*. For this we start from $\leftarrow\ p \equiv \neg p$ (resulting from the resolution of p^* with the integrity constraint $\neg\,(p \wedge p^*) \equiv \neg\, p \vee \neg\, p^*$) and resolve backwards in SLD-fashion to show that all branches end in failure, depicted here by a black box. During this consistency phase for p^* a new abductive phase (shown in the single box) is generated when q^* is selected since the disjunctive integrity constraint $q^* \vee q$ implies that failure of q^* is only allowed provided that q is provable. The SLD proof of q requires the addition of r^* to Δ, which in turn generates a new consistency phase for r^* shown in the inner double box. The goal $\leftarrow\ r$ fails trivially because there are no rules for r and so r^* and the enlarged explanation $\Delta = \{p^*,\, r^*\}$ satisfy the integrity constraints. Tracing the computation backwards, we see that q holds, therefore q^* fails and, therefore p^* satisfies the integrity constraints and the original query $\leftarrow\ s$ succeeds.

In general, an abductive phase succeeds if and only if one of its branches ends in a white box (indicating that no subgoals remain to be solved). It fails finitely if and only if all branches end in a black box (indicating that some subgoal cannot be solved). A consistency phase fails if and only if one of its branches ends in a white box (indicating that integrity has been violated). It succeeds finitely if and only if all branches end in a black box (indicating that integrity has not been violated).

It is instructive to compare the computation space of the abductive proof procedure with that of SLDNF. It is easy to see that these are closely related. In particular, in both cases negative atoms need to be variable-free before they are selected. On the other hand, the two proof procedures have some important differences. A successful derivation of the abductive proof procedure will produce, together with the usual answer obtained from SLDNF, additional information, namely the abductive explanation Δ. This additional information can be useful in different ways, in particular to avoid recomputation of negative subgoals. More importantly, as the next example will show, this information will allow the procedure to handle non-stratified programs and queries for which SLDNF is incomplete. In this way the abductive proof procedure generalises SLDNF. Furthermore, the abductive explanation Δ produced by the procedure can be recorded and used in any subsequent revision of the beliefs held by the program, in a similar fashion to truth maintenance systems [94]. In fact, this abductive treatment of NAF allows us to identify a close connection between logic programming and truth maintenance systems in general (see section 8). Another important difference is the distinction that the abductive proof procedure for NAF makes between the abductive and consistency phases. This allows a natural extension of the procedure to a more general framework where we have other hypotheses and integrity constraints in addition to those for NAF [91, 92, 93] (see section 5.2).

To see how the abductive proof procedure extends SLDNF, consider the following program.

Example 4.3

$$\begin{array}{lcl} s & \leftarrow & q \\ s & \leftarrow & p \\ p & \leftarrow & \sim q \\ q & \leftarrow & \sim p \end{array}$$

The last two clauses in this program give rise to a **two-step loop via NAF**, in the sense that p (and, similarly, q) "depends" negatively on itself through two applications of NAF. This causes the SLDNF proof procedure, executing the query $\leftarrow s$, to go into an infinite loop. Therefore, the query has no SLDNF refutation. However, in the corresponding abductive framework the query has two answers, $\Delta = \{p^*\}$ and $\Delta = \{q^*\}$, corresponding to the two stable models of the program. The computation for the first answer is shown in figure 2. The outer abductive phase generates the hypothesis p^* and triggers the consistency phase for p^* shown in the double box. In general, whenever a hypothesis is tested for integrity, we can add the hypothesis to Δ either at the beginning or at the end of the consistency phase. When this addition is done at the beginning (as originally defined in [54]) this extra information can be used in any subordinate abductive phase. In this example, the hypothesis p^* is used in the subordinate abductive proof of q to justify the failure of q^* and consequently to render p^* acceptable. In other words, the acceptability

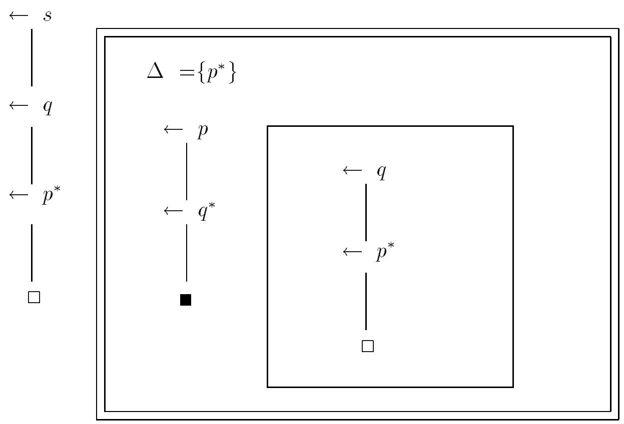

Figure 2: computation for example 4.3

of p^* as a hypothesis is proved under the assumption of p^*. The same abductive proof procedure, but where each new hypothesis is added to Δ only at the successful completion of its consistency phase, provides a sound proof procedure for the well-founded semantics [187].

Example 4.4
Consider the query $\leftarrow p$ with respect to the abductive framework corresponding to the following program

$$\begin{array}{lcl} r & \leftarrow & \sim r \\ r & \leftarrow & q \\ p & \leftarrow & \sim q \\ q & \leftarrow & \sim p. \end{array}$$

Note that the first clause of this program give rise to a **one-step loop via NAF**, in the sense that r "depends" negatively on itself through one application of NAF. The abductive proof procedure succeeds with the explanation $\{q^*\}$, but the only set of hypotheses which satisfies the integrity constraints is $\{p^*\}$.

So, as Eshghi and Kowalski [54] show by means of this example, the abductive proof procedure is not always sound with respect to the above abductive semantics of NAF. In fact, following the result in [39], it can be proved that the proof procedure is sound for the class of order-consistent logic programs defined by Sato [168]. Intuitively, this is the class of programs which do not contain clauses giving rise to odd-step loops via NAF.

For the overall class of general logic programs, moreover, it is possible to argue that it is the semantics and not the proof procedure that is at fault. Indeed, Saccà and Zaniolo [164], Przymusinski [153] and others have argued that the totality requirement of stable models is too strong. They relax this requirement and consider partial or three-valued stable models instead. In the context of the abductive semantics of NAF this is an argument against the disjunctive integrity constraints.

An abductive semantics of NAF without disjunctive integrity constraints has been proposed by Dung [39] (see section 4.3 below). The abductive proof procedure is sound with respect to this improved semantics.

An alternative abductive semantics of NAF without disjunctive integrity constraints has been proposed by Brewka [14], following ideas presented in [104]. He suggests that the set which includes both accepted and refuted NAF hypotheses be maximised. For each such set of hypotheses, the logic program admits a "model" which is the union of the sets of accepted hypotheses together with the "complement" of the refuted hypotheses. For example 4.4 the only "model" is $\{p^*, q, r\}$. Therefore, the abductive proof procedure is still unsound with respect to this semantics. Moreover, this semantics has other undesirable consequences. For example, the program

$$p \leftarrow \sim p, \sim q$$

admits both $\{\sim q\}$ and $\{\sim p\}$ as "models", while the only intuitively correct "model" is $\{\sim q\}$.

An alternative three-valued semantics for NAF has been proposed by Giordano, Martelli and Sapino [72]. According to their semantics, given the program

$$p \leftarrow p$$

p and p^* are both undefined. In contrast, p^* holds in the semantics of [39], as well as in the stable model [68] and well-founded semantics [187]. Giordano, Martelli and Sapino [72] modify the abductive proof procedure so that the modification is sound and complete with respect to their semantics.

Satoh and Iwayama [171], on the other hand, show how to extend the abductive proof procedure of [54] to deal correctly with the stable model semantics. Their extension modifies the integrity checking method of [165] and deals more generally with arbitrary integrity constraints expressed in the form of denials.

Casamayor and Decker [20] also develop an abductive proof procedure for NAF. Their proposal combines features of the Eshghi-Kowalski procedure with ancestor resolution.

Finally, we note that, to show that $\sim p$ holds for programs such as $p \leftarrow p$, it is possible to define a non-effective extension of the proof procedure, that allows infinite failure in the consistency phases.

4.3 An argumentation-theoretic interpretation

Dung [39] replaces the disjunctive integrity constraints by a weaker requirement similar to the requirement that that the set of negative hypotheses Δ be a maximally consistent

set. Unfortunately, simply replacing the disjunctive integrity constraints by maximality does not work, as shown in the following example.

Example 4.5
With this change the program

$$p \leftarrow \sim q$$

has two maximally consistent extensions $\Delta_1 = \{p^*\}$ and $\Delta_2 = \{q^*\}$. However, only the second extension is computed both by SLDNF and by the abductive proof procedure. Moreover, for the same reason as in the case of the propositional Yale shooting problem discussed before, only the second extension is intuitively correct.

To avoid such problems Dung's notion of maximality is a more subtle. He associates with every logic program P an abductive framework $\langle P^*, A^*, I^* \rangle$ where I^* contains only denials

$$\forall X \neg [p(X) \wedge p^*(X)]$$

as integrity constraints. Then, given sets Δ, E of (negative) hypotheses, i.e. $\Delta \subseteq A^*$ and $E \subseteq A^*$, E can be said to **attack** Δ (relative to $\langle P^*, A^*, I^* \rangle$) if $P^* \cup E \vdash p$ for some $p^* \in \Delta$. [11] Dung calls an extension $P^* \cup \Delta$ of P^* **preferred** if

- $P^* \cup \Delta$ is consistent with I^* and
- Δ is maximal with respect to the property that for every attack E against Δ, Δ attacks E (i.e. Δ "counterattacks" E or "defends" itself against E).

Thus a preferred extension can be thought of as a maximally consistent set of hypotheses that contains its own defence against all attacks. In [39] a consistent set of hypotheses Δ (not necessarily maximal) satisfying the property of containing its own defence against all attacks is said to be **admissible** (to P^*). In fact, Dung's definition is not formulated explicitly in terms of the notions of attack and defence, but is equivalent to the one just presented.

Preferred extensions solve the problem with disjunctive integrity constraints in example 4.4 and with maximal consistency semantics in example 4.5. In example 4.4 the preferred extension semantics sanctions the derivation of p by means of an abductive derivation with generated hypotheses $\{q^*\}$. In fact, Dung proves that the abductive proof procedure is sound with respect to the preferred extension semantics. In example 4.5 the definition of preferred extension excludes the maximally consistent extension $\{p^*\}$, because there is no defence against the attack q^*.

The preferred extension semantics provides a unifying framework for various approaches to the semantics of negation in LP. Kakas and Mancarella [95] show that it is equivalent to Saccà and Zaniolo's partial stable model semantics [164]. Like the partial stable model semantics, it includes the stable model semantics as a special case.

Dung [39] also defines the notion of **complete** extension. An extension $P^* \cup \Delta$ is complete if

[11] Alternatively, instead of the symbol $\models$ we could use the symbol $\vdash$, here and elsewhere in the paper where we define the notion of "attack".

- $P^* \cup \Delta$ is consistent with I^* and
- $\Delta = \{p^* \mid \text{for each attack } E \text{ against } \{p^*\},\ \Delta \text{ attacks } E\}$
  (i.e. Δ is admissible and it contains all hypotheses it can defend against all attacks).

Stationary expansions [154] are equivalent to complete extensions, as shown in [16]. Moreover, Dung shows that the **well-founded model** [187] is the smallest complete extension that can be constructed bottom-up from the empty set of negative hypotheses, by adding incrementally all admissible hypotheses. Thus the well-founded semantics is minimalist and sceptical, whereas the preferred extension semantics is maximalist and credulous. The relationship between these two semantics is further investigated in [47], where the well-founded model and preferred extensions are shown to correspond to the least fixed point and greatest fixed point, respectively, of the same operator.

Kakas and Mancarella [96, 97] propose an improvement of the preferred extension semantics. Their proposal can be illustrated by the following example.

Example 4.6
Consider the program

$$\begin{array}{lcl} p & \leftarrow & \sim q \\ q & \leftarrow & \sim q. \end{array}$$

Similarly to example 4.4, the last clause gives rise to a one-step loop via NAF, since q "depends" negatively on itself through one application of NAF. In the abductive framework corresponding to this program consider the set of hypotheses $\Delta = \{p^*\}$. The only attack against Δ is $E = \{q^*\}$, and the only attack against E is E itself. Thus Δ is not an admissible extension of the program according to the preferred extension semantics, because Δ cannot defend itself against E. The empty set is the only preferred extension. However, intuitively Δ should be admissible because the only attack E against Δ attacks itself, and therefore should not be regarded as an admissible attack against Δ.

To deal with this kind of example, Kakas and Mancarella [96, 97] modify Dung's semantics, increasing the number of ways in which an attack E can be defeated. Whereas Dung only allows Δ to defeat an attack E, they also allow E to defeat itself. They call a set of hypotheses Δ **weakly stable** if

- for every attack E against Δ, $E \cup \Delta$ attacks $E - \Delta$.

Moreover, they call an extension $P^* \cup \Delta$ of P^* a **stable theory** if Δ is maximally weakly stable. Note that here the condition "$P^* \cup \Delta$ is consistent with I^*" of the definition of preferred extensions and admissible sets of hypotheses is subsumed by the new condition. This is a consequence of another difference between [96, 97] and [39], namely that for each attack E against Δ the counter-attack is required to be against $E - \Delta$ rather than against E. In other words, the defence of Δ must be a genuine attack that does not at the same time also attack Δ. Therefore, if Δ is inconsistent, it contains as a subset an attack E, which can not be counterattacked because $E - \Delta$ is empty. In [97], Kakas and Mancarella show how these notions can also be used to extend the sceptical well-founded model semantics. In example 4.6 above this extension of the well-founded model will contain the negation of p.

Like the original definition of admissible sets of hypotheses and preferred extension, the definition of weakly stable sets of hypotheses and stable theories was not originally formulated in terms of attack, but is equivalent to the one presented here.

Kakas and Mancarella [97] argue that the notion of defeating an attack needs to be liberalised further. They illustrate their argument with the following example.

Example 4.7
Consider the program P

$$\begin{array}{lcl} s & \leftarrow & \sim p \\ p & \leftarrow & \sim q \\ q & \leftarrow & \sim r \\ r & \leftarrow & \sim p. \end{array}$$

The last three clauses give rise to a three-step loop via NAF, since p (and, similarly, q and r) "depends" negatively on itself through three application of NAF. In the corresponding abductive framework, the only attack against the hypothesis s^* is $E = \{p^*\}$. But although $P^* \cup \{s^*\} \cup E$ does not attack E, E is not a valid attack because it is not stable (or admissible) according to the definition above.

To generalise the reasoning in this example so that it gives an intuitively correct semantics to any program with clauses giving rise to an odd-step loop via NAF, we need to liberalise further the conditions for defeating E. Kakas and Mancarella suggest a recursive definition in which a set of hypotheses is deemed acceptable if no attack against it is acceptable. More precisely, given an initial set of hypotheses Δ_0, a set of hypotheses Δ is **acceptable** to Δ_0 iff

$$\text{for every attack } E \text{ against } \Delta - \Delta_0,\ E \text{ is not acceptable to } \Delta \cup \Delta_0.$$

The semantics of a program P can be identified with any Δ which is maximally acceptable to the empty set of hypotheses $\emptyset$. As before with weak stability and stable theories, the consideration of attacks only against $\Delta - \Delta_0$ ensures that attacks and counterattacks are genuine, i.e. they attack the new part of Δ that does not contain Δ_0.

Notice that, as a special case, we obtain a basis for the definition:

$$\Delta \text{ is acceptable to } \Delta_0 \text{ if } \Delta \subseteq \Delta_0.$$

Therefore, if Δ is acceptable to $\emptyset$ then Δ is consistent.

Notice, too, that applying the recursive definition twice, and starting with the base case, we obtain an approximation to the recursive definition

$$\begin{array}{c} \Delta \text{ is acceptable to } \Delta_0 \text{ if for every attack } E \text{ against } \Delta - \Delta_0, \\ E \cup \Delta \cup \Delta_0 \text{ attacks } E - (\Delta \cup \Delta_0). \end{array}$$

Thus, the stable theories are those which are maximally acceptable to $\emptyset$, where acceptability is defined by this approximation to the recursive definition.

A related argumentation-theoretic interpretation for the semantics of NAF in LP has also been developed by Geffner [67]. This interpretation is equivalent to the well-founded semantics [43]. Based upon Geffner's notion of argumentation, Torres [185] has proposed an argumentation-theoretic semantics for NAF that is equivalent to Kakas and Mancarella's stable theory semantics [96, 97], but is formulated in terms of the following notion of attack: E attacks Δ (relative to P^*) if $P^* \cup E \cup \Delta \vdash p$ for some $p^* \in \Delta$.

Alferes and Pereira [4] apply the argumentation-theoretic interpretation introduced in [88] to expand the well-founded model of normal and extended logic programs (see section 5). In the case of normal logic programming, their semantics gives the same result as the acceptability semantics in example 4.7.

Simari and Loui [177] define an argumentation-theoretic framework for default reasoning in general. They combine a notion of acceptability with Poole's notion of "most specific" explanation [143], to deal with hierarchies of defaults.

In section 7 we will present an abstract argumentation-theoretic framework which is based upon the framework for LP but unifies many other approaches to default reasoning.

4.4 An argumentation-theoretic interpretation of the abductive proof procedure

As mentioned above, the incorrectness (with respect to the stable model semantics) of the abductive proof procedure can be remedied by adopting the preferred extension, stable theory or acceptability semantics. This reinterpretation of the original abductive proof procedure in terms of an improved semantics, and the extension of the proof procedure to capture further improvements in the semantics, is an interesting example of the interaction that can arise between a program (proof procedure in this case) and its specification (semantics).

To illustrate the argumentation-theoretic interpretation of the proof procedure, consider again figure 1 of example 4.2. The consistency phase for p^*, shown in the outer-most double box, can be understood as searching for any attack against $\{p^*\}$. The only attack, namely $\{q^*\}$, is counterattacked (thereby defending $\{p^*\}$) by assuming the additional hypothesis r^*, as this implies q. Hence the set $\Delta = \{p^*, r^*\}$ is admissible, i.e. it can defend itself against any attack, since all attacks against $\{p^*\}$ are counterattacked by $\{r^*\}$ and there are no attacks against $\{r^*\}$.

In general, the proof procedure constructs an admissible set of negative hypotheses in two steps. First, it constructs a set of hypotheses which is sufficient to solve the original goal. Then, it augments this set with the hypotheses necessary to defend the first set against attack.

The argumentation-theoretic interpretation suggests how to extend the proof procedure to capture more fully the stable theory semantics and more generally the semantics given by the recursive definition for acceptability. The extension, presented in [182], involves temporarily remembering a (selected) attack E and using E itself together with the subset

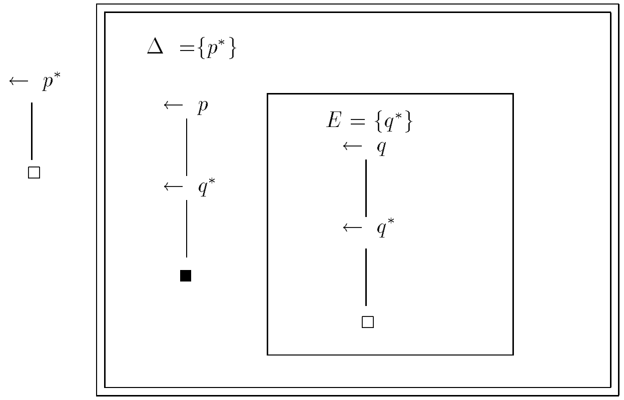

Figure 3: computation for example 4.6 with respect to the revisited proof procedure

of Δ generated so far, to counterattack E, in the subordinate abductive phase.

For example 4.6 of section 4.3, as shown in figure 3, to defend against the attack q^* on p^*, we need to temporarily remember q^* and use it in the subordinate abductive phase to prove q and therefore to attack q^* itself.

In the original abductive proof procedure of [54], hypotheses in defences are always added to Δ. However, in the proof procedure for the acceptability semantics defences D can not always be added to Δ, because even though D might be acceptable to Δ, $\Delta \cup D$ might not be acceptable to $\emptyset$. This situation arises for the three step loop program of example 4.7, where $D = \{q^*\}$ is used to defend $\Delta = \{s^*\}$ against the attack $E = \{p^*\}$, but $\Delta \cup D$ is not acceptable to $\emptyset$.

To cater for this characteristic of the acceptability semantics, the extended proof procedure non-deterministically considers two cases. For each hypothesis in a defence D against an attack E against Δ, the hypothesis either can be added to Δ or can be remembered temporarily to counterattack any attack E' against D, together with Δ and E. In general, a sequence of consecutive attacks and defences E, D, E', D', ... can be generated before an acceptable abductive explanation Δ is found, and the same non-deterministic consideration of cases is applied to D' and all successive defences in the sequence.

The definitions of admissible, stable and acceptable sets Δ of hypotheses all require that every attack against Δ be counterattacked. Although every superset of an attack is also an attack, the abductive proof procedure in [54] only considers those "minimal" attacks generated by SLD, [12] without examining superset attacks. This is possible because all supersets of an attack can be counterattacked in exactly the same way as the attack itself, which is generated by SLD. For this reason, the proof procedure of [54] is sound for the admissibility semantics. Unfortunately, supersets of attacks need to be considered to guarantee soundness of the proof procedure for the acceptability semantics. In [182], however, Toni and Kakas prove that only certain supersets of "minimally generated" attacks need to be considered.

The additional features required for the proof procedure to capture more fully the acceptability semantics render the proof procedure considerably more complex and less efficient than proof procedures for simpler semantics. However, this extra complexity is due to the treatment of any odd-step loops via NAF and such programs seem to occur very rarely in practice. Therefore, in most cases it is sufficient to consider the approximation of the proof procedure which computes the preferred extension and stable theory semantics. This approximation improves upon the Eshghi-Kowalski proof procedure, since in the case of finite failure it terminates earlier, avoiding unnecessary computation.

5 Abductive Logic Programming

Abductive Logic Programming (ALP), as understood in the remainder of this paper, is the extension of LP to support abduction in general, and not only the use of abduction for NAF. This extension was introduced already in section 1, as the special case of an abductive framework $\langle T, A, I\rangle$, where T is a logic program. In this paper we will assume, without loss of generality, that abducible predicates do not have definitions in T, i.e. do not appear in the heads of clauses in the program T [13]. This assumption has the advantage that all explanations are thereby guaranteed to be basic.

Semantics and proof procedures for ALP have been proposed by Eshghi and Kowalski [53], Kakas and Mancarella [90] and Chen and Warren [23]. Chen and Warren extend the perfect model semantics of Przymusinski [152] to include abducibles and integrity constraints over abducibles. Here we shall concentrate on the proposal of Kakas and Mancarella, which extends the stable model semantics.

5.1 Generalised stable model semantics

Kakas and Mancarella [90] develop a semantics for ALP by generalising the stable model semantics for LP. Let $\langle P, A, I\rangle$ be an abductive framework, where P is a general logic program, and let Δ be a subset of A. $M(\Delta)$ is a **generalised stable model** of $\langle P, A, I\rangle$ iff

- $M(\Delta)$ is a stable model of $P \cup \Delta$, and
- $M(\Delta) \models I$.

Here the semantics of the integrity constraints I is defined by the second condition in the definition above. Consequently, an abductive extension $P \cup \Delta$ of the program P **satisfies** I if and only if there exists a stable model $M(\Delta)$ of $P \cup \Delta$ such that I is true in $M(\Delta)$.

Note that in a similar manner, it is possible to generalise other model-theoretic semantics for logic programs, by considering only those models of $P \cup \Delta$ (of the appropriate kind, e.g. partial stable models, well-founded models etc.) in which the integrity constraints are all true.

Generalised stable models are defined independently from any query. However, given a query Q, we can define an abductive explanation for Q in $\langle P, A, I\rangle$ to be any subset Δ of A such that

- $M(\Delta)$ is a generalised stable model of $\langle P, A, I\rangle$, and
- $M(\Delta) \models Q$.

Example 5.1
Consider the program P

$$p \leftarrow a$$

$$q \leftarrow b$$

with $A = \{a, b\}$ and integrity constraint I

$$p \leftarrow q.$$

The interpretations $M(\Delta_1) = \{a, p\}$ and $M(\Delta_2) = \{a, b, p, q\}$ are generalised stable models of $\langle P, A, I\rangle$. Consequently, both $\Delta_1 = \{a\}$ and $\Delta_2 = \{a, b\}$ are abductive explanations of p. On the other hand, the interpretation $\{b, q\}$, corresponding to the set of abducibles $\{b\}$, is not a generalised stable model of $\langle P, A, I\rangle$, because it is not a model of I as it does not contain p. Moreover, the interpretation $\{b, q, p\}$, although it is a model of $P \cup I$ and therefore satisfies I according to the consistency view of constraint satisfaction, is not a generalised stable model of $\langle P, A, I\rangle$, because it is not a stable model of P. This shows that the notion of integrity satisfaction for ALP is stronger than the consistency view. It is also possible to show that it is weaker than the theoremhood view and to argue that it is similar to the metalevel or epistemic view.

An alternative, and perhaps more fundamental way of understanding the generalised stable model semantics is by using abduction both for hypothetical reasoning and for NAF. The negative literals in $\langle P, A, I\rangle$ can be viewed as further abducibles according to the transformation described in section 4. The set of abducible predicates then becomes $A \cup A^*$, where A^* is the set of negative abducibles introduced by the transformation. This results in a new abductive framework $\langle P^*, A \cup A^*, I \cup I^*\rangle$, where I^* is the set of special

[12] As illustrated in section 1, these attacks are genuinely minimal unless the logic program encodes non-minimal explanations.

[13] In the case in which abducibile predicates have definitions in T, auxiliary predicates can be introduced in such a way that the resulting program has no definitions for the abducible predicates. This can be done by means of a transformation similar to the one used to separate extensional and intensional predicates in deductive databases [124]. For example, for each abducible predicate $a(X)$ in T we can introduce a new predicate $\delta_a(X)$ and add the clause

$$a(X) \leftarrow \delta_a(X).$$

The predicate $a(X)$ is no longer abducible, whereas $\delta_a(X)$ is now abducible.

integrity constraints introduced by the transformation of section 4 [14]. The semantics of the abductive framework $\langle P^*, A \cup A^*, I \cup I^* \rangle$ can then be given by the sets Δ^* of hypotheses drawn from $A \cup A^*$ which satisfy the integrity constraints $I \cup I^*$.

Example 5.2
Consider P

$$\begin{array}{rcl} p & \leftarrow & a, \sim q \\ q & \leftarrow & b \end{array}$$

with $A = \{a, b\}$ and $I = \emptyset$. If Q is $\leftarrow p$ then $\Delta^* = \{a, q^*, b^*\}$ is an explanation for $Q^* = Q$ in $\langle P^*, A \cup A^*, I^* \rangle$. Note that b^* is in Δ^* because I^* contains the disjunctive integrity constraint $b \vee b^*$.

Kakas and Mancarella show a one to one correspondence between the generalised stable models of $\langle P, A, I \rangle$ and the sets of hypotheses Δ^* that satisfy the transformed framework $\langle P^*, A \cup A^*, I \cup I^* \rangle$. Moreover they show that for any abductive explanation Δ^* for a query Q in $\langle P^*, A \cup A^*, I \cup I^* \rangle$, $\Delta = \Delta^* \cap A$ is an abductive explanation for Q in $\langle P, A, I \rangle$.

Example 5.3
Consider the framework $\langle P, A, I \rangle$ and the query Q of example 5.2. We have already seen that $\Delta^* = \{a, q^*, b^*\}$ is an explanation for Q^* in $\langle P^*, A \cup A^*, I^* \rangle$. Accordingly the subset $\Delta = \{a\}$ is an explanation for Q in $\langle P, A, I \rangle$.

Note that the generalised stable model semantics as defined above requires that for each abducible a, either a or a^* holds. This can be relaxed by dropping the disjunctive integrity constraints $a \vee a^*$ and defining the set of abducible hypotheses A to include both a and a^*. Such a relaxation would be in the spirit of replacing stable model semantics by admissible or preferred extensions in the case of ordinary LP.

Generalised stable models combine the use of abduction for default reasoning (in the form of NAF) with the use of abduction for other forms of hypothetical reasoning. In the generalised stable model semantics, abduction for default reasoning is expressed solely by NAF. However, in the event calculus persistence axiom presented in section 2 the predicate *persists* is a positive abducible that has a default nature. Therefore, instances of *persists* should be abduced unless some integrity constraint is violated. Indeed, in standard formulations of the persistence axiom the positive atom $persists(T_1, P, T_2)$ is replaced by a negative literal $\sim clipped(T_1, P, T_2)$ [176, 35]. In contrast, the abduction of *happens* is used for non-default hypothetical reasoning. The distinction between default reasoning and non-default abduction is also made in Konolige's proposal [103], which combines abduction for non-default hypothetical reasoning with default logic [158] for default reasoning. This proposal is similar, therefore, to the way in which generalised stable models combine abduction with NAF. Poole [147], on the other hand, proposes an abductive framework where abducibles can be specified either as default, like *persists*, or

[14] Note that the transformation described in section 4 also needs to be applied to the set I of integrity constraints. For notational convenience, however, we continue to use the symbol I to represent the result of applying the transformation to I (otherwise we would need to use the symbol I^*, conflicting with the use of the symbol I^* for the special integrity constraints introduced in section 4).

non-default, like *happens*. In [183], Toni and Kowalski show how both default and non-default abducibles can be reduced to NAF. This reduction is discussed in section 5.5 below.

The knowledge representation problem in ALP is complicated by the need to decide whether information should be represented as part of the program, as an integrity constraint, or as an observation to be explained, as illustrated by the following example taken from [9].

Example 5.4

$$\begin{array}{l} fly(X) \leftarrow bird(X), \sim abnormal\_bird(X) \\ abnormal\_bird(X) \leftarrow penguin(X) \\ has\_beak(X) \leftarrow bird(X). \end{array}$$

Suppose that *bird* is abducible and consider the three cases in which

$$fly(tweety)$$

is either added to the program, added to the integrity constraints, or considered as the observation to be explained. In the first case, the abducible $bird(tweety)$ and, as a consequence, the atom $has\_beak(tweety)$ belong to some, but not all, generalised stable models. Instead, in the second case every generalised stable model contains $bird(tweety)$ and $has\_beak(tweety)$. In the last case, the observation is assimilated by adding the explanation $\{bird(tweety)\}$ to the program, and therefore $has\_beak(tweety)$ is derived in the resulting generalised stable model. Thus, the last two alternatives have similar effects. Denecker and DeSchreye [35] argue that the second alternative is especially appropriate for knowledge representation in the temporal reasoning domain.

5.2 An abductive proof procedure for ALP

In [91, 92, 93], a proof procedure is given to compute abductive explanations in ALP. This extends the abductive proof procedure for NAF [54] described in section 4.2, retaining the basic structure which interleaves an abductive phase that generates and collects abductive hypotheses with a consistency phase that incrementally checks these hypotheses for integrity. We will illustrate these extended proof procedure by means of examples.

Example 5.5
Consider again example 4.2. The abductive proof procedure for NAF fails on the query $\leftarrow p$. Ignoring, for the moment, the construction of the set Δ, the computation is that shown inside the outer double box of figure 1 with the abductive and consistency phases interchanged, i.e. the type of each box changed from a double box to a single box and vice-versa. Suppose now that we have the same program and query but in an ALP setting where the predicate r is abducible. The query will then succeed with the explanation $\Delta = \{q^*, r\}$ as shown in figure 4. As before the computation arrives at a point where r needs to be proved. Whereas this failed before, this succeeds now by abducing r. Hence by adding the hypothesis r to the explanation we can ensure that q^* is acceptable.

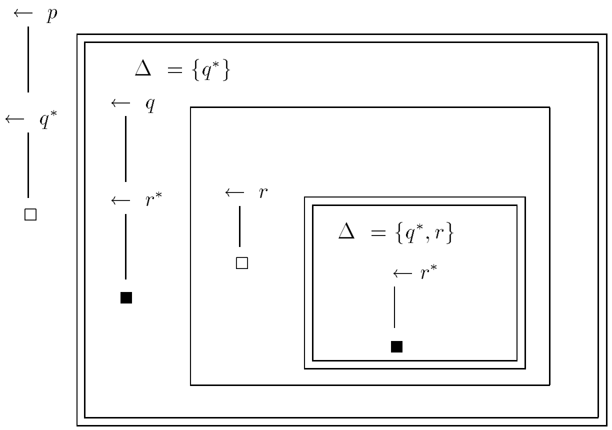

Figure 4: extended proof procedure for example 5.5

An important feature of the abductive proof procedures is that they avoid performing a full general-purpose integrity check (such as the forward reasoning procedure of [111]). In the case of a negative hypothesis, q^* for example, a general-purpose forward reasoning integrity check would have to use rules in the program such as $p \leftarrow q^*$ to derive p. The optimised integrity check in the abductive proof procedure avoids this inference and only reasons forward one step with the integrity constraint $\neg(q \wedge q^*)$, deriving the resolvent $\leftarrow q$, and then reasoning backward from the resolvent.

Similarly, the integrity check for a positive hypothesis, r for example, avoids reasoning forward with any rules which might have r in the body. Indeed, in a case, such as example 5.5 above, where there are no domain specific integrity constraints, the integrity check for a positive abducible, such as r, simply consists in checking that its complement, in our example r^*, does not belong to Δ.

To ensure that this optimised form of integrity check is correct, the proof procedure is extended to record those positive abducibles it needs to assume absent to show the integrity of other abducibles in Δ. So whenever a positive abducible, which is not in Δ, is selected in a branch of a consistency phase, the procedure fails on that branch and at the same time records that this abducible needs to be absent. This extension is illustrated by the following example.

Example 5.6
Consider the program

$$\begin{array}{lll} p & \leftarrow & \sim q, r \\ q & \leftarrow & r \end{array}$$

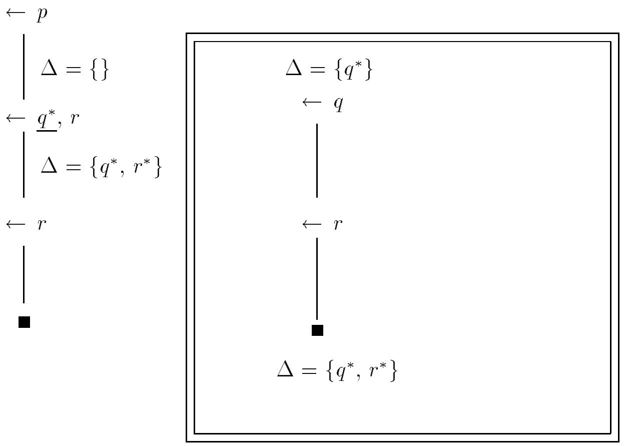

Figure 5: extended proof procedure for example 5.6

where r is abducible and the query is $\leftarrow p$ (see figure 5). The acceptability of q^* requires the absence of the abducible r. The simplest way to ensure this is by adding r^* to Δ. This, then, prevents the abduction of r and the computation fails. Notice that the proof procedure does not reason forward from r to test its integrity. This test has been performed backwards in the earlier consistency phase for q^*, and the addition of r^* to Δ ensures that it is not necessary to repeat it.

The way in which the absence of abducibles is recorded depends on how the negation of abducibles is interpreted. Under the stable and generalised stable model semantics, as we have assumed in example 5.6 above, the required failure of a positive abducible is recorded by adding its complement to Δ. However, in general it is not always appropriate to assume that the absence of an abducible implies its negation. On the contrary, it may be appropriate to treat abducibles as open rather than closed (see section 6.1), and correspondingly to treat the negation of abducible predicates as open. As we shall argue later, this might be done by treating such a negation as a form of explicit negation, which is also abducible. In this case recording the absence of a positive abducible by adding its complement to Δ is too strong, and we will use a separate (purely computational) data structure to hold this information.

Integrity checking can also be optimised when there are domain specific integrity constraints, provided the constraints can be formulated as denials [15] containing at least one literal whose predicate is abducible. In this case the abductive proof procedure needs

[15] Notice that any integrity constraint can be transformed into a denial (possibly with the introduction of new auxiliary predicates). For example:

$$p \leftarrow q \equiv \neg[q \wedge \neg p],$$

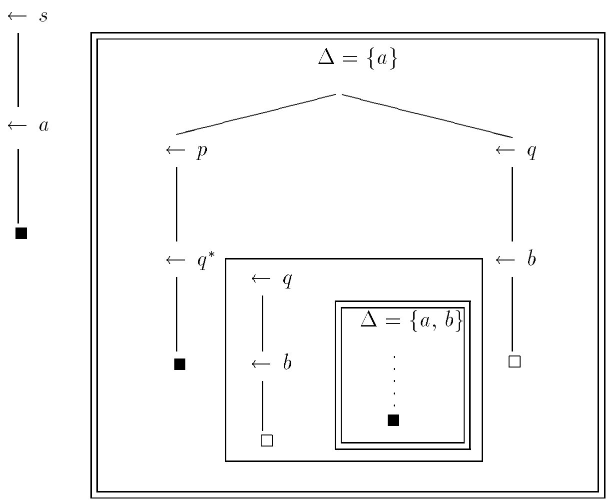

Figure 6: extended computation for example 5.7

only a minor extension [92, 93]: when a new hypothesis is added to Δ, the proof procedure resolves the hypothesis against any integrity constraint containing that hypothesis, and then reasons backward from the resolvent. To illustrate this extension consider the following example.

Example 5.7
Let the abductive framework be:

$$\begin{array}{llll} P: & s \leftarrow a & I: & \neg\,[a \,\wedge\, p] \\ & p \leftarrow\sim q & & \neg\,[a \,\wedge\, q] \\ & q \leftarrow b & & \end{array}$$

where a, b are abducible and the query is $\leftarrow\ s$ (see figure 6).

Assume that the integrity check for a is performed Prolog-style, by resolving first with the first integrity constraint and then with the second. The first integrity constraint requires the additional hypothesis b as shown in the inner-most single box. The integrity check for b is trivial, as b appears only in the integrity constraint $\neg[b \wedge b^*]$ in I^*, and the goal $\leftarrow b^*$ trivially fails, given $\Delta = \{a, b\}$ (inner-most double box). But $\Delta = \{a, b\}$ violates the integrity constraints, as can be seen by reasoning forward from b to q and then resolving with the second integrity constraint $\neg\,[a \,\wedge\, q]$. However, the proof procedure does not perform this forward reasoning and does not detect this violation of integrity at this stage.

$p \vee q \equiv \neg\,[\neg\, p \,\wedge\, \neg\, q]$.

Nevertheless the proof procedure is sound because the violation is found later by backward reasoning when a is resolved with the second integrity constraint.

In summary, the overall effect of additional integrity constraints is to increase the size of the search space during the consistency phase, with no significant change to the basic structure of the backward reasoning procedure.

Even if the absence of abducibles is not identified with the presence of their complement, the abductive proof procedure [91, 92, 93] described above suffers from the same soundness problem shown in section 4 for the abductive proof procedure for NAF. This problem can be solved similarly, by replacing stable models with any of the non-total semantics for NAF mentioned in section 4 (partial stable models, preferred extensions, stable theories or acceptability semantics). Replacing the stable models semantics by any of these semantics requires that the notion of integrity satisfaction be revised appropriately. This is an interesting problem for future work.

The soundness problem can also be addressed by providing an argumentation-theoretic semantics for ALP which treats integrity constraints and NAF uniformly via an appropriately extended notion of attack. In section 5.3 we will see that this alternative approach arises naturally from an argumentation-theoretic re-interpretation of the abductive proof procedure for ALP.

The proof procedure can be also modified to provide a sound computational mechanism for the generalised stable model semantics. This approach has been followed by Satoh and Iwayama [170], as we illustrate in section 5.4.

5.3 An argumentation-theoretic interpretation of the abductive proof procedure for ALP

Similarly to the LP case, the abductive proof procedure for ALP can be reinterpreted in argumentation-theoretic terms. For the ALP procedure, attacks can be provided as follows:

- via NAF:
  Relative to $\langle P^*, A \cup A^*, I \cup I^* \rangle$, E **attacks Δ via NAF** if

  E attacks Δ as in section 4.3, i.e. $P^* \cup E \vdash p$ for some $p^* \in \Delta$, or

  a^* is in E, for some abducible a in Δ;

- via integrity constraints:
  Relative to $\langle P^*, A \cup A^*, I \cup I^* \rangle$, E **attacks Δ via an integrity constraint** $\neg(L_1 \wedge \ldots \wedge L_n)$ in I if $P^* \cup E \vdash L_1, \ldots, L_{i-1}, L_{i+1}, \ldots, L_n$, for some L_i in Δ. [16]

To illustrate the argumentation-theoretic interpretation of the proof procedure for ALP, consider again figure 6 of example 5.7. The consistency phase for a, shown in the outer double box, can be understood as searching for attacks against $\{a\}$. There are two such

[16]Recall that the abductive proof procedure for ALP employs the restriction that each integrity constraint contains at least one literal with an abducible predicate.

attacks, $\{q^*\}$ and $\{b\}$, shown by the two branches in the figure. $\{q^*\}$ attacks $\{a\}$ via the integrity constraint $\neg(a \wedge p)$ in I, since q^* implies p. Analogously, $\{b\}$ attacks $\{a\}$ via the integrity constraint $\neg(a \wedge q)$ in I, since b implies q. The first attack $\{q^*\}$ is counterattacked by $\{b\}$, via NAF (as in section 4.3), since this implies q. This is shown in the single box. The hypothesis b is added to Δ since the attack $\{b^*\}$ against $\{b\}$, via NAF, is trivially counterattacked by $\{b\}$, via NAF, as sketched in the inner double box. However, $\{b\}$ attacks $\{a\}$, as shown by the right branch in the outer double box. Therefore, Δ attacks itself, and this causes failure of the proof procedure.

The analysis of the proof procedure in terms of attacks and counterattacks suggests the following argumentation-theoretic semantics for ALP. A set of hypotheses Δ is **KM-admissible** if

- for every attack E against Δ,
  Δ attacks $(E - \Delta)$ via NAF alone.

In section 6.5 we will see that the notion of KM-admissible set of hypotheses is similar to the notion of admissibility proposed by Dung [45] for extended logic programming, in that only attacks via NAF are allowed to counterattack.

The argumentation-theoretic interpretation of ALP suggests several ways in which the semantics and proof procedure for ALP can be modified. Firstly, the notion of attack itself can be modified, e.g. following Torres' equivalent formulation of the stable theory semantics [185] (see section 4.3). Secondly, the notion of admissibility can be changed to allow counterattacks via integrity constraints, as well as via NAF. Finally, as in the case of standard LP, the notion of admissibility can be replaced by other semantic notions such as weak stability and acceptability (see section 4.3). The proof procedure for ALP can be modified appropriately to reflect each of these modifications. Such modifications of the semantics and the corresponding modifications of the proof procedure require further investigation.

Using the definition of well-founded semantics given in section 4.3, (non-default) abducibles are always undefined, and consequently fulfill no function, in the well-founded semantics of ALP, as illustrated by the following example.

Example 5.8
Consider the propositional abductive framework $\langle P, A, I\rangle$ where P is

$$p \leftarrow a$$

$A = \{a\}$, and $I = \emptyset$. The well-founded model of $\langle P, A, I\rangle$ is $\emptyset$.

In [136], Pereira, Aparicio and Alferes define an alternative, generalised well-founded semantics for ALP where first programs are extended by a set of abducibles as in the case of generalised stable models, and then the well-founded semantics (rather than stable model semantics) is applied to the extended programs. As a result, the well-founded models of an abductive framework are not unique. In the example above, $\emptyset$, $\{p^*, a^*\}$ and $\{p, a\}$ are the generalised well-founded models of $\langle P, A, I\rangle$. Note that in this application of the well-founded semantics, if an abducible is not in a set of hypotheses Δ then its

negation does not necessarily belong to Δ. Thus the negation of an abducible is not interpreted as NAF. Moreover, since abducible predicates can be undefined some of the non abdicible predicates can also be undefined.

5.4 Computation of abduction through TMS

Satoh and Iwayama [170] present a method for computing generalised stable models for logic programs with integrity constraints represented as denials. The method is a bottom-up computation based upon the TMS procedure of [36]. Although the computation is not goal-directed, goals (or queries) can be represented as denials and be treated as integrity constraints.

Compared with other bottom-up procedures for computing generalised stable model semantics, which first generate stable models and then test the integrity constraints, the method of Satoh and Iwayama dynamically uses the integrity constraints during the process of generating the stable models, in order to prune the search space more efficiently.

Example 5.9
Consider the program P

$$\begin{array}{lcl} p & \leftarrow & q \\ r & \leftarrow & \sim q \\ q & \leftarrow & \sim r \end{array}$$

and the set of integrity constraints $I = \{\neg\, p\}$. P has two stable models $M_1 = \{p, q\}$ and $M_2 = \{r\}$, but only M_2 satisfies I. The proof procedure of [170] deterministically computes only the intended model M_2, without first computing and then rejecting M_1.

In section 8 we will see more generally that truth maintenance systems can be regarded as a form of ALP.

5.5 Simulation of abduction

Satoh and Iwayama [170] also show that an abductive logic program can be transformed into a logic program without abducibles but where the integrity constraints remain. For each abducible predicate p in A, a new predicate p' is introduced, which intuitively represents the complement of p, and a new pair of clauses [17]

$$p(X) \leftarrow\sim p'(X)$$

$$p'(X) \leftarrow\sim p(X)$$

is added to the program. In effect abductive assumptions of the form $p(t)$ are thereby transformed into NAF assumptions of the form $\sim\ p'(t)$. Satoh and Iwayama apply the generalised stable model semantics to the transformed program. However, the transformational semantics, which is effectively employed by Satoh and Iwayama, has the advantage that any semantics can be used for the resulting transformed program.

[17] Satoh and Iwayama use the notation p^* instead of p' and explicitly consider only propositional programs.

Example 5.10
Consider the abductive framework $\langle P, A, I\rangle$ of example 5.1. The transformation generates a new theory P' with the additional clauses

$$a \leftarrow \sim a'$$

$$a' \leftarrow \sim a$$

$$b \leftarrow \sim b'$$

$$b' \leftarrow \sim b.$$

P' has two generalised stable models that satisfy the integrity constraints, namely $M'_1 = M(\Delta_1) \cup \{b'\} = \{a, p, b'\}$, and $M'_2 = M(\Delta_2) = \{a, b, p, q\}$ where $M(\Delta_1)$ and $M(\Delta_2)$ are the generalised stable models seen in example 5.1.

An alternative way of viewing abduction, which emphasises the defeasibility of abducibles, is **retractability** [70]. Instead of regarding abducibles as atoms to be consistently added to a theory, they can be considered as assertions in the theory to be retracted in the presence of contradictions until consistency (or integrity) is restored (c.f. section 6.2).

One approach to this understanding of abduction is presented in [111]. Here, Kowalski and Sadri present a transformation from a general logic program P with integrity constraints I, together with some indication of how to restore consistency, to a new general logic program P' without integrity constraints. Restoration of consistency is indicated by nominating one atom as retractable in each integrity constraint [18]. Integrity constraints are represented as denials, and the atom to be retracted must occur positively in the integrity constraint. The (informally specified) semantics is that whenever an integrity constraint of the form

$$\neg [p \wedge q]$$

has been violated, where the atom p has been nominated as retractable, then consistency should be restored by retracting the instance of the clause of the form

$$p \leftarrow r$$

which has been used to derive the inconsistency.

The transformation of [111] replaces a program P with integrity constraints I by a program P' without integrity constraints which is always consistent with I; and if P is inconsistent with I, then P' represents one possible way to restore consistency (relative to the choice of the retractable atom).

Given an integrity constraint of the form

$$\neg [p \wedge q]$$

where p is retractable, the transformation replaces the integrity constraints and every clause of the form

$$p \leftarrow r$$

[18]Many different atoms can be retractable in the same integrity constraint. Alternative ways of nominating retractable atoms correspond to alternative ways of restoring consistency in P.

by

$$p \leftarrow r, \sim q$$

where the condition $\sim q$ may need to be transformed further, if necessary, into general logic program form, and where the transformation needs to be repeated for every integrity constraint. Kowalski and Sadri show that if P is a stratified program with appropriately stratified integrity constraints I, so that the transformed program P' is stratified, then P' computes the same consistent answers as P with I.

Notice that retracting abducible hypotheses is a special case where the abducibility of a predicate a is represented by an assertion

$$a(X).$$

The following example illustrates the behaviour of the transformation when applied to ALP.

Example 5.11
Consider the simplified version of the event calculus presented in example 2.1. If the integrity constraint

$$\neg[persists(T_1, P, T_2) \wedge happens(E, T) \wedge terminates(E, P) \wedge T_1 \leq T \leq T_2]$$

is violated, then it is natural to restore integrity by retracting the instance of $persists(T_1, P, T_2)$ that has led to the violation. Thus, $persists(T_1, P, T_2)$ is the retractible in this integrity constraint. By applying the transformation sketched above, the integrity constraint and the use of abduction can be replaced by the clauses obtained by further transforming

$$persists(T_1, P, T_2) \leftarrow\sim (happens(E, T), terminates(E, P), T_1 \leq T \leq T_2)$$

into general LP form.

One problem with the retractability semantics is that the equivalence of the original program with the transformed program was proved only in the case where the resulting transformed program is locally stratified. Moreover the proof of equivalence was based on a tedious comparison of search spaces for the two programs. This problem was addressed in a subsequent paper [112] in which integrity constraints are re-expressed as extended clauses and the retractable atoms become explicitly negated conclusions. This use of extended clauses in place of integrity constraints with retractibles is discussed later in section 6.3.

The transformation of [111], applied to ALP, treats all abducibles as default abducibles. In particular, abducibles which do not occur as retractibles in integrity constraints are simply asserted in the transformed program P'. Therefore, this transformation can only be used to eliminate default abducibles together with their integrity constraints. A more complete transformation [183] can be obtained by combining the use of retractibles to eliminate integrity constraints with the transformation of [170] for reducing non-default abducibles to NAF. The new transformation is defined for abductive frameworks where every integrity constraint has a retractible which is either an abducible or the NAF of an abducible.

As an example, consider the propositional abductive logic program $\langle P, A, I\rangle$ where P contains the clause

$$p \leftarrow a$$

a is in A, and I contains the integrity constraint

$$\neg[a \wedge q]$$

where a is retractible. If a is a default abducible, the transformation generates the logic program P'

$$\begin{aligned} &p \leftarrow\sim a' \\ &a' \leftarrow q \\ &a \leftarrow\sim a' \end{aligned}$$

where, as before, a' stands for the complement of a. The first clause in P' is obtained by replacing the positive condition a in the clause in P by the NAF literal $\sim a'$. The second clause replaces the integrity constraint in I. Note that this replaces "a should be retracted" if the integrity constraint $\neg[a \wedge q]$ is violated by "the complement a' of a should be asserted". Finally, the last clause in P' expresses the nature of a as a default abducible. Namely, a holds by default, unless some integrity constraint is violated. In this example, a holds if q does not hold.

If a is a non-default abducible, then the logic program P' obtained by transforming the same abductive program $\langle P, A, I\rangle$ also contains the fourth clause

$$a' \leftarrow\sim a$$

that, together with the third clause, expresses that neither a nor a' need hold, even if no integrity constraint is violated. Note that the last two clauses in P' are those used by Satoh and Iwayama [170] to simulate non-default abduction by means of NAF.

Toni and Kowalski [183] prove that the transformation is correct and complete in the sense that there is a one-to-one correspondence between attacks in the framework $\langle P, A, I\rangle$ and in the framework corresponding to the transformed program P'. Thus, for any semantics that can be defined argumentation-theoretically there is a one-to-one correspondence between the semantics for an abductive logic program and the semantics of the transformed program. As a consequence, any proof procedure for LP which is correct for one of these semantics provides a correct proof procedure for ALP for the analogous semantics (and, less interestingly, vice versa).

In addition to the transformations from ALP to general LP, discussed above, transformations between ALP and disjunctive logic programming (DLP) have also been investigated. Inoue et al. [83] [19], in particular, translate ALP clauses of the form

$$p \leftarrow q, a$$

[19] A description of this work can also be found in [76].

where a is abducible, into DLP clauses

$$(p \wedge a) \vee a' \leftarrow q$$

where a' is a new atom that stands for the complement of a, as expressed by the integrity constraint

$$\neg(a \wedge a'). \tag{3}$$

A model generation theorem-prover (such as SATCHMO or MGTP [58]) can then be applied to compute all the minimal models that satisfy the integrity constraints (3). This transformation is related to a similar transformation [82] for eliminating NAF.

Elsewhere [167], Sakama and Inoue demonstrate a one-to-one correspondence between generalised stable models for ALP and possible models [166] for DLP. Consider, for example, the abductive logic program $\langle P, A, I\rangle$ where P is

$$p \leftarrow a$$

$A = \{a\}$ and I is empty. $M_1 = \emptyset$ and $M_2 = \{a, p\}$ are the generalised stable models of $\langle P, A, I\rangle$. The program can be transformed into a disjunctive logic program P_D

$$\begin{aligned} &p \leftarrow a \\ &a \vee \epsilon. \end{aligned}$$

P_D has possible models $M_1' = \{\epsilon\}$, $M_2' = \{a, p\}$ and $M_3' = \{\epsilon, a, p\}$, such that $M_1' - \{\epsilon\} = M_1$ and $M_2' - \{\epsilon\} = M_3' - \{\epsilon\} = M_2$.

Conversely, [167] shows how to transform DLP programs into ALP. For example, consider the disjunctive logic program P_D

$$\begin{aligned} &a \vee b \leftarrow c \\ &c \end{aligned}$$

whose possible models are $M_1 = \{c, a\}$, $M_2 = \{c, b\}$ and $M_3 = \{c, a, b\}$. It can be transformed into an abductive logic program $\langle P, A, I\rangle$ where P consists of

$$\begin{aligned} &a \leftarrow c, a' \\ &b \leftarrow c, b' \\ &c \end{aligned}$$

a' and b' are new atoms, $A = \{a', b'\}$, and I consists of

$$\neg[c \wedge \sim a \wedge \sim b].$$

$\langle P, A, I\rangle$ has generalised stable models $M_1' = \{c, a, a'\}$, $M_2' = \{c, b, b'\}$ and $M_3' = \{c, a, a', b, b'\}$, such that, if HB is the Herbrand base of P_D, $M_i' \cap HB = M_i$, for each $i = 1, 2, 3$.

Whereas the transformation of [167] deals with inclusive disjunction, Dung [41] presents a simpler transformation that deals with exclusive disjunction, but works only for the case of acyclic programs. For example, the clause

$$p \vee q$$

can be replaced by the two clauses

$$\begin{array}{l} p \leftarrow\sim q \\ q \leftarrow\sim p. \end{array}$$

With this transformation, for acyclic programs, the Eshghi-Kowalski procedure presented in section 4.2 is sound. For the more general case, Dung [42] represents disjunction explicitly and extends the Eshghi-Kowalski procedure by using resolution-based techniques similar to those employed in [57].

5.6 Abduction through deduction from the completion

In the approaches presented so far, hypotheses are generated by backward reasoning with the clauses of logic programs used as inference rules. An alternative approach is presented by Console, Duprè and Torasso [26]. Here clauses of programs are interpreted as if-halves of if-and-only-if definitions that are obtained from the completion of the program [24] restricted to non-abducible predicates. Abductive hypotheses are generated deductively by replacing atoms by their definitions, starting from the observation to be explained.

Given a propositional logic program P with abducible predicates A without definitions in P, let P_C denote the completion of the non-abducible predicates in P. An **explanation formula** for an observation O is the most specific formula F such that

$$P_C \cup \{O\} \models F,$$

where F is formulated in terms of abducible predicates only, and F is **more specific** than F' iff $\models F \rightarrow F'$ and $\not\models F' \rightarrow F$.

Based on this specification, a proof procedure that generates explanation formulas is defined. This proof procedure replaces atoms by their definitions in P_C, starting from a given observation O. Termination and soundness of the proof procedure are ensured for hierarchical programs. The explanation formula resulting from the computation characterises all the different abductive explanations for O, as exemplified in the following example.

Example 5.12
Consider the following program P

$$\begin{array}{lcl} \mathit{wobbly\text{-}wheel} & \leftarrow & \mathit{broken\text{-}spokes} \\ \mathit{wobbly\text{-}wheel} & \leftarrow & \mathit{flat\text{-}tyre} \\ \mathit{flat\text{-}tyre} & \leftarrow & \mathit{punctured\text{-}tube} \\ \mathit{flat\text{-}tyre} & \leftarrow & \mathit{leaky\text{-}valve}, \end{array}$$

where the predicates without definitions are considered to be abducible. The completion P_C is:

$$\mathit{wobbly\text{-}wheel} \leftrightarrow \mathit{broken\text{-}spokes} \vee \mathit{flat\text{-}tyre}$$

$$\mathit{flat\text{-}tyre} \leftrightarrow \mathit{punctured\text{-}tube} \vee \mathit{leaky\text{-}valve}.$$

If O is *wobbly-wheel* then the most specific explanation F is

$$\mathit{broken\text{-}spokes} \vee \mathit{punctured\text{-}tube} \vee \mathit{leaky\text{-}valve},$$

corresponding to the abductive explanations

$$\begin{array}{lcl} \Delta_1 & = & \{\mathit{broken\text{-}spokes}\}, \\ \Delta_2 & = & \{\mathit{punctured\text{-}tube}\}, \\ \Delta_3 & = & \{\mathit{leaky\text{-}valve}\}. \end{array}$$

Console, Duprè and Torasso extend this approach to deal with propositional abductive logic programs with integrity constraints I in the form of denials of abducibles and of clauses expressing taxonomic relationships among abducibles. An explanation formula for an observation O is now defined to be the most specific formula F, formulated in terms of abducible predicates only, such that

$$P_C \cup I \cup \{O\} \models F.$$

The proof procedure is extended by using the denial and taxonomic integrity constraints to simplify F.

In the more general case of non-propositional abductive logic programs, the Clark equality theory CET [24], is used; the notion that F is more specific than F' requires that $F \rightarrow F'$ be a logical consequence of CET and that $F' \rightarrow F$ not be a consequence of CET. The explanation formula is unique up to equivalence with respect to CET. The proof procedure is extended to take into account the equality theory CET.

Denecker and De Schreye [33] compare the search space obtained by reasoning backward using the if-half of the if-and-only-if form of a definite program with that obtained by reasoning forward using the only-if-half. They show an equivalence between the search space for SLD-resolution extended with abduction and the search space for model generation with SATCHMO [122] augmented with term rewriting to simulate unification.

5.7 Abduction and Constraint Logic Programming

ALP has many similarities with constraint logic programming (CLP). Recognition of these similarities has motivated a number of recent proposals to unify the two frameworks.

Both frameworks distinguish two kinds of predicates. The first kind is defined by ordinary LP clauses, and is eliminated during query evaluation. The second kind is "constrained" either by integrity constraints in the case of ALP or by means of a built-in semantic domain in the case of CLP. In both cases, an answer to a query is a "satisfiable" formula involving only the second kind of predicate.

Certain predicates, such as inequality, can be treated either as abducible or as constraint predicates. Treated as abducible, they are constrained by explicitly formulated integrity constraints such as

$$X < Z,\, Z < Y \rightarrow X < Y$$
$$\neg[X < Y \wedge Y < X].$$

Treated as constraint predicates, they are tested for satisfiability by using specialised algorithms which respect the semantics of the underlying domain. Constraints can also be simplified, replacing, for example,

$$2 < t \wedge 3 < t$$

by

$$3 < t.$$

Such simplification is less common in abductive frameworks.

A number of proposals have been made recently to unify the treatment of abducibles and constraints. Several of these, [50, 176, 120, 100] in particular, have investigated the implementation of specialised constraint satisfaction and simplification algorithms of CLP (specifically for inequality) by means of general-purpose integrity checking methods applied to domain-specific integrity constraints as in the case of ALP.

Kowalski [109] proposes a general framework which attempts to unify ALP and CLP using if-and-only-if definitions for ordinary LP predicates and using integrity constraints for abducible and constraint predicates. Abduction is performed by means of deduction in the style of [26] (see section 5.6). This framework has been developed further by Fung [60] and has been applied to job-shop scheduling by Toni [184]. A related proposal, to include user-defined constraint handling rules within a CLP framework, has been made by Frühwirth [75].

Bürchert [18] and Bürchert and Nutt [19], on the other hand, define a framework for general clausal resolution and show how abduction without integrity constraints can be treated as a special case of constrained resolution.

Another approach, which integrates both frameworks while preserving their identity, has been developed by Kakas and Michael [101]. In this approach, the central notions of the two frameworks are combined, so that abduction and constraint handling cooperate to solve a common goal. Typically, the goal is reduced first by abduction to abducible hypotheses whose integrity checking reduces this further to a set of constraints to be satisfied in CLP.

Constructive abduction is the generation of non-ground abductive explanations, such as $\Delta = \{\exists X\, a(X)\}$. The integrity checking of such abducible hypotheses involves the introduction of equality assumptions, which can naturally be understood in CLP terms. A

procedure for performing constructive abduction within a framework that treats equality as an abducible predicate and the Clark equality theory as a set of integrity constraint was first proposed by Eshghi [50]. Building upon this proposal, Kakas and Mancarella [98] extend the abductive proof procedure for LP in [54] (see section 4.2) to combine constructive negation with constructive abduction in a uniform way, by reducing the former to the latter using the abductive interpretation of NAF.

The problem of constructive abduction has also been studied within the completion semantics. Denecker and De Schreye [34] define a proof procedure for constructive abduction, SLDNFA, which they show is sound and complete. Teusink [181] extends Drabent's [38] procedure, SLDNA, for constructive negation to perform constructive abduction and uses three-valued semantics to show soundness and completeness. In both proposals, [34] and [181], integrity constraints are dealt with by means of a transformation, rather than explicitly.

6 Extended Logic Programming

Extended Logic Programming (ELP) extends general LP by allowing, in addition to NAF, a second, explicit form of negation. Explicit negation can be used, when the definition of a predicate is incomplete, to explicitly define negative instances of the predicate, instead of having them inferred implicitly using NAF.

Clauses with explicit negation in their conclusions can also serve a similar function to integrity constraints with retractibles. For example, the integrity constraint

$$\neg[persists(T_1,\, P,\, T_2) \wedge happens(E,\, T) \wedge terminates(E,\, P) \wedge T_1 \leq T \leq T_2]$$

with $persists(T_1,\, P,\, T_2)$ retractible can be reformulated as a clause with explicit negation in the conclusion

$$\neg persists(T_1,\, P,\, T_2) \leftarrow happens(E,\, T), terminates(E,\, P), T_1 \leq T \leq T_2.$$

6.1 Answer set semantics

In general logic programs, negative information is inferred by means of NAF. This is appropriate when the closed world assumption [157], that the program gives a complete definition of the positive instances of a predicate, can safely be applied. It is not appropriate when the definition of a predicate is incomplete and therefore "open", as in the case of abducible predicates.

For open predicates it is possible to extend logic programs to allow **explicit negation** in the conclusions of clauses. In this section we will discuss the extension proposed by Gelfond and Lifschitz [69]. This extension is based on the stable model semantics, and can be understood, therefore, in terms of abduction, as we have already seen.

Gelfond and Lifschitz define the notion of **extended logic programs**, consisting of clauses of the form:

$$L_0 \leftarrow L_1, \ldots, L_m, \sim L_{m+1}, \ldots, \sim L_n,$$

where $n \geq m \geq 0$ and each L_i is either an atom (A) or the explicit negation of an atom ($\neg A$). This negation denoted by "$\neg$" is called "classical negation" in [69]. However, as we will see below, because the contrapositives of extended clauses do not hold, the term "classical negation" can be regarded as inappropriate. For this reason we use the term "explicit negation" instead.

A similar notion has been investigated by Pearce and Wagner [130], who develop an extension of definite programs by means of Nelson's strong negation. They also suggest the possibility of combining strong negation with NAF. Akama [1] argues that the semantics of this combination of strong negation with NAF is equivalent to the answer set semantics for extended logic programs developed by Gelfond and Lifschitz.

The semantics of an extended program is given by its answer sets, which are like stable models but consist of both positive and (explicit) negative literals. Perhaps the easiest way to understand the semantics is to transform the extended program P into a general logic program P' without explicit negation, and to apply the stable model semantics to the resulting general logic program. The transformation consists in replacing every occurrence of explicit negation $\neg p(t)$ by a new (positive) atom $p'(t)$. The stable models of P' which do not contain a ontradiction of the form $p(t)$ and $p'(t)$ correspond to the **consistent answer sets** of P. The corresponding answer sets of P contain explicit negative literals $\neg p(t)$ wherever the stable models contain $p'(t)$. In [69] the answer sets are defined directly on the extended program by modifying the definition of the stable model semantics. The consistent answer sets of P also correspond to the generalised stable models (see section 5.1) of P' with a set of integrity constraints $\forall X \neg [p(X) \wedge p'(X)]$, for every predicate p.

In the general case a stable model of P' might contain a contradiction of the form $p(t)$ and $p'(t)$. In this case the corresponding **inconsistent answer set** is defined to be the set of all the variable-free (positive and explicit negative) literals. It is in this sense that explicit negation can be said to be "classical". The same effect can be obtained by explicitly augmenting P' by the clauses

$$q(X) \leftarrow p(X),\ p'(X)$$

for all predicate symbols q and p in P'. Then the **answer sets** of P simply correspond to the stable models of the augmented set of clauses. If these clauses are not added, then the resulting treatment of explicit negation gives rise to a **paraconsistent** logic, i.e. one in which contradictions are tolerated.

Notice that, although Gelfond and Lifschitz define the answer set semantics directly without transforming the program and then applying the stable model semantics, the transformation can also be used with any other semantics for the resulting transformed program. Thus Przymusinski [153] for example applies the well-founded semantics to extended logic programs. Similarly any other semantics can also be applied. As we have seen before, this is one of the main advantages of transformational semantics in general.

An important problem for the practical use of extended programs is how to distinguish whether a negative condition is to be interpreted as explicit negation or as NAF. This

problema will be addressed in sections 6.4 and 9.

6.2 Restoring consistency of answer sets

The answer sets of an extended program are not always consistent.

Example 6.1
The extended logic program:

$$\begin{aligned}&\neg fly(X) \leftarrow \sim bird(X)\\&fly(X) \leftarrow bat(X)\\&bat(tom)\end{aligned}$$

has no consistent answer set.

As mentioned in section 6.1, this problem can be dealt with by employing a paraconsistent semantics. Alternatively, in some cases it is possible to restore consistency by removing some of the NAF assumptions implicit in the answer set. In the example above we can restore consistency by rejecting the NAF assumption $\sim bird(tom)$ even though $bird(tom)$ does not hold. We then get the consistent set $\{bat(tom), fly(tom)\}$. This problem has been studied in [46] and [137]. Both of these studies are primarily concerned with the related problem of inconsistency of the well-founded semantics when applied to extended logic programs [153].

To deal with the problem of inconsistency in extended logic programs, Dung and Ruamviboonsuk [46] apply the preferred extension semantics to a new abductive framework derived from an extended logic program. An extended logic program P is first transformed into an ordinary general logic program P' by renaming explicitly negated literals $\neg p(t)$ by positive literals $p'(t)$. The resulting program is then further transformed into an abductive framework by renaming NAF literals $\sim q(t)$ by positive literals $q^*(t)$ and adding the integrity constraints

$$\forall X \neg [q(X) \wedge q^*(X)]$$

as described in section 4.3. Thus if p' expresses the explicit negation of p the set A^* will contain p'^* as well as p^*. Moreover Dung includes in I^* additional integrity constraints of the form

$$\forall X \neg [p(X) \wedge p'(X)]$$

to prevent contradictions.

Extended preferred extensions are then defined by modifying the definition of preferred extensions in section 4 for the resulting abductive framework with this new set I^* of integrity constraints. The new integrity constraints in I^* have the effect of removing a NAF hypothesis when it leads to a contradiction. Clearly, any other semantics for logic programs with integrity constraints could also be applied to this framework.

Pereira, Aparicio and Alferes [137] employ a similar approach within the context of Przymuszynski's extended stable models [153]. It consists in identifying explicitly all the possible sets of NAF hypotheses which lead to an inconsistency and then restoring consistency by removing at least one hypothesis from each such set. This method can be viewed as a form of belief revision, where if inconsistency can be attributed to an abducible hypothesis or a retractable atom (see below section 5.5), then we can reject the hypothesis to restore consistency. In fact Pereira, Aparicio and Alferes have also used this method to study counterfactual reasoning [139]. Alferes and Pereira [5] have shown that this method of restoring consistency can also be viewed in terms of inconsistency avoidance.

This method [137] is not able to restore consistency in all cases, as illustrated by the following example.

Example 6.2
given the extended logic program

$$\begin{aligned} &p \leftarrow q \\ &q \leftarrow p \\ &r \leftarrow\sim p \\ &\neg r \leftarrow\sim p \end{aligned}$$

the method of [137] is unable to restore consistency by withdrawing the hypothesis p^*.

In [134] and [140], Pereira and Alferes present two different modifications of the method of [137] to deal with this problem. For the program in example 6.2, the method in [134] restores consistency by letting p undefined, while the method in [140] restores consistency by assigning p to truth. This second method is more suitable for diagnosis applications.

Both methods, [46] and [137, 134, 140], can deal only with inconsistencies that can be attributed to NAF hypotheses, as shown by the following example.

Example 6.3
It is not possible to restore consistency by removing NAF hypotheses given the program:

$$p$$

$$\neg p.$$

However, Inoue [81, 80] suggests a general method for restoring consistency, which is applicable to this case. This method (see also section 6.3) is based on [66] and [145] and consists in isolating inconsistencies by finding maximally consistent subprograms. In this approach a knowledge system is represented by a pair (P, H), where:

1. P and H are both extended logic programs,
2. P represents a set of facts,
3. H represents a set of assumptions.

The semantics is given using abduction as in [145] (see section 3) by means of theory extensions $P \cup \Delta$ of P, with $\Delta \subseteq H$ maximal with respect to set inclusion, such that $P \cup \Delta$ has a consistent answer set.

In this approach, whenever the answer set of an extended logic program P is inconsistent, it is possible to restore consistency by regarding it as a knowledge system of the form

$$(\emptyset, P).$$

For example 6.3 this will give two alternative semantics, $\{p\}$ or $\{\neg p\}$.

A similar approach to restoring consistency follows also from the work in [87, 99] (see section 7), where argumentation-based semantics can be used to select acceptable (and hence consistent) subsets of an inconsistent extended logic program.

6.3 Rules and exceptions in LP

Another way of restoring consistency of answer sets is presented in [112], where sentences with explicitly negated conclusions are given priority over sentences with positive conclusions. In this approach, extended clauses with negative conclusions are similar to integrity constraints with retractibles.

Example 6.4
Consider the program

$$\begin{aligned} &fly(X) \leftarrow bird(X) \\ &walk(X) \leftarrow ostrich(X) \\ &bird(X) \leftarrow ostrich(X) \\ &ostrich(john) \end{aligned}$$

and the integrity constraint

$$\neg[fly(X) \wedge walk(X)],$$

with $fly(X)$ retractable. The integrity constraint is violated, because both $walk(john)$ and $fly(john)$ hold. Following the approach presented in section 5.5, integrity can be restored by retracting the instance

$$fly(john) \leftarrow bird(john)$$

of the first clause in the program. Alternatively, the integrity constraint can be formulated as a clause with an explicit negative conclusion

$$\neg fly(X) \leftarrow walk(X).$$

In the new formulation it is natural to interpret clauses with negative conclusions as exceptions, and clauses with positive conclusions as default rules. In this example, the extended clause

$$\neg fly(X) \leftarrow walk(X)$$

can be interpreted as an exception to the "general" rule

$$fly(X) \leftarrow bird(X).$$

To capture the intention that exceptions should override general rules, Kowalski and Sadri [112] modify the answer set semantics, so that instances of clauses with positive conclusions are retracted if they are contradicted by explicit negative information.

Kowalski and Sadri [112] also present a transformation, which preserves the new semantics, and is arguably a more elegant form of the transformation presented in [111] (see section 5.5). In the case of the flying-birds example described above the new transformation gives the clause

$$fly(X) \leftarrow bird(X), \sim \neg fly(X).$$

This can be further transformed by "macroprocessing" the call to $\neg fly(X)$, giving the result of the original transformation in [111]

$$fly(X) \leftarrow bird(X), \sim walk(X).$$

In general, the new transformation introduces a new condition

$$\sim \neg p(t)$$

into every clause with a positive conclusion $p(t)$. The condition is vacuous if there are no exceptions with $\neg p$ in the conclusion. The answer set semantics of the new program is equivalent to the modified answer set semantics of the original program, and both are consistent. Moreover, the transformed program can be further transformed into a general logic program by renaming explicit negations $\neg p$ by new positive predicates p'. Because of this renaming, positive and negative predicates can be handled symmetrically, and therefore, in effect, clauses with positive conclusions can represent exceptions to rules with (renamed) negative conclusions. Thus, for example, a negative rule such as

$$\neg fly(X) \leftarrow walk(X)$$

with a positive exception

$$fly(X) \leftarrow super\_ostrich(X)$$

can be transformed into a clause

$$\neg fly(X) \leftarrow walk(X), \sim fly(X)$$

and all occurrences of the negative literal $\neg fly(X)$ can be renamed by a new positive literal $fly'(X)$. This is not entirely adequate for a proper treatment of exceptions to exceptions. However, this approach can be extended, as we shall see in section 6.6.

More direct approaches to the problem of treating positive and negative predicates symmetrically in default reasoning are presented in [81, 80], following the methods of [66] and [145] (see section 6.2 for a discussion), and in [87, 99], based on an argumentation-theoretic framework (see sections 6.4 and 7).

6.4 (Extended) Logic Programming without Negation as Failure

Kakas, Mancarella and Dung [99] show that the Kowalski-Sadri transformation presented in section 6.3 can be applied in the reverse direction, to replace clauses with NAF by clauses with explicit negation together with a priority ordering between extended clauses. Thus, for example,

$$fly(X) \leftarrow bird(X), \sim walk(X)$$

can be transformed "back" to

$$fly(X) \leftarrow bird(X)$$
$$\neg fly(X) \leftarrow walk(X)$$

together with an ordering that indicates that the second clause has priority over the first. In general, the extended clauses

$$\begin{aligned} &r : \ p \leftarrow q_1, \ldots, q_n \\ &r_1 : \ \neg p \leftarrow s_1 \\ &\vdots \\ &r_k : \ \neg p \leftarrow s_k \end{aligned}$$

generated by transforming the clause

$$p \leftarrow q_1 \ldots q_n, \sim s_1, \ldots, \sim s_k$$

are ordered so that $r_j > r$ for $1 \leq j \leq k$. In [99], the resulting prioritised clauses are formulated in an ELP framework (with explicit negation) without NAF but with an ordering relation on the clauses of the given program.

This new framework for ELP is proposed in [99] as an example of a general theory of the acceptability semantics (see section 4.3) developed within the argumentation-theoretic framework introduced in [88] (see section 7). Its semantics is based upon an appropriate notion of attack between subtheories consisting of partially ordered extended clauses in a theory T. Informally, for any subsets E and Δ of T such that $E \cup \Delta$ have a contradictory consequence, E attacks Δ if and only if either E does not contain a clause which is lower than some clause in Δ or if E does contain such a clause, it also contains some clause which is higher than a clause in Δ. Thus, the priority ordering is used to break the symmetry between the incompatible sets E and Δ. Hence in the example above, if we have a bird that walks, then the subtheory which, in addition to these two facts, consists of the second clause

$$\neg fly(X) \leftarrow walk(X)$$

attacks the subtheory consisting of the clause

$$fly(X) \leftarrow bird(X)$$

and the same two facts, but not vice versa; so, the first subtheory is acceptable whereas the second one is not.

Kakas, Mancarella and Dung show that, with this notion of attack in the new framework with explicit negation but without NAF, it is possible to capture exactly the semantics of NAF in LP. This shows that, if LP is extended with explicit negation, then NAF can be simulated by introducing a priority ordering between clauses. Moreover, the new framework of ELP is more general than conventional ELP as it allows any ordering relation on the clauses of extended logic programs.

In the extended logic program which results from the transformation described above, if $\neg p$ holds then $\sim p$ holds in the corresponding general logic program, for any atom p. We can argue, therefore, that the transformed extended logic program satisfies the **coherence principle**, proposed by Pereira and Alferes [135], namely that whenever $\neg p$ holds then $\sim p$ must also hold. They consider the satisfaction of this principle to be a desirable property of any semantics for ELP, as illustrated by the following example, taken from [3].

Example 6.5
Given the extended logic program

$$\neg drivers\_strike$$
$$take\_bus \leftarrow \sim drivers\_strike$$

one should derive the conclusion $take\_bus$.

The coherence principle automatically holds for the answer set semantics. Pereira and Alferes [135] and Alferes, Dung and Pereira [3] have defined new semantics for ELP that incorporates the coherence principle. These semantics are adaptations of Przymuszynski's extended stable model semantics [153] and Dung's preferred extension semantics [39], respectively, to ELP. Alferes, Damasio and Pereira [2] provide a sound and complete proof procedure for the semantics in [135]. The proof procedure is implemented in Prolog by means of an appropriate transformation from ELP to general LP.

6.5 An argumentation-theoretic approach to ELP

The Dung and Ruamviboonsuk semantics for ELP [46] in effect reduces ELP to ALP by renaming the explicit negation $\neg p$ of a predicate p to a new predicate p' and employing integrity constraints

$$\forall X \neg[p(X) \wedge p'(X)]$$

for all predicates p in the program. This reduction automatically provides us with an argumentation-theoretic interpretation of ELP, where attacks via these integrity constraints become **attacks via explicit negation**. Such notions of attack via explicit negation have been defined by Dung [45] and Kakas, Mancarella and Dung [99]. Dung's notion can be formulated as follows: a set of NAF literals E [20] attacks another such set Δ via explicit negation (relative to a program P') [21] if

[20]Note that, for simplicity, here we use NAF literals directly as hypotheses, without renaming them as positive atoms.

[21]P' stands for the extended logic program P where all explicitly negated literals of the form $\neg p(t)$ are rewritten as atoms $p'(t)$.

- $P' \cup E \cup \Delta \vdash p, p'$, for some atom p, and
  $P' \cup E \not\vdash p, p'$, and $P' \cup \Delta \not\vdash p, p'$, for all atoms p.

Kakas, Mancarella and Dung's notion can be formulated as follows: E attacks a non-empty set Δ via explicit negation (relative to a program P') if

- $P' \cup E \vdash l$ and $P' \cup \Delta \vdash \overline{l}$, for some literal l,

where $\overline{p} = p'$ and $\overline{p'} = p$.

Augmenting the notion of attack via NAF by either of these new notions of attack via explicit negation, we can define admissibility, weak stability and acceptability semantics similarly to the definitions in section 4.3. However, the resulting semantics might give unwanted results, as illustrated by the following example given in [45].

Example 6.6
Given the extended logic program

$$fly(X) \leftarrow bird(X), \sim ab\_bird(X)$$
$$\neg fly(X) \leftarrow penguin(X), \sim ab\_penguin(X)$$
$$bird(X) \leftarrow penguin(X)$$
$$penguin(tweety)$$
$$ab\_bird(X) \leftarrow penguin(X), \sim ab\_penguin(X)$$

$\{ab\_penguin^*(tweety)\}$ attacks $\{ab\_bird^*(tweety)\}$ via NAF. However, $\{ab\_bird^*(tweety)\}$ attacks $\{ab\_penguin^*(tweety)\}$ via explicit negation (and vice versa). Therefore, $\{ab\_bird^*(tweety)\}$ counterattacks all attacks against it, and is admissible. As a consequence, $fly(tweety)$ holds in the extension given by $\{ab\_bird^*(tweety)\}$. However, intuitively $fly(tweety)$ should hold in no extension.

To cope with this problem, Dung [45] suggests the following semantics, while keeping the definition of attack unchanged. A set of hypotheses is **D-admissible** if

- Δ does not attack itself, either via explicit negation or via NAF, and
- for every attack E against Δ, either via explicit negation or via NAF, Δ attacks E via NAF.

Note that, if ELP is seen as a special instance of ALP, then D-admissibility is very similar to KM-admissibility, presented in section 5.3 for ALP, in that the two notions share the feature that counterattacks can only be provided by means of attacks via NAF.

It can be argued, however, that the problem in this example lies not so much with the semantics but with the representation itself. The last clause

$$ab\_bird(X) \leftarrow penguin(X), \sim ab\_penguin(X)$$

can be understood as attempting to assign a higher priority to the second clause of the program over the first. This can be done, without this last clause, explicitly in the ELP framework with priorities of [99] (section 6.4) or in the rules and exceptions approach

[112] (section 6.3).

An argumentation-theoretic interpretation for ELP has also been proposed by Bondarenko, Toni and Kowalski [11]. Their proposal, which requires that $P' \cup \Delta$ be consistent with the integrity constraints

$$\forall X \neg[p(X) \wedge p'(X)]$$

for each predicate p, instead of using a separate notion of attack via explicit negation, has certain undesirable consequences, as shown in [4]. For example, the program

$$p \leftarrow \sim p, \sim q$$

admits both $\{\sim q\}$ and $\{\sim p\}$ as admissible extensions, while the only intuitively correct extension is $\{\sim q\}$.

Alferes and Pereira [4] use argumentation-theoretic notions to extend the well-founded semantics for ELP in [135]. Kakas, Mancarella and Dung [99] also define a well-founded semantics for ELP based upon argumentation-theoretic notions.

6.6 A methodology for default reasoning with explicit negation

Compared with other authors, who primarily focus on extending or modifying the semantics of LP to deal with default reasoning, Pereira, Aparicio and Alferes [136] develop a methodology for performing default reasoning with extended logic programs. Defaults of the form "normally if q then p" are represented by an extended clause

$$p \leftarrow q, \sim \neg nameqp, \sim \neg p \tag{4}$$

where the condition $nameqp$ can be understood as a name given to the default. The condition $\sim \neg p$ deals with exceptions to the conclusion of the rule, whilst the condition $\sim \neg nameqp$ deals with exceptions to the rule itself. An exception to the rule would be represented by an extended clause of the form

$$\neg nameqp \leftarrow r$$

where the condition r represents the conditions under which the exception holds. In the flying-birds example, the second clause of

$$fly(X) \leftarrow bird(X), \sim \neg birds\_fly, \sim \neg fly(X) \tag{5}$$
$$\neg birds\_fly(X) \leftarrow penguin(X) \tag{6}$$

expresses that the default named $birds\_fly$ does not apply for penguins.

The possibility of expressing both exceptions to rules as well as exceptions to predicates is useful for representing hierarchies of exceptions. Suppose we want to change (6) to the default rule "penguins usually don't fly". This can be done by replacing (6) by

$$\neg fly(X) \leftarrow penguin(X), \sim \neg penguins\_don't\_fly(X), \sim fly(X) \tag{7}$$

where $penguins\_don't\_fly$ is the name assigned to the new rule. To give preference to the more specific default represented by (7) over the more general default (5), we add the additional clause

$$\neg birds\_fly(X) \leftarrow penguin(X), \sim \neg penguins\_don't\_fly(X).$$

Then to express that superpenguins fly, we can add the rule:

$$\neg penguins\_don't\_fly(X) \leftarrow superpenguin(X).$$

Pereira, Aparicio and Alferes [136] use the well-founded semantics extended with explicit negation to give a semantics for this methodology for default reasoning. However it is worth noting that any other semantics of extended logic programs could also be used. For example Inoue [81, 80] uses an extension of the answer set semantics (see section 6.2), but for a slightly different transformation.

6.7 ELP with abduction

Inoue [80] (see also section 6.3) and Pereira, Aparicio and Alferes [136] investigate extended logic programs with abducibles but without integrity constraints. They transform such programs into extended logic programs without abduction by adding a new pair of clauses

$$p(X) \leftarrow \sim \neg p(X)$$
$$\neg p(X) \leftarrow \sim p(X)$$

for each abducible predicate p. Notice that the transformation is identical to that of Satoh and Iwayama [170] presented in section 5.5, except for the use of explicit negation instead of new predicates. Inoue [80] and Pereira, Aparicio and Alferes [136] assign different semantics to the resulting program. Whereas Inoue applies the answer set semantics, Pereira, Aparicio and Alferes apply the extended stable model semantics of [153]. Pereira, Aparicio and Alferes [138] have also developed proof procedures for this semantics.

As mentioned above, Pereira, Aparicio and Alferes [136] understand the transformed programs in terms of (three-valued) extended stable models. This has the advantage that it gives a semantics to every logic program and it does not force abducibles to be either believed or disbelieved. But the advantage of the transformational approach, as we have already remarked, is that the semantics of the transformed program is independent of the transformation. Any semantics can be used for the transformed program (including even a transformational one, e.g. replacing explicitly negated atoms $\neg p(t)$ by a new atom $p'(t)$).

7 An Abstract Argumentation-based Framework for Default Reasoning

Following the argumentation-theoretic interpretation of NAF introduced in [88], Kakas [87] generalised the interpretation and showed how other logics for default reasoning can be based upon a similar semantics. In particular, he showed how default logic can be understood in such terms and proposed a default reasoning framework based on the argumentation-theoretic acceptability semantics (see section 4.3) as an alternative to default logic.

Dung [44] proposed an abstraction of the argumentation-theoretic interpretation of NAF introduced in [88], where arguments and the notion of one argument attacking another are treated as primitive concepts which can be superimposed upon any monotonic logic and can even be introduced into non-linguistic contexts. Stable, admissible, preferred, and well-founded semantics can all be defined in terms of sets of arguments that are able to attack or defend themselves against attack by other arguments. Dung shows that many problems from such different areas as AI, game theory and economics can be formulated and studied within this argumentation-theoretic framework.

Bondarenko, Toni and Kowalski [11] modified Dung's notion of an abstract argumentation-theoretic framework by defining an argument to be a monotonic derivation from a set of abductive assumptions. This new framework, like that of [87], can be understood as a natural abstraction and extension of the Theorist framework in two respects. First, the underlying logic can be any monotonic logic and not just classical first-order logic. Second, the semantics of the non-monotonic extension can be formulated in terms of any argumentation-theoretic notion, and not just in terms of maximal consistency.

To give an idea of this framework, we show here how a simplified version of the framework can be used to define an abstract notion of stable semantics which includes as special cases stable models for logic programs and extensions for default logic [158], autoepistemic logic [126] and non-monotonic logic II [119]. We follow the approach of Bondarenko, Dung, Kowalski and Toni [12] (see also [87]).

Let T be a set of sentences in any monotonic logic, $\vdash$ the provability operator for that logic and A a set of candidate abducible sentences. For any $\alpha \in A$, let $\overline{\alpha}$ be some sentence that represents the "contrary" of α. Then, a set of assumptions E is said to attack a set of assumptions Δ iff

- $T \cup E \vdash \overline{\alpha}$ for some $\alpha \in \Delta$.

Note that the notion a sentence $\overline{\alpha}$ being the contrary of an assumption α can be regarded as a special case of the more general notion that α is retractible in an integrity constraint

$$\neg[\alpha \wedge \beta] \text{ (here } \beta \text{ is "a contrary" of } \alpha).$$

This more general notion is useful for capturing the semantics of ALP.

To cater for the semantics of LP, T is a general logic program, $\vdash$ is modus ponens and A is the set of all negative literals. The contrary of $\sim p$ is p.

For default logic, default rules are rewritten as sentences of the form

$$\gamma(x) \leftarrow \alpha(x) \wedge M\beta_1(x) \wedge \ldots \wedge M\beta_n(X)$$

(similarly to Poole's simulation of default logic, section 3), where the underlying language is first-order logic augmented with a new symbol "M" which creates a sentence from a sentence not containing M, and with a new implication symbol $\leftarrow$ in addition to the usual implication symbol for first-order logic. The theory T is $\mathcal{F} \cup D$, where $\mathcal{F}$ is the set of "facts" and D is the set of defaults written as sentences. $\vdash$ is ordinary provability for classical logic augmented with modus ponens for the new implication symbol. (This is different from Poole's simulation, which treats $\leftarrow$ as ordinary implication.) The set A is the set of all sentences of the form $M\alpha$. The contrary of $M\alpha$ is $\neg\alpha$.

For autoepistemic logic, the theory T is any set of sentences written in modal logic. However, $\vdash$ is provability in classical (non-modal) logic. The set A is the set of all sentences of the form $\neg L\phi$ or $L\phi$. The contrary of $\neg L\phi$ is ϕ, whereas the contrary of $L\phi$ is $\neg L\phi$.

For non-monotonic logic II, T is any set of sentences of modal logic, as in the case of autoepistemic logic, but $\vdash$ is provability in modal logic (including the inference rule of necessitation, which derives $L\phi$ from ϕ). The set A is the set of all sentences of the form $\neg L\phi$. The contrary of $\neg L\phi$ is ϕ.

Given any theory T in any monotonic logic, candidate assumptions A and notion of the "contrary" of an assumption, a set of assumptions Δ is **stable** iff

- Δ does not attack itself and
- Δ attacks all $\{\alpha\}$ such that $\alpha \in A - \Delta$.

This notion of stability includes as special cases stable models in LP and extensions in default logic, autoepistemic logic and non-monotonic logic II.

Based upon this abductive interpretation of default logic, Satoh [169] proposes a sound and complete proof procedure for default logic, by extending the proof procedure for ALP of [172].

At a similar level of abstraction, Kakas, Mancarella and Dung [99] also propose a general argumentation-theoretic framework based primarily on the acceptability semantics. As with LP, other semantics such as preferred extension and stable theory semantics can be obtained as approximations of the acceptability semantics. A sceptical form of semantics, analogous to the well-founded semantics for LP, is also given in [99], based on a strong form of acceptability.

Kakas, Mancarella and Dung define a notion of attack between conflicting sets of sentences, but these can be any subtheories of a given theory, rather than being subtheories drawn from a pre-assigned set of assumption sentences as in [11, 12]. Also as in the special case of LP (see section 4.3) this notion of attack together with the acceptability semantics ensures that defences are genuine counterattacks, i.e. that they do not at the same time attack the theory that we are trying to construct.

Because this framework does not separate the theory into facts and candidate assumptions, the attacking relation would be symmetric. To avoid this, a priority relation can be given on the sentences of the theory. As an example of this approach, Kakas, Mancarella and Dung propose a framework for ELP where programs are accompanied by a priority ordering on their clauses and show how in this framework NAF can be removed from the object-level language (see also section 6.4). More generally, this approach provides a framework for default reasoning with priorities on sentences of a theory, viewed as default rules. It also provides a framework for restoring consistency in a theory T by using the acceptable subsets of T (see sections 6.2 and 6.3).

Brewka and Konolige [15] also propose an abductive framework which unifies and provides new semantics for LP, autoepistemic logic and default logic, but does not use argumentation-theoretic notions. This semantics generalises the semantics for LP given in [14].

8 Abduction and Truth Maintenance

In this section we will consider the relationship between truth maintenance (TM) and abduction. TM systems have historically been presented from a procedural point of view. However, we will be concerned primarily with the semantics of TM systems and the relationship to the semantics of abductive logic programming.

A TM system is part of an overall reasoning system which consists of two components: a domain dependent problem solver which performs inferences and a domain independent TM system which records these inferences. Inferences are communicated to the TM system by means of **justifications**, which in the simplest case can be written in the form

$$p \leftarrow p_1, \ldots, p_n$$

expressing that the proposition p can be derived from the propositions $p_1, \ldots, p_n$. Justifications include **premises**, in the case $n = 0$, representing propositions which hold in all contexts. Propositions can depend upon **assumptions** which vary from context to context.

TM systems can also record **nogoods**, which can be written in the form

$$\neg (p_1, \ldots, p_n),$$

meaning that the propositions $p_1, \ldots, p_n$ are incompatible and therefore cannot hold together.

Given a set of justifications and nogoods, the task of a TM system is to determine which propositions can be derived on the basis of the justifications, without violating the nogoods.

For any such TM system there is a straight-forward correspondence with abductive logic programs:

- justifications correspond to propositional Horn clause programs,
- nogoods correspond to propositional integrity constraints,
- assumptions correspond to abducible hypotheses, and
- contexts correspond to acceptable sets of hypotheses.

The semantics of a TM system can accordingly be understood in terms of the semantics of the corresponding propositional logic program with abducibles and integrity constraints.

The two most popular systems are the justification-based TM system (JTMS) of Doyle [36] and the assumption-based TM system (ATMS) of deKleer [102].

8.1 Justification-based truth maintenance

A justification in a JTMS can be written in the form

$$p \leftarrow p_1, \ldots, p_n, \sim p_{n+1}, \ldots, \sim p_m,$$

expressing that p can be derived (i.e. is IN in the current set of beliefs) if $p_1, \ldots, p_n$ can be derived (are IN) and $p_{n+1}, \ldots, p_m$ cannot be derived (are OUT).

For each proposition occurring in a set of justifications, the JTMS determines an IN or OUT label, taking care to avoid circular arguments and thus ensuring that each proposition which is labelled IN has well-founded support. The JTMS incrementally revises beliefs when a justification is added or deleted.

The JTMS uses nogoods to record contradictions discovered by the problem solver and to perform **dependency-directed backtracking** to change assumptions in order to restore consistency. In the JTMS changing an assumption is done by changing an OUT label to IN.

Suppose, for example, that we are given the justifications

$$p \leftarrow \sim q$$

$$q \leftarrow \sim r$$

corresponding to the propositional form of the Yale shooting problem. As Morris [127] observes, these correctly determine that q is labelled IN and that r and p are labelled OUT. If the JTMS is subsequently informed that p is true, then dependency-directed backtracking will install a justification for r, changing its label from OUT to IN. Notice that this is similar to the behaviour of the extended abductive proof procedure described in example 5.5, section 5.2.

Several authors have observed that the JTMS can be given a semantics corresponding to the semantics of logic programs, by interpreting justifications as propositional logic program clauses, and interpreting $\sim p_i$ as NAF of p_i. The papers [49, 71, 92, 141], in particular, show that a well-founded labelling for a JTMS corresponds to a stable model of the corresponding logic program. Several authors [49, 59, 92, 156], exploiting the interpretation of stable models as autoepistemic expansions [68], have shown a correspondence between well-founded labellings and stable expansions of the set of justifications viewed as autoepistemic theories.

The JTMS can also be understood in terms of abduction using the abductive approach to the semantics of NAF, as shown in [40, 71, 92]. This has the advantage that the nogoods of the JTMS can be interpreted as integrity constraints of the abductive framework. The correspondence between abduction and the JTMS is reinforced by [170], which gives a proof procedure to compute generalised stable models using the JTMS (see section 5.4).

8.2 Assumption-based truth maintenance

Justifications in ATMS have the more restricted Horn clause form

$$p \leftarrow p_1, \ldots, p_n.$$

However, whereas the JTMS maintains only one implicit context of assumptions at a time, the ATMS explicitly records with every proposition the different sets of assumptions which provide the foundations for its belief. In ATMS, assumptions are propositions that have been pre-specified as assumable. Each record of assumptions that supports a proposition p can also be expressed in Horn clause form

$$p \leftarrow a_1, \ldots, a_n$$

and can be computed from the justifications, as we illustrate in the following example.

Example 8.1
Suppose that the ATMS contains justifications

$$\begin{array}{lcl} p & \leftarrow & a, b \\ p & \leftarrow & b, c, d \\ q & \leftarrow & a, c \\ q & \leftarrow & d, e \end{array}$$

and the single nogood

$$\neg (a, b, e)$$

where a, b, c, d, e are assumptions. Given the new justification

$$r \leftarrow p, q$$

the ATMS computes explicit records of r's dependence on the assumptions:

$$\begin{array}{lcl} r & \leftarrow & a, b, c \\ r & \leftarrow & b, c, d, e. \end{array}$$

The dependence

$$r \leftarrow a, b, d, e.$$

is not recorded because its assumptions violate the nogood. The dependence

$$r \leftarrow a, b, c, d$$

is not recorded because it is subsumed by the dependence

$$r \leftarrow a, b, c.$$

Reiter and deKleer [162] show that, given a set of justifications, nogoods, and candidate assumptions, the ATMS can be understood as computing minimal and consistent abductive explanations in the propositional case (where assumptions are interpreted as abductive hypotheses). This abductive interpretation of ATMS has been developed further by Inoue [79], who gives an abductive proof procedure for the ATMS.

Given an abductive logic program P and goal G, the explicit construction in ALP of a set of hypotheses Δ, which together with P implies G and together with P satisfies any integrity constraints I, is similar to the record

$$G \leftarrow \Delta$$

computed by the ATMS. There are, however, some obvious differences. Whereas ATMS deals only with propositional justifications, relying on a separate problem solver to instantiate variables, ALP deals with general clauses, combining the functionalities of both a problem solver and a TM system.

The extension of the ATMS to the non-propositional case requires a new notion of minimality of sets of assumptions. Minimality as subset inclusion is not sufficient, but needs to be replaced by a notion of minimal consequence from sets of not necessarily variable-free assumptions [115].

Ignoring the propositional nature of a TM system, ALP can be regarded as a hybrid of JTMS and ATMS, combining the non-monotonic negative assumptions of JTMS and the positive assumptions of ATMS, and allowing both positive and negative conditions in both justifications and nogoods [92]. Other non-monotonic extensions of ATMS have been developed in [84, 163].

It should be noted that one difference between ATMS and ALP is the requirement in ATMS that only minimal sets of assumptions be recorded. This minimality of assumptions is essential for the computational efficiency of the ATMS. However, it is not essential for ALP, but can be imposed as an additional requirement when it is needed.

9 Conclusions and Future Work

In this paper we have surveyed a number of proposals for extending LP to perform abductive reasoning. We have seen that such extensions are closely linked with other extensions including NAF, integrity constraints, explicit negation, default reasoning, belief revision and argumentation.

Perhaps the most important link, from the perspective of LP, is that between default abduction and NAF. On the one hand, we have seen that default abduction generalises NAF, to include not only negative but also positive hypotheses, and to include general integrity constraints. On the other hand, we have seen that logic programs with abduction and integrity constraints can be transformed into logic programs with NAF without integrity constraints. We have also seen that, in the context of ELP with explicit negation, that NAF can be replaced by a priority ordering between clauses. The link between abduction and NAF includes both their semantics and their implementations.

The use of default abduction for NAF is a special case of abduction in general. The distinction between default and non-default abduction has been clarified. Semantics, proof procedures and transformations that respect this distinction have all been defined. However, more work is needed to study the integration of these two kinds of abduction in a common framework. The argumentation-based approach seems to offer a promising framework for such an integration.

We have seen the importance of clarifying the semantics of abduction and of defining a semantics that helps to unify the different forms of abduction, NAF, and default reasoning within a common framework. We have seen, in particular, that a proof procedure which is incorrect under one semantics (e.g. [54]) can be correct under another improved semantics (e.g. [39]). We have also introduced an argumentation-theoretic interpretation for the semantics of abduction applied to NAF, and we have seen that this interpretation can help to understand the relationships between different semantics.

The argumentation-theoretic interpretation of NAF has been abstracted and shown to unify and simplify the semantics of such different formalisms for default reasoning as default logic, autoepistemic logic and non-monotonic logic. In each case the standard semantics of these formalisms has been shown to be a special instance of a single abstract notion that a set of assumptions is a (stable) semantics if it does not attack itself but does attack all other assumptions it does not contain. The stable model semantics, generalised stable model semantics and answer set semantics are other special cases. We have seen that stable model semantics and its extensions have deficiencies which are avoided with admissibility, preferred extension, complete scenaria, weak stability, stable theory and acceptability semantics. Because these more refined semantics for LP can be defined abstractly for any argumentation-based framework, they automatically and uniformly provide improvements for the semantics of other formalisms for default reasoning.

Despite the many advances in the application of abduction to LP and to non-monotonic reasoning more generally, there is still much scope for further development. Important problems in semantics still need to be resolved. These include the problem of clarifying the role of integrity constraints in providing attacks and counterattacks in ALP.

The further development, clarification and simplification of the abstract argumentation-theoretic framework and its applications both to existing formalisms and to new formalisms for non-monotonic reasoning is another important direction for future research.

Of special importance is the problem of relating circumscription and the if-and-only-if completion semantics to the argumentation-theoretic approach. An important step in this direction may be the "common sense" axiomatisation of NAF [188] by Van Gelder and Schlipf, which augments the if-and-only-if completion with axioms of induction. The inclusion of induction axioms relates this approach to circumscription, whereas the rewriting of negative literals by new positive literals relates it to the abductive interpretation of NAF.

The development of systems that combine ALP and CLP is another important area that is still in its infancy. Among the results that might be expected from this development are more powerful systems that combine constructive abduction and constructive negation, and systems in which user-defined constraint handling rules might be formulated and executed efficiently as integrity constraints.

It is an important feature of the abductive interpretation of NAF that it possesses elegant and powerful proof procedures, which significantly extend SLDNF and which can be extended in turn to accommodate other abducibles and other integrity constraints. Different semantics for NAF require different proof procedures. It remains to be seen whether the inefficiency of proof procedures for the acceptability semantics, in particular, can somehow be avoided in practice.

We have seen that abductive proof procedures for LP can be extended to ALP. We have also seen that ALP can be reduced to LP by transformations. The comparative efficiency of the two different approaches to the implementation of ALP needs to be investigated further.

We have argued that the implementation of abduction needs to be considered within a broader framework of implementing knowledge assimilation (KA). We have seen that abduction can be used to assist the process of KA and that abductive hypotheses themselves need to be assimilated. Moreover, the general process of checking for integrity in KA might be used to check the acceptability of abductive hypotheses.

It seems that an efficient implementation of KA can be based upon combining two processes: backward reasoning both to generate abductive hypotheses and to test whether the input is redundant and forward reasoning both to test input for consistency and to test whether existing information is redundant. Notice that the abductive proof procedure for ALP already has this feature of interleaving backward and forward reasoning. Such implementations of KA need to be integrated with improvements of the abductive proof procedure considered in isolation.

We have seen that the process of belief revision also needs to be considered within a KA context. In particular, it could be useful to investigate relationships between the belief revision frameworks of [37, 65, 128, 129] and various integrity constraint checking and restoration procedures.

The extension of LP to include integrity constraints is useful both for abductive LP and for deductive databases. We have seen, however, that for many applications the use of integrity constraints with retractibles can be replaced by clauses with explicitly negated conclusions with priorities. Moreover, the use of explicit negation with priorities seems to have several advantages, including the ability both to represent and derive negative information, as well as to obtain the effect of NAF.

The relationship between integrity constraints with retractibles and explicit negation with priorities needs to be investigated further: To what extent does this relationship, which holds for abduction and default reasoning, hold for other uses of integrity constraints, such as those employed in deductive databases; and what are the implications of this relationship on the semantics and implementation of integrity constraints?

We have remarked upon the close links between the semantics of LP with abduction and the semantics of truth maintenance systems. The practical consequences of these links, both for building applications and for efficient implementations, need further investigation. What is the significance, for example, of the fact that conventional TMSs and ATMSs correspond only to the propositional case of logic programs?

We have seen the rapid development of the abduction-based argumentation-theoretic approach to non-monotonic reasoning. But argumentation has wider applications in areas such as law and practical reasoning more generally. It would be useful to see to what extent the theory of argumentation might be extended to encompass such applications. It would be especially gratifying, in particular, if such an extended argumentation theory might be used, not only to understand how one argument can defeat another, but also to indicate how conflicting arguments might be reconciled.

Acknowledgements

This research was supported by Fujitsu Research Laboratories and by the Esprit Basic Research Action Compulog II. The authors are grateful to Katsumi Inoue and Ken Satoh for helpful comments on an earlier draft, and to José Júlio Alferes, Phan Minh Dung, Paolo Mancarella and Luis Moniz Pereira for many helpful discussions.

References

[1] Akama, S., Answer set semantics and constructive logic with strong negation. Technical Report (1992)

[2] Alferes, J.J., Damasio, C.V., Pereira, L.M., Top-down query evaluation for well-founded semantics with explicit negation. *Proc. European Conference on Artificial Intelligence, ECAI '94, John Wiley,* Amsterdam (1994)

[3] Alferes, J.J., Dung, P.M., Pereira, L.M., Scenario semantics of extended logic programs. *Proc. 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning* MIT press (Pereira and Nerode eds.), Lisbon (1993) 334–348

[4] Alferes, J.J., Pereira, L.M., An argumentation-theoretic semantics based on non-refutable falsity. *Proc. International Conference on Logic Programming*, MIT Press, Workshop on Non-monotonic Extensions of Logic Programming (Dix, Pereira, Przymusinski eds.) Santa Margherita Ligure, Italy (1994)

[5] Alferes, J.J., Pereira, L.M., Contradiction in logic programs: when avoidance equal removal, Parts I and II. *Proc. 4th Int. Workshop on Extensions of Logic Programming* (R. Dyckhoff ed.), (1993) 7–26, Lecture Notes in AI 798, Springer-Verlag

[6] Allemand, D., Tanner, M., Bylander, T., Josephson, J., The computational complexity of abduction. *Artificial Intelligence* 49 (1991) 25–60

[7] Apt, K.R., Bezem, M., Acyclic programs. *Proc. 7th International Conference on Logic Programming*, MIT Press, Jerusalem (1990) 579–597

[8] Aravindan, C., Dung, P.M., Belief dynamics, abduction and databases. *Proc. 4th European Workshop on Logics in AI*, (1994), To appear in Lecture Notes in AI, Springer Verlag

[9] Baral, C., Gelfond, M., Logic programming and knowledge representation. To appear in *Journal of Logic Programming* (1994)

[10] Barbuti, R., Mancarella, P., Pedreschi, D., Turini, F., A transformational approach to negation in logic programming. *Journal of Logic Programming* 8 (1990) 201–228

[11] Bondarenko, A., Toni, F., Kowalski, R. A., An assumption-based framework for non-monotonic reasoning. *Proc. 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning* MIT press (Pereira and Nerode eds.), Lisbon (1993) 171–189

[12] Bondarenko, A., Dung, P.M., Kowalski, R. A., Toni, F., An abstract, argumentation-theoretic framework for default reasoning. In draft (1995)

[13] Brewka, G., Preferred subtheories: an extended logical framework for default reasoning. *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, Mi (1989) 1043–1048

[14] Brewka, G., An abductive framework for generalised logic programs. *Proc. 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning* MIT press (Pereira and Nerode eds.), Lisbon (1993) 349–364

[15] Brewka, G., Konolige, K., An abductive framework for general logic programs and other non-monotonic systems. *Proc. 13th International Joint Conference on Artificial Intelligence*, Chambery, France (1993) 9–15

[16] A. Brogi, E. Lamma, P. Mello, P. Mancarella, Normal logic programs as open positive programs. *Proc. ICSLP '92* (1992)

[17] Bry, F., Intensional updates: abduction via deduction. *Proc. 7th International Conference on Logic Programming*, MIT Press, Jerusalem (1990) 561–575

[18] Bürchert, H.-J., A resolution principle for constrained logics. *Artificial Intelligence* 66 (1994) 235–271

[19] Bürchert, H.-J., Nutt, W., On abduction and answer generation through constraint resolution. Technical Report DFKI, Kaiserslautern (1991)

[20] Casamayor, J., Decker, H., Some proof procedures for computational first-order theories, with an abductive flavour to them. *Proc. 1st Compulog-Net Workshop on Logic Programming in Artificial Intelligence*, Imperial College, London (1992)

[21] Chan, D., Constructive negation based on the completed database. *Proc. 5th International Conference and Symposium on Logic Programming*, Washington, Seattle (1988) 111–125

[22] Charniak, E., McDermott, D., *Introduction to artificial intelligence.* (Addison-Wesley, Menlo Park, Ca,1985)

[23] Chen, W., Warren, D.S., Abductive logic programming. Technical Report Dept. of Comp. Science, State Univ. of New York at Stony Brook (1989)

[24] Clark,K.L., Negation as failure. *Logic and Data Bases*, Gallaire and Minker eds., Plenum, New York(1978) 293–322

[25] Console, L., Theseider Dupré, D., Torasso, P. A Theory for diagnosis for incomplete causal models. *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, Mi (1989) 1311–1317

[26] Console, L., Theseider Dupré, D., Torasso, P. On the relationship between abduction and deduction. *Journal of Logic and Computation* 2(5) (1991) 661–690

[27] Console, L., Saitta, L., Abduction, induction and inverse resolution. *Proc. 1st Compulog-Net Workshop on Logic Programming in Artificial Intelligence*, Imperial College, London (1992)

[28] Console, L., Sapino, M.L., Theseider Dupré, D., The role of abduction in database view updating. To appear in *Journal of Intelligent Information Systems* (1994)

[29] Cox, P. T., Pietrzykowski, T., Causes for events: their computation and applications. *Proc. 8th International Conference on Automated Deduction*, CADE '86 (1992) 608–621

[30] Decker, H., Integrity enforcement on deductive databases. *Proc. EDS '86,* Charleston, SC (1986) 271–285

[31] Demolombe, R., Fariñas del Cerro, L., An inference rule for hypotheses generation. *Proc. 12th International Joint Conference on Artificial Intelligence*, Sidney (1991) 152–157

[32] Denecker, M., De Schreye, D., Temporal reasoning with abductive event calculus. *Proc. 1st Compulog-Net Workshop on Logic Programming in Artificial Intelligence*, Imperial College, London (1992)

[33] Denecker, M., De Schreye, D., On the duality of abduction and model generation. *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo (1992) 650–657

[34] Denecker, M., De Schreye, D., SLDNFA: an abductive procedure for normal abductive programs. *Proc. International Conference and Symposium on Logic Programming*, (1992) 686–700

[35] Denecker, M., De Schreye, D., Representing incomplete knowledge in abductive logic programming. *Proc. ILSP'93*, Vancouver (1993)

[36] Doyle, J., A truth maintenance system. *Artificial Intelligence* 12 (1979) 231–272

[37] Doyle, J., Rational belief revision. *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Mass. (1991) 163–174

[38] Drabent, W., What is failure? An approach to constructive negation. To appear in *Acta Informatica* (1994)

[39] Dung, P.M., Negation as hypothesis: an abductive foundation for logic programming *Proc. 8th International Conference on Logic Programming*, MIT Press, Paris (1991) 3–17

[40] Dung, P.M., An abductive foundation for non-monotonic truth maintenance. *Proc. 1st World Conference on Fundamentals of Artificial Intelligence*, Paris, de Glas ed. (1991)

[41] Dung, P.M., Acyclic disjunctive logic programs with abductive procedure as proof procedure. *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo (1992) 555–561

[42] Dung, P.M., An abductive procedure for disjunctive logic programming. Technical Report Dept. of Computing, Asian Institute of Technology (1992)

[43] Dung, P.M., Personal Communication (1992)

[44] Dung, P.M., On the acceptability of arguments and its fundamental role in non-monotonic reasoning and logic programming. To appear in *Artificial Intelligence* (1994) (Extended Abstract in *Proc. International Joint Conference on Artificial Intelligence*, (1993), 852–859)

[45] Dung, P.M., An argumentation semantics for logic programming with explicit negation. *Proc. 10th International Conference on Logic Programming*, MIT Press, Budapest (1993)

[46] Dung, P.M., Ruamviboonsuk, P., Well-founded reasoning with classical negation. *Proc. 1st International Workshop on Logic Programming and Nonmonotonic Reasoning* (Nerode, Marek and Subrahmanian eds.), Washington DC (1991) 120–135

[47] Dung, P.M., Kakas, A.C., Mancarella, P., Negation as failure revisited. Technical Report (1992)

[48] Eiter, T., Gottlob, G., The complexity of logic-based abduction. *Proc. 10th Symposium on Theoretical Aspects of Computing (STACS-93)*, Springer Verlag lecture Notes on Computer Science 665, eds. Enjalbert, P., Finkel, A., Wagner, K.W. (1993) 70–79 (Extended paper to appear in *Journal of the ACM*)

[49] Elkan, C., A rational reconstruction of non-monotonic truth maintenance systems. *Artificial Intelligence* 43 (1990) 219–234

[50] Eshghi, K., Abductive planning with event calculus. *Proc. 5th International Conference and Symposium on Logic Programming*, Washington, Seattle (1988) 562–579

[51] Eshghi, K., Diagnoses as stable models. *Proc. 1st International Workshop on Principles of Diagnosis*, Menlo Park, Ca (1990)

[52] Eshghi, K., A tractable set of abduction problems. *Proc. 13th International Joint Conference on Artificial Intelligence*, Chambery, France (1993) 3–8

[53] Eshghi, K., Kowalski, R.A., Abduction through deduction. Technical Report Department of Computing, Imperial College, London (1988)

[54] Eshghi, K., Kowalski, R.A., Abduction compared with negation by failure. *Proc. 6th International Conference on Logic Programming*, MIT Press, Lisbon (1989) 234–255

[55] Evans, C.A., Negation as failure as an approach to the Hanks and McDermott problem. *Proc. 2nd International Symposium on Artificial intelligence*, Monterrey, Mexico (1989)

[56] Evans, C.A., Kakas, A.C., Hypothetico-deductive reasoning. *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo (1992) 546–554

[57] Finger, J.J., Genesereth, M.R., RESIDUE: a deductive approach to design synthesis. Technical Report no. CS-85-1035, Stanford University (1985)

[58] Fujita, M., Hasegawa, R., A model generation theorem prover in KL1 using a ramified-stack algorithm. *Proc. 8th International Conference on Logic Programming*, MIT Press, Paris (1991) 535–548

[59] Fujiwara, Y., Honiden, S., Relating the TMS to Autoepistemic Logic. *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, Mi (1989) 1199–1205

[60] Fung, T. H., *Theorem proving approach with constraint handling and its applications on databases*. MSc Thesis, Imperial College, London (1993)

[61] Gabbay, D.M., Abduction in labelled deductive systems. A conceptual abstract. *Proc. of the European Conference on Symbolic and Quantitative Approaches for uncertainty '91,* Springer Verlag lecture Notes on Computer Science 548, eds. R. Kruse and P. Siegel (1991) 3–12

[62] Gabbay, D.M., Kempson, R.M., Labelled abduction and relevance reasoning. *Workshop on Non-Standard Queries and Non-Standard Answers*, Toulose, France (1991)

[63] Gabbay, D.M., Kempson, R.M., Pitts, J., Labelled abduction and relevance reasoning. *Non-standard queries and answers*, R.Demolombe and T. Imielinski, eds., Oxford University press (1994) 155–185

[64] Gaifman, H., Shapiro, E., Proof theory and semantics of logic programming. *Proc. LICS'89*, IEEE Computer Society Press (1989) 50–62

[65] Gärdenfors, P., *Knowledge in flux: modeling the dynamics of epistemic states.* (MIT Press, Cambridge, Ma,1988)

[66] Geffner, H., Casual theories for non-monotonic reasoning. *Proc. AAAI '90* (1990)

[67] Geffner, H., Beyond negation as failure. *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Mass. (1991) 218–229

[68] Gelfond, M., Lifschitz, V., The Stable model semantics for logic programs. *Proc. 5th International Conference and Symposium on Logic Programming*, Washington, Seattle (1988) 1070–1080

[69] Gelfond, M., Lifschitz, V., Logic programs with classical negation. *Proc. 7th International Conference on Logic Programming*, MIT Press, Jerusalem (1990) 579–597

[70] Goebel, R., Furukawa, K., Poole, D., Using definite clauses and integrity constraints as the basis for a theory formation approach to diagnostic reasoning. *Proc. 3rd International Conference on Logic Programming*, MIT Press, London (1986) Springer Verlag Lecture Notes in Computer Science 225, 211–222

[71] Giordano, L., Martelli, A., Generalized stable model semantics, truth maintenance and conflict resolution. *Proc. 7th International Conference on Logic Programming*, MIT Press, Jerusalem (1990) 427–411

[72] Giordano, L., Martelli, A., Sapino, M. L., A semantics for Eshghi and Kowalski's abductive procedure. *Proc. 10th International Conference on Logic Programming*, MIT Press, Budapest (1993) 586–600

[73] Hanks, S., McDermott, D., Default reasoning, non-monotonic logics, and the frame problem. *Proc. 8th AAAI '86,* Philadelphia (1986) 328–333

[74] Hanks, S., McDermott, D., Non-monotonic logics and temporal projection. *Artificial Intelligence* 33 (1987)

[75] Frühwirth, T., Constraint simplification rules. Technical Report ECRC-92-18 (1992)

[76] Hasegawa, R., Fujita, M., Parallel theorem provers and their applications. *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo (1992) 132–154

[77] Hobbs, J.R., Stickel, M., Appelt, D., Martin, P., Interpretation as abduction. Technical Report 499, Artificial Intelligence Center, Computing and Engineering Sciences Division, Menlo Park, Ca (1990)

[78] Hobbs, J.R., An integrated abductive framework for discourse interpretation. *Proc. AAAI Symposium on Automated Abduction,* Stanford (1990) 10–12

[79] Inoue, K., An abductive procedure for the CMS/ATMS. *Proc. European Conference on Artificial Intelligence, ECAI '90* International Workshop on Truth Maintenance, Stockholm, Springer Verlag Lecture notes in Computer Science (1990)

[80] Inoue, K., Extended logic programs with default assumptions. *Proc. 8th International Conference on Logic Programming*, MIT Press, Paris (1991) 490–504

[81] Inoue, K., Hypothetical reasoning in logic programs. *Journal of Logic Programming* 18 (1994) 191–227

[82] Inoue, K., Koshimura, M., Hasegawa, R., Embedding negation as failure into a model generation theorem prover. *Proc. 11th International Conference on Automated Deduction*, CADE '92, Saratoga Springs, NY (1992)

[83] Inoue, K., Ohta, Y., Hasegawa, R., Nakashima, M., Hypothetical reasoning systems on the MGTP. Technical Report ICOT, Tokyo (in Japanese) (1992)

[84] Junker, U., A correct non-monotonic ATMS. *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, Mi (1989) 1049–1054

[85] Kakas, A. C., Deductive databases as theories of belief. Technical Report Logic Programming Group, Imperial College, London (1991)

[86] Kakas, A.C., On the evolution of databases. Technical Report Logic Programming Group, Imperial College, London (1991)

[87] Kakas, A.C., Default reasoning via negation as failure. *Proc. ECAI-92 workshop on "Foundations of Knowledge Representation and Reasoning"*, Lecture Notes in AI 810, Springer Verlag, eds. Lakemeyer and Nebel (1992)

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

[89] Kakas, A. C., Mancarella, P., Anomalous models and abduction. *Proc. 2nd International Symposium on Artificial intelligence*, Monterrey, Mexico (1989)

[90] Kakas, A. C., Mancarella, P., Generalized Stable Models: a Semantics for Abduction. *Proc. 9th European Conference on Artificial Intelligence, ECAI '90,* Stockolm (1990) 385–391

[91] Kakas, A. C., Mancarella, P., Database updates through abduction. *Proc. 16th International Conference on Very Large Databases, VLDB'90*, Brisbane, Australia (1990)

[92] Kakas, A. C., Mancarella, P., On the relation of truth maintenance and abduction. *Proc. of the 1st Pacific Rim International Conference on Artificial Intelligence, PRICAI'90*, Nagoya, Japan (1990)

[93] Kakas, A. C., Mancarella, P., Abductive LP. *Proc. NACLP '90,* Workshop on Non-Monotonic Reasoning and Logic Programming, Austin, Texas (1990)

[94] Kakas, A. C., Mancarella, P., Knowledge assimilation and abduction. *Proc. European Conference on Artificial Intelligence, ECAI '90* International Workshop on Truth Maintenance, Stockholm, Springer Verlag Lecture notes in Computer Science (1990)

[95] Kakas, A. C., Mancarella, P., Preferred extensions are partial stable models. *Journal of Logic Programming*, 14(3,4):341–348 (1993)

[96] Kakas, A. C., Mancarella, P., Negation as stable hypotheses. *Proc. 1st International Workshop on Logic Programming and Nonmonotonic Reasoning* (Nerode, Marek and Subrahmanian eds.), Washington DC (1991)

[97] Kakas, A. C., Mancarella, P., Stable theories for logic programs. *Proc. ISLP '91,* San Diego (1991)

[98] Kakas, A. C., Mancarella, P., Constructive abduction in logic programming. Technical Report Dipartimento di Informatica, Università di Pisa (1993)

[99] Kakas, A. C., Mancarella, P., Dung, P.M., The acceptability semantics for logic programs. *Proc. 11th International Conference on Logic Programming*, MIT Press, Santa Margherita Ligure, Italy (1994) 504–519

[100] Kakas, A. C., Michael, A., Scheduling through abduction. *Proc. ICLP'93 Post Conference workshop on Abductive Reasoning* (1993)

[101] Kakas, A. C., Michael, A., Integrating abductive and constraint logic programming. To appear in *Proc. International Logic Programming Conference*, (1995)

[102] deKleer, J., An assumption-based TMS. *Artificial Intelligence* 32 (1986)

[103] Konolige, K., A general theory of abduction. *Spring Symposium on Automated Abduction, Standford University* (1990) 62–66

[104] Konolige, K., Using defualt and causal reasoning in diagnosis. *Proc. 3rd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge (1992)

[105] Kowalski, R.A., *Logic for problem solving.* (Elsevier, New York,1979)

[106] Kowalski, R.A., Belief revision without constraints. *Computational Intelligence* 3(3), (1987)

[107] Kowalski, R.A., Problems and promises of computational logic. *Proc. Symposium on Computational Logic*, Lloyd ed., Springer Verlag Lecture Notes in Computer Science (1990)

[108] Kowalski, R.A., Database updates in the event calculus. *Journal of Logic Programming* 12 (1992) 121–146

[109] Kowalski, R. A., A dual form of logic programming. Lecture Notes, Workshop in Honour of Jack Minker, University of Maryland, November 1992

[110] Kowalski, R.A., Logic without model theory. To appear in *What is a Logical System?* (D. Gabbay, ed.) Oxford University Press (1994)

[111] Kowalski, R.A., Sadri, F., Knowledge representation without integrity constraints. Technical Report Department of Computing, Imperial College, London (1988)

[112] Kowalski, R.A., Sadri, F., Logic programs with exceptions. *Proc. 7th International Conference on Logic Programming*, MIT Press, Jerusalem (1990) 598–613

[113] Kowalski, R.A., Sergot, M., A logic-based calculus of events. *New Generation Computing* 4 (1986) 67–95

[114] Kunifuji, S., Tsurumaki, K., Furukawa, K., Consideration of a hypothesis-based reasoning system. *Journal of Japanese Society for Artificial Intelligence* 1(2) (1986) 228–237

[115] Lamma, E., Mello, P., An assumption-based truth maintenance system dealing with non ground justifications. *Proc. 1st Compulog-Net Workshop on Logic Programming in Artificial Intelligence*, Imperial College, London (1992)

[116] Lever, J. M., *Combining induction with resolution in logic programming.* PhD Thesis, Department of Computing, Imperial College, London (1991)

[117] Levesque, H.J., A knowledge-level account of abduction. *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, Mi (1989) 1061–1067

[118] Lloyd, J.W., Topor, R.W., A basis for deductive database system. *Journal of Logic Programming* 2 (1985) 93–109

[119] McDermott, D., Nonmonotonic logic II: nonmonotonic modal theories. *JACM* 29(1) (1982)

[120] Maim, E., Abduction and constraint logic programming. *Proc. European Conference on Artificial Intelligence, ECAI '92* Vienna, Austria (1992)

[121] Makinson, D., General theory of cumulative inference. *Proc. 2nd International Workshop on Monmonotonic reasoning*, Springer Verlag Lecture Notes in Computer Science 346 (1989)

[122] Manthey, R., Bry, F., SATCHMO: a theorem prover implemented in Prolog. *Proc. 9th International Conference on Automated Deduction*, CADE '88, Argonne, Illinois (1988) 415–434

[123] Marek, W., Truszczynski, M., Stable semantics for logic programs and default theories. *Proc. NACLP '89* (1989) 243–256

[124] Minker, J., On indefinite databases and the closed world assumption. *Proc. 6th International Conference on Automated Deduction*, CADE '82, New York, Springer Verlag Lecture Notes in Computer Science 138 (1982) 292-308

[125] Miyaki, T., Kunifuji, S., Kitakami, H., Furukawa, K., Takeuchi, A., Yokota, H., A knowledge assimilation method for logic databases. *International Symposium on Logic Programming*, Atlantic City, NJ (1984) 118–125

[126] Moore, R., Semantical considerations on non-monotonic logic. *Artificial Intelligence* 25 (1985)

[127] Morris, P. H., The anomalous extension problem in default reasoning. *Artificial Intelligence* 35 (1988) 383–399

[128] Nebel, B., A knowledge level analysis of belief revision. *Proc. 1st International Conference on Principles of Knowledge Representation and Reasoning*, Toronto (1989) 301–311

[129] Nebel, B., Belief revision and default reasoning: syntax-based approaches. *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Mass. (1991) 417–428

[130] Pearce, D., Wagner, G., Logic programming with strong negation. *Proc. Workshop on Extensions of Logic Programming,* Springer Verlag Lecture Notes in Computer Science (1991)

[131] Pearl, J., Embracing causality in formal reasoning. *Proc. AAAI '87,* Washington, Seattle (1987) 360–373

[132] Pearl, J., *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* (Morgan Kaufmann, San Mateo, California,1988)

[133] Peirce, C.S., *Collected papers of Charles Sanders Peirce.* Vol.2, 1931–1958, Hartshorn et al. eds., Harvard University Press

[134] Pereira, L.M., Alferes, J.J., Aparicio, J.N., Contradiction removal semantics with explicit negation *Proc. Applied Logic Conference*, Amsterdam (1992)

[135] Pereira, L.M., Alferes, J.J., Well-founded semantics for logic programs with explicit negation. *Proc. 92 European Conference on Artificial Intelligence, ECAI 'Vienna,Austria* 1992 (1)02–106

[136] Pereira, L.M., Aparicio, J.N., Alferes, J.J., Non-monotonic reasoning with well-founded semantics. *Proc. 8th International Conference on Logic Programming*, MIT Press, Paris (1991)

[137] Pereira, L.M., Aparicio, J.N., Alferes, J.J., Contradiction removal within well-founded semantics. *Proc. 1st International Workshop on Logic Programming and Nonmonotonic Reasoning* (Nerode, Marek and Subrahmanian eds.), Washington DC (1991)

[138] Pereira, L.M., Aparicio, J.N., Alferes, J.J., Derivation procedures for extended stable models. *Proc. 12th International Joint Conference on Artificial Intelligence*, Sidney (1991) 863–868

[139] Pereira, L.M., Aparicio, J.N., Alferes, J.J., Counterfactual reasoning based on revising assumptions. *Proc. ISLP '91,* San Diego (1991)

[140] Pereira, L.M., Damasio, C.V., Alferes, J.J., Diagnosis and debugging as contradiction removal. *Proc. 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning* MIT press (Pereira and Nerode eds.), Lisbon (1993) 316–330

[141] Pimentel, S. G., Cuadrado, J. L., A truth maintenance system based on stable models. *Proc. NACLP '89* (1989)

[142] Pople, H. E. Jr., On the mechanization of abductive logic. *Proc. 3rd International Joint Conference on Artificial Intelligence*, (1973) 147–152

[143] Poole, D., On the comparison of theories: preferring the most specific explanation. *Proc. 9th International Joint Conference on Artificial Intelligence*, Los Angeles, Ca (1985) 144–147

[144] Poole, D., Variables in hypotheses. *Proc. 10th International Joint Conference on Artificial Intelligence*, Milan (1987) 905–908

[145] Poole, D., A logical framework for default reasoning. *Artificial Intelligence* 36 (1988) 27–47

[146] Poole, D., Representing knowledge for logic-based diagnosis. *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo (1988) 1282–1290

[147] Poole, D., Explanation and prediction: an architecture for default and abductive reasoning. *Computational Intelligence Journal* 5 (1989) 97–110

[148] Poole, D., Logic programming, abduction and probability. *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo (1992) 530–538

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

[150] 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 (1987) 331–352

[151] Preist, C., Eshghi, K., Consistency-based and abductive diagnoses as generalised stable models. *Proc. International Conference on Fifth Generation Computer Systems*, Tokyo (1992) 514–521

[152] Przymusinski, T.C., On the declarative and procedural semantics of logic programs. *Journal of Automated Reasoning* 5 (1989) 167–205

[153] Przymusinski, T.C., Extended stable semantics for normal and disjunctive programs. *Proc. 7th International Conference on Logic Programming*, MIT Press, Jerusalem (1990) 459–477

[154] T. Przymusinski, Semantics of disjunctive logic programs and deductive databases. *Proc. DOOD '91* (1991)

[155] Reggia, J., Diagnostic experts systems based on a set-covering model. *International Journal of Man-Machine Studies* 19(5) (1983) 437–460

[156] Reinfrank, M., Dessler, O., On the relation between truth maintenance and nonmonotonic logics. *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, Mi (1989) 1206–1212

[157] Reiter, R., On closed world data bases. *Logic and Databases*, Gallaire and Minker eds., Plenum, New York(1978) 55–76

[158] Reiter, R., A Logic for default reasoning. *Artificial Intelligence* 13 (1980) 81–132

[159] Reiter, R., A theory of diagnosis from first principle. *Artificial Intelligence* 32 (1987)

[160] Reiter, R., On integrity constraints. *Proc. 2nd Conference on Theoretical Aspects of Reasoning about Knowledge*, Moshe Y. Vardi ed., Pacific Grove, California (1988)

[161] Reiter, R., On asking what a database knows. *Proc. Symposium on Computational Logic*, Lloyd ed., Springer Verlag Lecture Notes in Computer Science (1990)

[162] Reiter, R., deKleer, J., Foundations of assumption-based truth maintenance systems: preliminary report. *Proc. AAAI '87,* Washington, Seattle (1987) 183–188

[163] Rodi, W.L., Pimentel, S.G., A non-monotonic ATMS using stable bases. *Proc. 2nd International Conference on Principles of Knowledge Representation and Reasoning*, Cambridge, Mass. (1991)

[164] Saccà, D., Zaniolo, C., Stable models and non determinism for logic programs with negation *Proc. ACM SIGMOD-SIGACT Symposium on Principles of Database Systems* (1990) 205–217

[165] Sadri, F., Kowalski, R.A., An application of general purpose theorem-proving to database integrity. *Foundations of Deductive Databases and Logic Programming*, Minker ed., Morgan Kaufmann Publishers, Palo Alto (1987) 313–362

[166] Sakama, C., Inoue, K., Negation in disjunctive logic programs. *Proc. 10th International Conference on Logic Programming*, MIT Press, Budapest (1993) 703–719

[167] Sakama, C., Inoue, K., On the equivalence between disjunctive and abductive logic programs. *Proc. 11th International Conference on Logic Programming*, MIT Press, Santa Margherita Ligure, Italy (1994) 489–503

[168] Sato, T., Completed logic programs and their consistency. *Journal of Logic Programming* 9 (1990) 33–44

[169] Satoh, K., A top-down proof procedure for default logic by using abduction. *Proc. European Conference on Artificial Intelligence, ECAI '94,* Amsterdam (1994)

[170] Satoh, K., Iwayama, N., Computing abduction using the TMS. *Proc. 8th International Conference on Logic Programming*, MIT Press, Paris (1991) 505–518

[171] Satoh, K., Iwayama, N., A correct top-down proof procedure for general logic programs with integrity constraints. *Proc. 3rd International Workshop on Extensions of Logic Programming* (1992) 19–34

[172] Satoh, K., Iwayama, N., A query evaluation method for abductive logic programming. *Proc. International Conference and Symposium on Logic Programming*, (1992) 671–685

[173] Sattar, A., Goebel, R., Using crucial literals to select better theories. Technical Report Dept. of Computer Science, University of Alberta, Canada (1989)

[174] Selman, B., Levesque, H.J., Abductive and default reasoning: a computational core. *Proc. AAAI 90* 1990 ( )343–348

[175] Sergot,M., A query-the-user facility for logic programming. *Integrated Interactive Computer Systems*, Degano and Sandwell eds., North Holland Press (1983) 27–41

[176] Shanahan, M., Prediction is deduction but explanation is abduction. *Proc. 11th International Joint Conference on Artificial Intelligence*, Detroit, Mi (1989) 1055–1060

[177] Simari, G.R., Loui, R.P, A mathematical treatment of defeasible reasoning and its implementation. *Artificial Intelligence* 53 (1992) 125–157

[178] Sperber, D., Wilson, D., *Relevance: communication and cognition.* (Basil Blanckwell Ltd, Oxford, UK,1986)

[179] Stickel, M.E., A prolog-like inference system for computing minimum-cost abductive explanations in natural-language interpretation. *Proc. International Computer Science Conference (Artificial Intelligence: Theory and Applications)*, Honk Kong, Lassez and Chin eds. (1988) 343–350

[180] Stickel, M.E., Rationale and methods for abductive reasoning in natural language interpretation. *Proc. International Scientific Symposium on Natural Language and Logic*, Hamburg, Germany, Springer Verlag Lecture Notes in Artificial Intelligence (1989) 233–252

[181] Teusink, F., Using SLDFA-resolution with abductive logic programs. *ILPS '93* post-conference workshop "Logic Programming with Incomplete Information" (1993)

[182] Toni, F., Kakas, A. C., Computing the acceptability semantics. To appear in *Proc. International Workshop on Logic Programming and Nonmonotonic Reasoning* (1995)

[183] Toni, F., Kowalski, R. A., Reduction of abductive logic programs to normal logic programs. To appear in *Proc. International Logic Programming Conference*, (1995)

[184] Toni, F., A theorem-proving approach to job-shop scheduling. Technical Report Imperial College, London (1994)

[185] Torres, A., Negation as failure to support. *Proc. 2nd International Workshop on Logic Programming and Nonmonotonic Reasoning* MIT press (Pereira and Nerode eds.), Lisbon (1993) 223–243

[186] Van Belleghem, K., Denecker, M., De Schreye, D., Representing continuos change in the abductive event calculus. *Proc. 11th International Conference on Logic Programming*, MIT Press, Santa Margherita Ligure, Italy (1994) 225–239

[187] Van Gelder, A., Ross, K.A., Schlipf, J.S., Unfounded sets and the well-founded semantics for general logic programs. *Proc. ACM SIGMOD-SIGACT, Symposium on Principles of Database Systems* (1988)

[188] Van Gelder, K.A., Schlipf, J.S., Commonsense axiomatizations for logic programs. *Journal of Logic Programming* 17 (1993) 161–195

[189] Wallace, M., Negation by constraints: a sound and efficient implementation of negation in deductive databases. *Proc. 4th Symposium on Logic Programming*, San Francisco (1987)