Abductive Logic Programming [1]

A.C. Kakas
Department of Computer Science,
University of Cyprus,
75 Kallipoleos Street,
Nicosia P.O. Box 537, Cyprus.
`antonis@saturn.cca.ucy.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). We outline the general framework of Abduction and its applications to Knowledge Assimilation and Default Reasoning; and we introduce an argumentation-theoretic approach to the use of abduction as an interpretation for Negation as Failure. We also analyse the links 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.

1 Introduction

This paper is a survey and analysis of work on the extension of logic programming to perform abductive reasoning. 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

[1]Appeared in *Journal of Logic and Computation,* Vol 2 No. 6, pp. 719–770 (1993).

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 negative or incomplete information, the evolution of knowledge, and the semantics of logic programming and its extensions. We also introduce a novel argumentation-theoretic interpretation of abduction applied to negation as failure.

The philosopher Pierce first introduced the notion of abduction. In [95] 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"[95].

Abduction is widely used in common-sense reasoning, for instance in diagnosis, to reason from effect to cause [9, 101]. We consider here an example drawn from [94].

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-last-night* is a possible explanation, i.e. a set of hypotheses that together with the explicit knowledge in T implies the given observation. *Sprinkler-was-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 [14]. 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 [104].

- 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 [123] 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 [35] 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 [124], 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 [5, 9, 39, 52, 102, 106, 128].

Cox and Pietrzykowski [15] 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 example, 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

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

are.

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

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

$\{q, r\}$ is a non-minimal explanation for p while $\{q\}$ is minimal.

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 [15] construct hypotheses from the "dead ends" of linear resolution proofs. Finger and Genesereth [36] generate "deductive solutions to design problems" using the "residue" left behind in resolution proofs. Poole, Goebel and Aleliunas [107] also use linear resolution to generate hypotheses.

In contrast, the ATMS [71] computes abductive explanations bottom-up. The ATMS can be regarded as a form of hyper-resolution, augmented with subsumption, for propositional logic programs [118]. Lamma and Mello [81] 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 [17] and Gaifman and Shapiro [41].

Abduction can also be applied to logic programming. A **general logic program** is a set of Horn clauses extended by negation as failure [11], 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 $\sim A_i$ [2], A is an atom and each variable occurring in the clause is implicitly universally quantified. Abduction can be computed in logic programming by extending SLD and SLDNF [10, 32, 33, 64, 67]. 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 [124]. 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

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

advantage of ensuring that all explanations will be basic.

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 et al. [2] and Reggia [111] present a mathematical characterisation, where abduction is defined over sets of observations and hypotheses, in terms of coverings and parsimony.

- Levesque [83] 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, 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 logic programming, which restricts interpretations to Herbrand interpretations. This restriction to Herbrand interpretations means that interpretations are purely syntactic objects, which have no bearing on the correspondence between language and "reality".

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 [82].

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 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 A is the set of abducible predicates, i.e. $\Delta \subseteq A$ [3].

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:

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

[3]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.

These definitions have been proposed in the case where the theory is a logic program P by Kowalski and Sadri [76] and Lloyd and Topor [84] respectively, where KB is the Clark completion [11] of P.

Another view of integrity constraints [60, 63, 74, 116, 117] 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 is true about the knowledge base rather than what is true about the world modelled by the knowledge base. When later we consider abduction in logic programming (see sections 4,5), integrity satisfaction will be understood in a sense which is stronger than consistency, weaker than theoremhood, and arguable 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. [16, 76, 84]). 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 [76] 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 [33, 66, 67]. 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.

Abduction can be used to generate causal explanations for **fault diagnosis** (see for example [12, 108]). In medical diagnosis, for example, the candidate hypotheses are the possible causes (diseases), and the observations are the symptoms to be explained [105, 111]. Abduction can also be used for model-based diagnosis [31, 115]. 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** [15]. 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 [9, 40, 53, 127]. 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 [30, 125].

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** [67, 72, 80, 89]. The assimilation of a new datum can be performed by adding to the theory new hypotheses that are explanations for the datum. **Database view updates** [6, 64] 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. We will discuss knowledge assimilation in greater detail in section 2.

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 [32, 104]. As Poole [104] 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 logic programming is a form of default reasoning, its interpretation by means of abduction will be discussed in section 4.

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 knowledge, and the new KB which arises as a result [72].

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 = $\mathrm{KB}_1 \cup \mathrm{KB}_2$ 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 [67, 72, 80]. In [80] 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. Consider for example a problem of temporal reasoning formulated in the Event Calculus [79]. This contains an axiom that expresses the persistence of a property P from the time that it is initiated by an event E to a later time T:

$$\begin{array}{lll} \mathit{holds\text{-}at}(P, T_2) & \leftarrow & \mathit{happens}(E, T_1), \\ & & T_1 < T_2, \\ & & \mathit{initiates}(E, P), \\ & & \mathit{persists}(T_1, P, T_2). \end{array}$$

New information about the predicate *holds-at* can be assimilated by adding an explanation in terms of some event that generates this property together with an appropriate assumption that the property persists [30, 62, 125]. This has the additional effect that the new KB will imply that the property holds until it is terminated in the future [125]. This way of assimilating new information can also be used to resolve conflicts between the current KB and the new information [62, 125]. Suppose for example that the current KB contains the fact (expressed informally) "Mary has book1 at time t_0" and that the persistence axiom predicts that "Mary has book1 at time t_1" where $t_0 < t_1$. The new information "John has book1 at time t_1" contradicts the prediction, and cannot be added explicitly to the KB. It is however possible to remove the contradiction by adding the explanation that an event has happened where "Mary gives John book1 between t_0 and t_1".

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 2.1
Suppose the current KB contains

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

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.

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 any abductive hypotheses generated in case (4).

Any violation of integrity detected in case (3) can be remedied in several ways [72]. 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 [64]. Any violation of integrity is dealt with by changing the extensional part of the database.

Example 2.2
Suppose the current KB consists of the clauses

$$\begin{aligned}
&sibling(x,y) \leftarrow parent(z,x), parent(z,y)\\
&parent(x,y) \leftarrow father(x,y)\\
&parent(x,y) \leftarrow mother(x,y)\\
&father(John, Mary)\\
&mother(Jane, Mary)
\end{aligned}$$

together with the integrity constraints

$$\begin{aligned}
x = y &\leftarrow father(x,z),\ father(y,z)\\
x = y &\leftarrow mother(x,z),\ mother(y,z)
\end{aligned}$$

where *sibling* and *parent* are view predicates, *father* and *mother* are extensional, and $=$ is a "built-in" predicate such that

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

Suppose the view update

$$\text{insert } sibling(Mary, Bob)$$

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

$$\begin{aligned}
&\text{insert } father(John, Bob)\\
&\text{insert } mother(Jane, Bob)
\end{aligned}$$

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

$$\text{insert } mother(Joan, Bob).$$

The general problem of belief revision has been studied formally in [42, 91, 92, 21]. 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 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 [61]. It should also 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 2.2.

KA and belief revision are also related to truth maintenance systems. We will discuss truth maintenance and its relationship with abduction in section 6.

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 [114]. 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

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

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 et al. [107] and Poole [104] proposes 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 (theorem 5.1 in [104]) 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

$$w(x)$$

[4]We use the notation x to indicate a tuple of variables $x_1, \ldots, x_n$.

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]

$$\begin{aligned} \mathcal{F} \quad &= \{ \quad penguin(x) \rightarrow bird(x), \\ &\qquad\; penguin(x) \rightarrow \neg\, fly(x), \\ &\qquad\; penguin(Tweety), \\ &\qquad\; bird(John)\} \\ D \quad &= \{ \quad bird(x) \rightarrow fly(x)\,\}. \end{aligned} \tag{1}$$

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 $\mathcal{F}'$.

[5]Here, we use the conventional notation of first-order logic, rather than logic programming form. However, as in logic programming 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.

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.

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:

$$\frac{\alpha(x):\mathrm{M}\beta(x)}{\gamma(x)}$$

are translated into "facts", which are implications

$$\alpha(x) \wedge M_\beta(x) \rightarrow \gamma(x)$$

where M_β is a new predicate, and $M_\beta(x)$ is a default formula (abducible). An integrity constraint

$$\neg\, \beta(x) \rightarrow \neg\, M_\beta(x)$$

is needed to link the new predicate appropriately with the predicate β. A second integrity constraint

$$\neg\, \gamma(x) \rightarrow \neg\, M_\beta(x)$$

is needed to prevent the application of the contrapositive

$$\neg\, \gamma(x) \wedge M_\beta(x) \rightarrow \neg\, \alpha(x)$$

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 example can be dealt with correctly in certain extensions of default logic, such as Cumulative Default Logic [85], 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 [49, 50]. This can be reduced to the propositional logic program

$$\begin{aligned} &\textit{alive-after-load-wait-shoot} \leftarrow \textit{alive-after-load-wait}, \\ &\qquad\qquad\qquad\qquad\qquad\sim \textit{abnormal-alive-shoot} \end{aligned}$$

$$\begin{aligned} &\textit{loaded-after-load-wait} \leftarrow \textit{loaded-after-load}, \\ &\qquad\qquad\qquad\qquad\quad\sim \textit{abnormal-loaded-wait} \\ &\textit{abnormal-alive-shoot} \leftarrow \textit{loaded-after-load-wait} \\ &\textit{alive-after-load-wait} \\ &\textit{loaded-after-load}, \end{aligned}$$

As argued in [90], these clauses can be simplified further: First, the facts *alive-after-load-wait* and *loaded-after-load* can be eliminated by resolving them against the corresponding conditions of the first two clauses, giving

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

Then the atom *loaded-after-load-wait* can be resolved away from the second and third clauses leaving the two clauses

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

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 the two propositional sentences above and whose defaults are the two normal defaults

$$\frac{: \mathrm{M} \sim q}{\sim q} \qquad \frac{: \mathrm{M} \sim r}{\sim r}$$

has two extensions: one in which $\sim r$, and therefore q holds; and one in which $\sim q$, 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 [90] showed that the default theory which has no facts but contains the two non-normal defaults

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

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 [32], Evans [34] and Apt and Bezem [3] 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 logic programming. Moreover, [32] and [62] 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 logic programming, 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 logic programming 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 logic programming 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 and 5 we will see other ways of performing default reasoning in logic programming. In section 4 we will introduce negation as failure as a form of default reasoning, and we will study its relationship with abduction. In section 5 we will consider an extended logic programming framework that contains clauses with negative conclusions and avoids the use of explicit integrity constraints, in some cases.

4 Negation as Failure as Abduction

We noted in the previous section that default reasoning can be performed by means of abduction in logic programming by explicitly introducing abducibles into rules. Default reasoning can also be performed with the use of negation as failure (NAF) [11] 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.

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 [32] and [33], 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 substituted for by $p^*(t)$.

- I^* is a set of all integrity constraints of the form [6]:

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

The semantics of the abductive framework $\langle P^*, A^*, I^* \rangle$, in terms of **extensions** $P^* \cup \Delta$ 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 Q^* 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 t,

$$\begin{array}{rcl} P^* \cup \Delta & \not\models & t \wedge t^*, \text{ and} \\ P^* \cup \Delta & \models & t \text{ or } P^* \cup \Delta \models t^*. \end{array}$$

Eshghi and Kowalski [33] show that there is a one to one correspondence between stable models [45] of P and abductive extensions of P^*. We recall the definition of stable model:

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

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

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.

Let P be a general logic program, and assume that all the clauses in P are variable-free [7]. 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 $\Delta = \{d^* \mid d \text{ is a variable-free atom}, d \notin M\}$.
- For any Δ such that $P^* \cup \Delta$ satisfies I^*, there is a stable model M of P, where $M = \{d \mid d \text{ is a variable-free atom}, d^* \notin \Delta\}$.

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 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 more syntactic 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 not so much a semantics as a non-constructive specification of what should be computed. The computation itself is performed by means of a proof procedure.

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

4.2 An abductive proof procedure for logic programming

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 [104, 107] provides such an implementation of abduction by means of a resolution based proof procedure.

In their study of NAF through abduction Eshghi and Kowalski [33] 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).

The abductive proof procedure interleaves two types of computation. The first type, referred to as the **abductive phase**, is standard SLD- resolution, that generates (negative) hypotheses and adds them to the set of abducibles being generated, while the second type, referred to as the **consistency phase** [8], incrementally checks that the hypotheses satisfy the integrity constraints for NAF, I^*. 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.

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.

[8] 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.

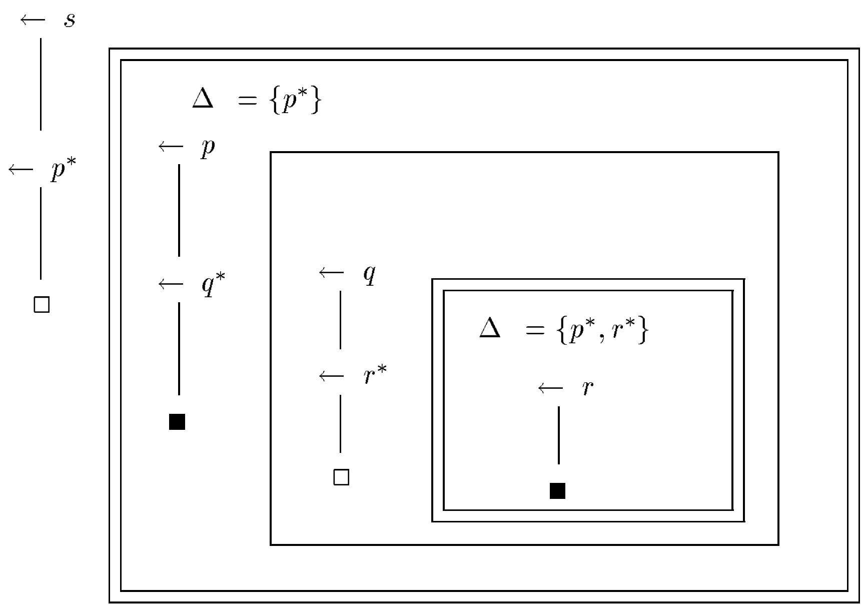

Figure 1: computation for example 4.2

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 significantly. 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 [67]. 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 6). 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 [64, 65, 66] (see section 5.2).

To see how the abductive proof procedure extends SLDNF, consider the

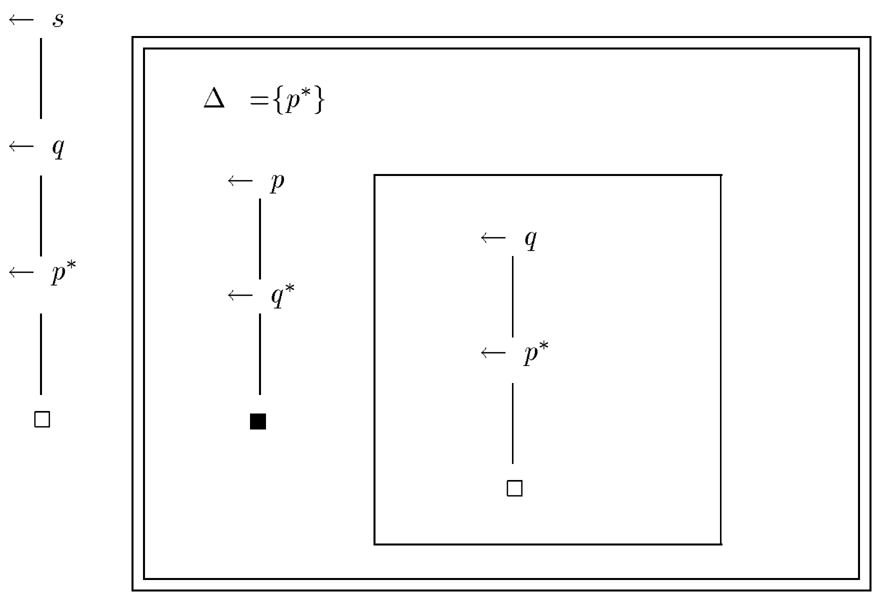

Figure 2: computation for example 4.3

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 query $\leftarrow s$ has no SLDNF refutation. Moreover, the SLDNF proof procedure, executing the query, goes into an infinite loop. 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 [33]) 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 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 [129].

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

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

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 [33] show by means of this example, the abductive proof procedure is not always sound with respect to the above abductive semantics of NAF. It is possible, however, to argue that it is the semantics and not the proof procedure that is at fault. Indeed, Saccà and Zaniolo [120], Przymusinski [110] 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 [22]. The abductive proof procedure is sound with respect to this improved semantics. Satoh and Iwayama [122], on the other hand, show how to extend the abductive proof procedure of [33] to deal correctly with the stable model semantics. Their extension modifies the integrity checking method of [76] and deals with arbitrary integrity constraints expressed in the form of denials.

Casamayor and Decker [7] also develop an abductive proof procedure for NAF. Their proposal combines features of the Eshghi-Kowalski procedure with ancestor resolution.

Dung [25] shows that in certain cases disjunctions such as

$$p \vee q$$

can be represented by clauses of the form

$$p \leftarrow \sim q$$

$$q \leftarrow \sim p$$

and for these cases the Eshghi-Kowalski procedure is adequate. For the more general case in which this representation is not adequate and disjunction needs to be represented explicitly, Dung [26] extends the Eshghi-Kowalski procedure by using resolution-based techniques similar to those employed in [36].

Finally, we note that, in order to capture the semantics more closely for programs such as $p \leftarrow p$ where $\sim p$ holds, we can define a non-effective extension of the proof procedure, that allows infinite failure in the consistency phases.

4.3 An argumentation-theoretic interpretation

Dung [22] replaces the disjunctive integrity constraints by a weaker requirement that the set of negative hypotheses Δ be maximal. 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 above, only the second extension is intuitively correct.

To avoid such problems Dung defines a more subtle notion of maximality. 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 P^*) if $P^* \cup E \vdash p$ for some $p^* \in \Delta$. 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.

Thus a preferred extension can be thought of as a maximal consistent set of hypotheses that contains its own defence against all attacks. In [22] a consistent set of hypotheses Δ (not necessarily maximal) satisfying the property of containing its own defence against all attacks is said to be **acceptable** (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 logic programming. Kakas and Mancarella [68] show that it is equivalent to Saccà and Zaniolo's partial stable model semantics [120]. Like the partial stable model semantics, it includes the stable model semantics as a special case. Dung also shows that the well-founded model [129] is the least complete extension that can be constructed bottom-up from the empty set of negative hypotheses, by adding incrementally all acceptable hypotheses. Thus the well-founded semantics is minimalist and sceptical, whereas the preferred extension semantics is maximalist and credulous. A fixpoint construction of the preferred extension semantics is given in [28].

Kakas and Mancarella [69, 70] propose a modification of the preferred extension semantics. Their proposal can be illustrated by the following example.

Example 4.6
In the abductive framework corresponding to the program

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

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 acceptable 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 acceptable because the only attack E against Δ attacks itself, and therefore should not be regarded as an acceptable attack against Δ.

To deal with this kind of example, Kakas and Mancarella 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 **stable** if

- Δ is maximal with respect to the property that for every attack E against Δ, $E \cup \Delta$ attacks $E - \Delta$.

Note that here the condition "$P^* \cup \Delta$ is consistent with I^*" is subsumed by the new maximality condition. Like the original definition of preferred extension, the definition of stable set of hypotheses was not originally formulated in terms of attack, but is equivalent to the one presented above.

Kakas and Mancarella [70] 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

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

Here 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 acceptable) according to the definition above.

To generalise the reasoning in example 4.7, 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 any hypothesis in the set is acceptable. More precisely, given an initial set of hypotheses Δ_0, a set of hypotheses Δ is **acceptable** to Δ_0 iff

for every attack E against $\Delta - \Delta_0$, E 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 $\emptyset$.
Notice that, as a special case, we obtain a basis for the definition:

Δ is acceptable to Δ_0 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

Δ is acceptable to Δ_0 if for every attack E against $\Delta - \Delta_0$,
$E \cup \Delta \cup \Delta_0$ attacks $E - \Delta$.

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

An "argumentation-theoretic" interpretation for the semantics of NAF in logic programming has also been developed by Geffner [44]. However, his interpretation is only an approximation to the definition of acceptability given above and is equivalent to the well-founded semantics [27]. A similar approximation to the notion of acceptability has also been proposed by

Simari and Loui [126], who define an argumentation-theoretic framework for default reasoning in general. They combine a notion of acceptability with Poole's notion of "most specific" explanation [102], to deal with hierarchies of defaults.

4.4 The abductive proof procedure revisited

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 is because the different phases of the proof procedure can be interpreted in terms of the notions of attack and defence. To illustrate this interpretation, consider again figure 1 of example 4.2. The consistency phase for p^*, shown in the outer 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 acceptable, 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^*. Similarly, figure 2 of example 4.3 shows how the attack q^* against p^* is counterattacked by p^* itself.

In general, the proof procedure constructs an acceptable set of negative hypotheses, a subset of which is sufficient to solve the original goal. The remaining hypotheses are necessary to defend this sufficient subset against any attack. With the help of this new interpretation it is possible 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 at the end of section 4.3. The extension of the proof procedure involves temporarily remembering a (selected) attack E and using E itself together with the subset 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.

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

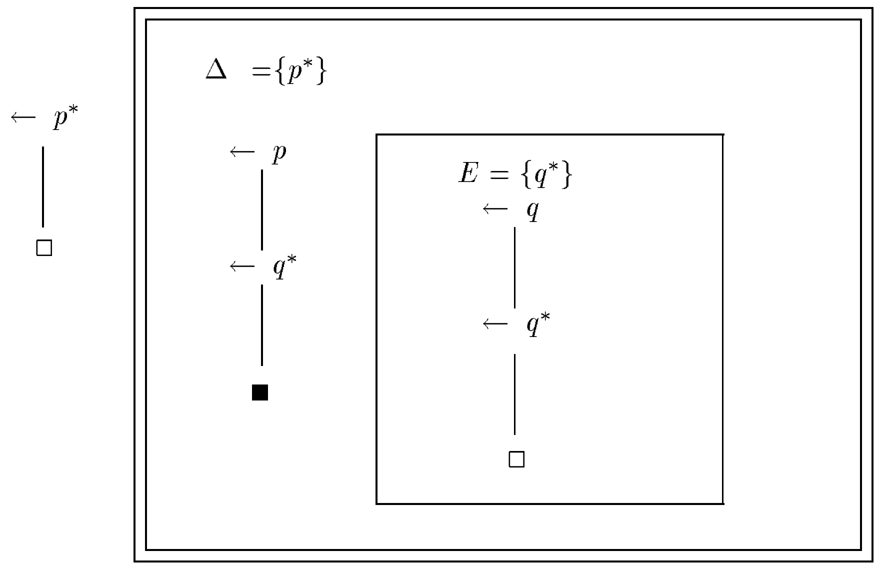

Figure 3: computation for example 4.6 with respect to the revisited proof procedure

interaction that can arise between a program (proof procedure in this case) and its specification (semantics).

5 Abductive Logic Programming

Abductive Logic Programming (ALP), as understood in the remainder of this paper, is the extension of logic programming 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 lost 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 [9]. This assumption

[9]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

has the important advantage that all explanations are thereby guaranteed to be basic.

Semantics and proof procedures for ALP have been proposed by Eshghi and Kowalski [32], Kakas and Mancarella [63] and Chen and Warren [10]. Chen and Warren extend the perfect model semantics of Przymusinski [109] 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 [63] develop a semantics for ALP by generalising the stable model semantics for logic programming. 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 $M(\Delta)$.

The 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

$$p \leftarrow a$$

used to separate extensional and intensional predicates in deductive databases [88]. 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 not abducible anymore, while $\delta_a(x)$ becomes abducible.

$$q \leftarrow b$$

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 integrity constraints introduced by the transformation of section 4 [10]. 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

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

[10] 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).

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^*.

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. The first kind of abduction requires hypotheses to be maximised, while the second kind usually requires them to be minimised. The definition of generalised stable models appropriately maximises NAF hypotheses, but neither maximises nor minimises other hypotheses. In practice, however, the abductive proof procedure generates only hypotheses that are relevant for a proof. Because of this, it tends to minimise the generation of both kinds of hypotheses. On the other hand, the proof procedure also generates as many hypotheses as it needs for a proof. In this sense, it tends to maximise the generation of hypotheses. This property of the proof procedure and its relationship with the semantics needs to be investigated further.

5.2 Abductive proof procedure for ALP

In [64, 65, 66], proof procedures are given to compute abductive explanations in ALP. These extend the abductive proof procedure for NAF [33] described in section 4.2, retaining the basic structure which interleaves an abductive phase that generates and collects abductive hypotheses with a

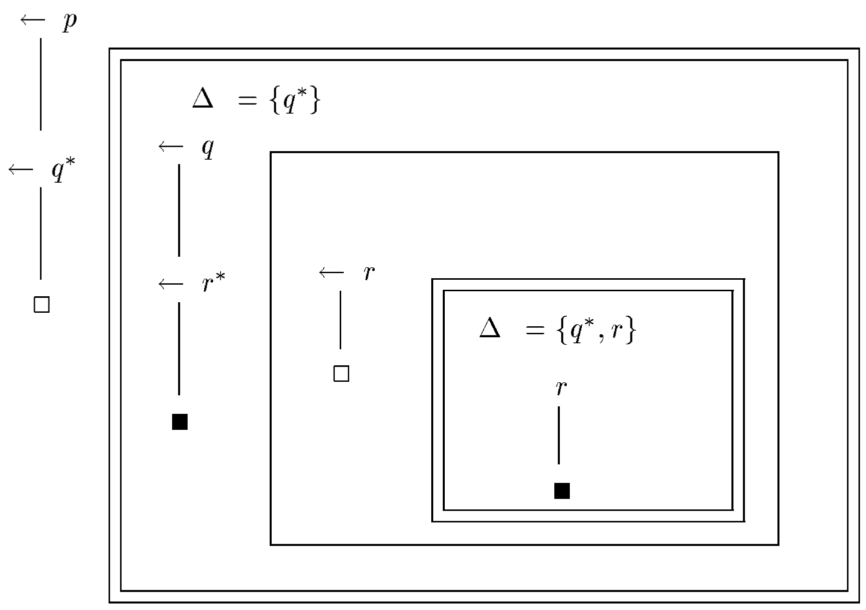

Figure 4: extended proof procedure for example 4.2

consistency phase that incrementally checks these hypotheses for integrity. We will illustrate these extended proof procedures by means of examples.

Example 5.4
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.

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 [77]). 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 procedures, however, 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 the example 5.4 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 5.5
Consider the program

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

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

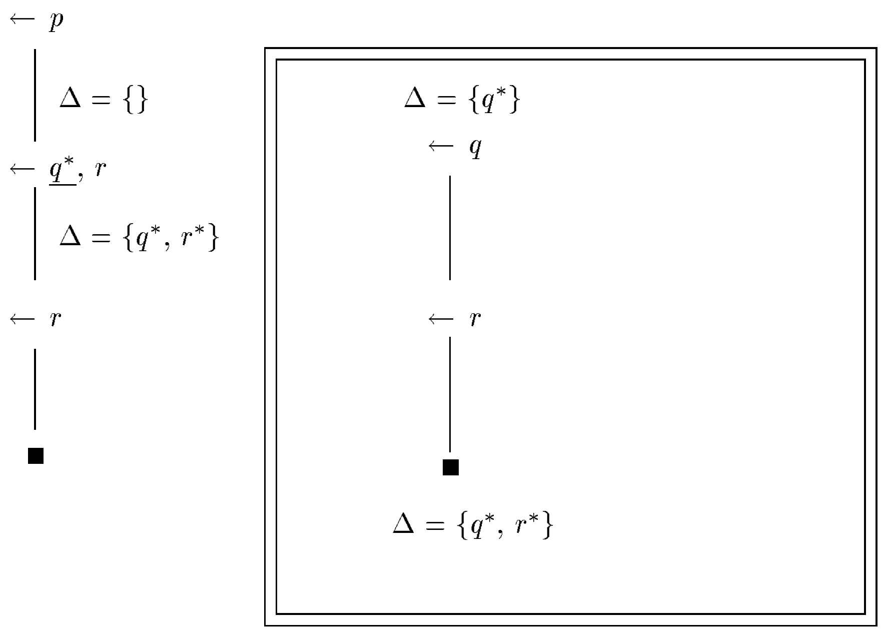

Figure 5: extended proof procedure for example 5.5

stable model semantics, as we have assumed in example 5.5 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 5.3), 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 [11] containing at least one literal whose predicate is abducible. In this case the abductive proof procedure needs only a minor extension [65, 66]: 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 5.6
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 single box. The integrity check for b is trivial, as b does not appear in any integrity constraint. 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. Nevertheless the proof procedure is sound because the violation is found later by backward reasoning when a is resolved with the second integrity constraint. This shows that $\Delta = \{a\}$ is unacceptable because it is incompatible with b which is needed to defend Δ against the attack q^*.

In summary, the overall effect of additional integrity constraints is to increase the size of the search space during the consistency phase, with no

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

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

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

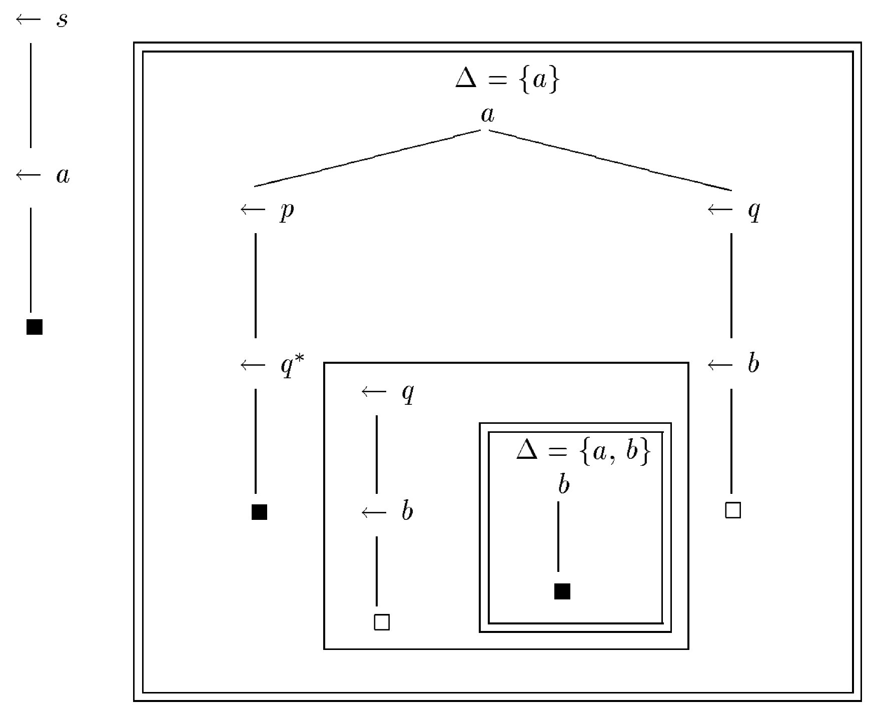

Figure 6: extended computation for example 5.6

significant change to the basic structure of the backward reasoning procedure.

The abductive proof procedures described above suffer 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).

Finally, we note that the abductive proof procedures described here perform many of the functions of a truth maintenance system. The relationships between ALP and truth maintenance will be discussed in section 6.

5.3 Stable model semantics extended with explicit negation

In general logic programs, negative information is inferred by means of NAF. This is appropriate when the closed world assumption [113], 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. (As we shall see later, in sections 5.7 and 5.8, this is related to the use of integrity constraints expressed in the form of denials.) In this section we will discuss the extension proposed by Gelfond and Lifschitz [46]. 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 [46]. However, as we will see below, because the contrapositives of extended clauses do not hold, the term "classical negation" is inappropriate. For this reason we use the term "explicit negation" instead.

A similar notion has been investigated by Pearce and Wagner [93], who develop an extension of Horn 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 an implicit contradiction 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 [46] 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 an implicit 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 [110] for example applies the well-founded semantics to extended logic programs. Similarly any other semantics can also be applied. 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. We will discuss this problem in section 7.

5.4 Simulation of abduction through NAF

Satoh and Iwayama [121] show that an abductive logic program can be transformed into a logic program without abducibles but where the integrity constraints remain. Although they do not employ explicit negation, the transformation implicitly simulates explicit negation by the introduction of new predicates. For each abducible predicate p in A, a new predicate p' is introduced representing the complement of p and a new pair of clauses [12]:

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

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, as we have already remarked in the case of the semantics of explicit negation, 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 (e.g. as in [96], see below).

Example 5.7
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.

Similar methods for transforming abductive assumptions into NAF assumptions are employed by Inoue [56] and Pereira, Aparicio and Alferes [96]. They transform extended logic programs augmented with abduction into extended logic programs without abduction by adding to the program a new pair of clauses

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

[12]Satoh and Iwayama use the notation $p^*(x)$ instead of $p'(x)$ and consider only propositional programs.

for each abducible predicate p. Notice that the transformation is identical to that of Satoh and Iwayama, except for the use of explicit negation instead of new predicates. Inoue and Pereira, Aparicio and Alferes assign different semantics to the resulting program. Whereas Inoue applies the answer set semantics, Pereira, Aparicio and Alferes apply the well-founded semantics and the extended stable model semantics of [110]. The well-founded semantics can be thought of as representing a minimal incomplete view of the world and the extended stable model semantics as representing different ways of extending this view by abducing negative hypotheses. Pereira, Aparicio and Alferes [98] have also developed proof procedures for this semantics. These procedures can be used as abductive proof procedures for ALP.

As mentioned above, Pereira, Aparicio and Alferes [96] understand the transformed programs in terms of (three-valued) extended stable models. The extended stable model semantics 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 $p'(t)$).

5.5 Computation of abduction through TMS

Satoh and Iwayama [121] 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 [20]. 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 5.8
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 [121] deterministically computes only the intended model M_2, without also computing and rejecting M_1.

5.6 Restoring consistency of answer sets

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

Example 5.9
The extended logic program:

$$\begin{array}{l} \neg\, fly(x) \leftarrow\sim bird(x) \\ fly(x) \leftarrow bat(x) \\ bat(Tom) \end{array}$$

has no consistent answer set.

As mentioned in section 5.3, 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 [23] and [97]. Both of these studies are primarily concerned with the related problem of inconsistency of the well-founded semantics when applied to extended logic programs [110].

To deal with the problem of inconsistency in extended logic programs, Dung [23] applies 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 in the same way as preferred extensions in section 4 but 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.

Pereira, Aparicio and Alferes [97] employ a similar approach in the context of Przymuszynski's extended stable models [110]. 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 an application of belief revision, where if inconsistency can be attributed to an abducible hypothesis or a retractable atom (see below section 5.7), then we can reject the hypothesis to restore consistency. In fact Pereira, Aparicio and Alferes have also used this method to study counterfactual reasoning [99].

Both methods, [23] and [97], can deal only with inconsistencies that can be attributed to NAF hypotheses, as shown by the following example.

Example 5.10
It is not possible to restore consistency by removing NAF hypotheses given the program:

$$p$$

$$\neg p.$$

However, Inoue [55, 56] suggests a general method for restoring consistency, which is applicable to this case. This method (see also section 5.8) is based on [43] and [104] 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 [104] (see section 3) in terms 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 reason with it by regarding it as a knowledge system of the form

$$(\emptyset, P).$$

For example 5.10 this will give two alternative semantics, $\{p\}$ or $\{\neg p\}$.

5.7 Abduction as retractability

An alternative way of viewing abduction, which emphasises the defeasibility of abducibles, is **retractability** [47]. 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 5.6).

One approach to this understanding of abduction is presented in [77]. 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 [13]. 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

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

[13]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.

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

$$a(x).$$

To avoid inconsistency, the program P with integrity constraints I can be transformed to 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 every clause of the form

$$p \leftarrow r$$

by the clause

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

where the condition $\sim q$ needs to be further transformed, 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.

The Kowalski-Sadri transformation is (almost) the inverse of the Eshghi-Kowalski transformation, which interprets NAF as abduction. To see this, consider again the propositional form of the Yale shooting problem.

Example 5.11
Given the program

$$p \leftarrow \sim q$$

$$q \leftarrow \sim r$$

applying the Eshghi-Kowalski transformation gives

$$p \leftarrow q^*$$

$$q \leftarrow r^*$$

$$\neg [p \wedge p^*]$$

$$\neg [q \wedge q^*]$$

$$\neg [r \wedge r^*]$$

together with the disjunctive integrity constraints. To apply the Kowalski-Sadri transformation these disjunctive integrity constraints are replaced by the stronger (but retactable) assertions

$$p^*$$

$$q^*$$

$$r^*.$$

Applying the Kowalski-Sadri transformation now yields

$$\begin{array}{lll} p & \leftarrow & q^* \\ q & \leftarrow & r^* \\ p^* & \leftarrow & \sim p \\ q^* & \leftarrow & \sim q \\ r^* & \leftarrow & \sim r. \end{array}$$

If we are only interested in the clauses defining the predicates, p, q and r, in the original program, this can be simplified to

$$p \leftarrow \sim q$$

$$q \leftarrow \sim r$$

which is the original program.

It is interesting to note that the (informal) **retraction semantics** of the intermediate program with integrity constraints and retractable assumptions yields the single (correct) semantics for this example, namely the one in which the assumption q^* is retracted. It would be useful to study the retraction semantics in more general and more formal terms and to compare it with the preferred extension, stable theory and acceptability semantics.

The retraction semantics and the associated transformation can be applied more generally to cases of default reasoning where the retractable atoms do not correspond to abducible predicates.

Example 5.12
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

$$\neg\,[fly(x) \wedge ostrich(x)],$$

with $fly(x)$ retractable. The integrity constraint is violated, because both $ostrich(John)$ and $fly(John)$ hold. Integrity can be restored by retracting the instance

$$fly(John) \;\leftarrow\; bird(John)$$

of the first clause in the program.

Similarly the transformed program avoids inconsistency in general by replacing the first clause and the integrity constraint by the more restrictive clause

$$fly(x) \leftarrow bird(x), \sim ostrich(x).$$

5.8 Rules and exceptions in logic programming

One problem with the retraction semantics is that the equivalence of the original program with the transformed program was proved only in the case where the transformed program is locally stratified. Moreover the proof of equivalence is based on a tedious comparison of search spaces for the two programs. This problem was solved in a subsequent paper [78] by re-expressing integrity constraints as extended clauses where the retractable atoms are expressed as explicitly negated conclusions. By appropriately modifying the answer set semantics to retract clauses whose positive conclusions contradict clauses with negative conclusions, the equivalence of the original program and the transformed program can be proved more simply and without any restrictions. Moreover, the new formulation with explicitly negated conclusions is more informative than the earlier formulation with integrity constraints, which only constrained positive information and did not add negative information explicitly.

In the new formulation it is natural to interpret clauses with negative conclusions as exceptions, and clauses with positive conclusions as default rules. In the flying-bird example of the previous section, in particular, the integrity constraint

$$\neg\, [fly(x) \wedge ostrich(x)]$$

with $fly(x)$ retractable would now be formulated as an exception

$$\neg\, fly(x) \leftarrow ostrich(x)$$

to the "general" rule

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

To capture the intention that exceptions should override general rules, the answer set semantics is modified, so that instances of clauses with positive conclusions are retracted if they are contradicted by explicit negative information.

Kowalski and Sadri [78] also present a new transformation, which preserves the new semantics, and is a more elegant form of the original transformation. In the case of the flying-birds example 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

$$fly(x) \leftarrow bird(x), \sim\, ostrich(x).$$

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

$$\neg\, fly(x) \leftarrow ostrich(x)$$

with a positive exception

$$fly(x) \leftarrow super\text{-}ostrich(x)$$

can be transformed into a clause

$$\neg\, fly(x) \leftarrow ostrich(x),\ \sim\ fly(x)$$

and all occurrences of the negative literal $\neg\, fly(x)$ can be renamed by a new positive literal $fly'(x)$.

A more direct approach to the problem of treating positive and negative predicates symmetrically in default reasoning is presented by Inoue [55, 56] following the methods of [43] and [104] (see section 5.6 for a discussion). This work is another interesting application of the notion of maximal consistency to extend logic programming for default reasoning.

As a possible direction for future work, it would be desiderable to reconcile the different approaches of Inoue and of Kowalski and Sadri. Such a reconciliation might attempt to treat NAF hypotheses and other kinds of defaults uniformly as cases of abductive reasoning, generalising appropriately the preferred extension, stable theory and acceptability semantics of NAF.

5.9 A methodology for default reasoning with explicit negation

Compared with other authors, who primarily focus on extending or modifying the semantics of logic programming to deal with default reasoning, Pereira, Aparicio and Alferes [96] 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

$$p \leftarrow q, \sim \neg\, nameqp, \sim \neg\, p \tag{3}$$

where the condition $nameqp$ is 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

$$fly(x) \leftarrow bird(x), \sim \neg\, birds\text{-}fly, \sim \neg\, fly(x) \tag{4}$$

$$\neg\, birds\text{-}fly(x) \leftarrow penguin(x) \tag{5}$$

expresses that the default named $birds\text{-}fly$ 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 (5) to the default rule "penguins usually don't fly". This can be done by replacing (5) by

$$\neg\, fly(x) \leftarrow penguin(x), \sim \neg\, penguins\text{-}don't\text{-}fly(x), \sim fly(x) \tag{6}$$

where $penguins\text{-}don't\text{-}fly$ is the name assigned to the new rule. To give preference to the more specific default represented by (6) over the more general default (4), we add the additional clause

$$\neg\, birds\text{-}fly(x) \leftarrow penguin(x), \sim \neg\, penguins\text{-}don't\text{-}fly(x).$$

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

$$\neg penguins\text{-}don't\text{-}fly(x) \leftarrow superpenguin(x).$$

Pereira, Aparicio and Alferes [96] 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 [55, 56] uses an extension of the answer set semantics (see section 5.6), but for a slightly different transformation.

Note that these methodologies can be seen as a refinement of the direct use of the transformation presented in section 5.7.

5.10 Abduction through deduction from the completion

In the proposals 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 [13]. Here clauses of programs are interpreted as if-halves of if-and-only-if definitions that are obtained from the completion of the program [11] restricted to non-abducible predicates. Forward reasoning with the only-if-halves of these definitions, starting from the observation to be explained, generates abductive hypotheses deductively.

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 $\not\models F' \rightarrow F$.

The authors also define a proof procedure that generates explanation formulas for observations. This proof procedure reasons forward from a given observation O by means of the only-if-halves of the completion P_C. Termination and soundness of the proof procedure are ensured for a restricted class of programs (i.e. hierarchical). The explanation formula resulting from the computation characterises all the different abductive explanations for O, as exemplified in the following example.

Example 5.13
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:

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

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

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

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

corresponding to the abductive explanations $\Delta_1 = \{broken\text{-}spokes\}$, $\Delta_2 = \{punctured\text{-}tube\}$ and $\Delta_3 = \{leaky\text{-}valve\}$.

Console, Duprè and Torasso extend this approach to deal with non-propositional abductive logic programs. In this more general case an equality theory, identical to the one presented in [11], is needed; and in the definition of explanation formula, the notion of F being more specific than F' requires that $F \rightarrow F'$ be a logical consequence of such an equality theory and that $F' \rightarrow F$ not be a consequence of the equality theory. As a consequence, the explanation formula is unique up to equivalence in the equality theory, and the proof procedure is more complex than for the propositional case, because it needs to generate consequences of the equality theory.

Denecker and De Schreye [19] compare the search spaces for reasoning backward using the if-halves of definitions with those for reasoning forward using the only-if-halves for logic programs without NAF. They show a structural equivalence between the search spaces for SLD-resolution extended with abduction and the search spaces for model generation with SATCHMO [86] augmented with term rewriting to simulate unification.

A discussion of the general phenomenon that reasoning with the if-halves of definitions can often simulate reasoning with the only-if-halves, and vice versa can be found in [75].

A different deductive framework for abduction is presented in [58] [14]. This method is related to a similar method for NAF presented in [57]. Inoue et al. translate each abductive logic program rule of the form

$$p \leftarrow q, a$$

where a is abducible, into a rule of a disjunctive logic program

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

[14]A description of this work can also be found in [51].

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

$$\neg (a \wedge a'). \tag{7}$$

A model generator (like SATCHMO or MGTP [37]) can then be applied to compute all the minimal models that satisfy the integrity constraints (7).

6 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 [20] and the assumption-based TM system (ATMS) of deKleer [71].

6.1 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 [90] 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 5.4, 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 [29, 48, 65, 100], in particular, show that a well-founded labelling for a JTMS corresponds to a stable model of the corresponding logic program. Several authors [29, 38, 65, 112], exploiting the interpretation of stable models as autoepistemic expansions [45], 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 [24, 48, 65]. 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 [121], who give a proof procedure to compute generalised stable models using the JTMS (see section 5.5).

6.2 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 6.1
Suppose that the ATMS contains justifications

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

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:

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

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 [118] 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 [54], 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 [81].

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 [65]. Other non-monotonic extensions of ATMS have been developed in [59, 119].

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.

7 Conclusions and Future Work

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

Perhaps the most important link, from the perspective of logic programming, is that between abduction and NAF. On the one hand, we have seen that 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 can be transformed into logic programs with NAF together with integrity constraints or explicit negation. The link between abduction and NAF includes both their semantics and their implementations.

We have argued that semantics can best be understood as providing a specification for an implementation. From this point of view, a semantics is a "declarative" specification, which might be non-constructive, but need not be concerned with meaning-theoretic notions such as "truth" and "falsity". Thus an overtly syntactic, but non-constructive, specification given in terms of maximally consistent extensions is just as much a "semantics" as one involving (covertly syntactic) stable models.

We have seen the importance of clarifying the semantics of abduction and of defining a semantics that helps to unify abduction, NAF, and default reasoning within a common framework. We have seen, in particular, that an implementation which is incorrect under one semantics (e.g. [33]) can be correct under another (e.g. [22]). We have also introduced an argumentation-theoretic interpretation for the semantics of abduction applied to NAF, and we have seen that this intepretation can help to understand the relationships between different semantics.

Despite the recent advances in the semantics of NAF there is still room for improvement. One possibility is to explore further the direction set in [28] and [70] which characterises the acceptability of a set of hypotheses Δ recursively in terms of the non-acceptability of all attacks against Δ. Another is to identify an appropriate concept of maximal consistency, perhaps along the lines of the retractability semantics suggested in [77]. The two possibilities need not be mutually exclusive. The former, recursive specification would be closer to an implementation than the latter. But the two specifications might otherwise be equivalent.

The use of abduction for NAF is a special case. It is necessary therefore to define a semantics that deals appropriately both with this case and with the other cases. In particular, we need to deal both with abductive hypotheses which need to be maximised for default reasoning and with other abductive hypotheses which need to be minimised. It is interesting that the abductive proof procedure can be regarded as both maximising and minimising the two kinds of abducibles. It maximises them in the sense that it (locally) makes as many abductive assumptions as are necessary to construct a proof. It minimises them in the sense that it makes no more assumptions than necessary. Perhaps this is another case where the implementation of abduction is more correct than the (semantic) specification.

It is an important feature of the abductive interpretation of NAF that it possesses an elegant and powerful proof procedure, which significantly extends SLDNF and which can be extended in turn to accommodate other abducibles and other integrity constraints. Future work on the semantics of ALP needs to preserve and develop further this existing close relationship between semantics and proof procedure.

The abductive proof procedure needs to be extended and improved in various ways. One such extension is the generation of non-variable-free hypotheses, containing variables. This problem, which has been studied in part in [10], [30] and [103], and also by [18] with respect to the completion semantics, involves the treatment of the equality predicate as a further abducible. Because NAF is a special case of abduction, the problem of constructive negation in logic programming [4, 8, 130] is a special case of abduction with non-variable-free hypotheses.

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 [21, 42, 91, 92] and various integrity constraint checking and restoration procedures.

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

The relationship between integrity constraints and explicit negation 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 concerning deductive databases; and what are the implications of this relationship on the semantics and implementation of integrity constraints?

Whatever the answers to these questions, it is clear that the combination of explicit negation and implicit NAF is very useful for knowledge representation in general. It is important, however, to obtain a deeper understanding of the relationships between these two forms of negation. It is clear, for example, that if $\sim p$ holds then $\neg p$ must be consistent. However, it is not the case that if $\neg p$ is consistent, then $\sim p$ holds, as in the following example

$$p \leftarrow \sim p$$
$$\neg p.$$

Thus, there is no simple relationship whereby one form of negation clearly subsumes the other.

Another problem, which we have already mentioned, is how to decide whether a negative condition should be understood as explicit negation or as NAF. One possibility might be simply to interpret the negation as NAF if the closed world assumption applies, and as explicit negation if the open world assumption applies. Moreover the presence of any rules in which the predicate of the condition occurs explicitly negated in a conclusion would suggest that the open world assumption applies and the negated condition therefore is explicit. Another, complementary possibility is to recognise that the open world assumption must apply to any predicate explicitly declared as abducible. Consequently, any negated condition whose predicate is abducible must be interpreted as explicit negation.

We have seen that explicit negation does not obey the laws of contraposition. This is further strong evidence that the semantics of clauses should be interpreted in terms of inference rules and not in terms of implications. Because of the similarity between default rules in Default Logic and clauses interpreted as inference rules in logic programming, this provides further evidence also for the possibility of developing a uniform semantics and implementation in which NAF, abduction, and default reasoning can be combined.

We have remarked upon the close links between the semantics of logic programming 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 TMSs and ATMSs correspond only to the propositional case of logic programs?

We have observed a duality between forward reasoning with only-if-halves of definitions and logic programming-style backward reasoning with if-halves. Could this duality apply also to a possible correspondence between inconsistency in truth maintenance systems and failures in logic programming?

We believe that our survey supports the belief that abduction is an important and powerful extension of logic programming. It also points forward to the possibility that at some time in the future further extensions of logic programming might be fully adequate and appropriate for many, if not all, knowledge representation and reasoning tasks in AI.

Acknowledgements

This research was supported by Fujitsu Research Laboratories and by the Esprit Basic Research Action Compulog. The authors are grateful to Katsumi Inoue and Ken Satoh for helpful comments on an earlier draft, and to

Phan Minh Dung and Paolo Mancarella for many helpful discussions.

References

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

[2] Allemand, D., Tanner, M., Bylander, T., Josephson, J., On the computational complexity of hypothesis assembly. *Proc. 10th International Joint Conference on Artificial Intelligence*, Milan (1987) 1112–1117

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

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

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

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

[7] 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)

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

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

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

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

[12] Console, L., 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

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

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

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

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

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

[18] 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)

[19] 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

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

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

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

[23] 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

[24] 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)

[25] 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

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

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

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

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

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

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

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

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

[34] 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)

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

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

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

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

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

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

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

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

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

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

[45] 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

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

[47] 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*, London (1986) Springer Verlag Lecture Notes in Computer Science 225, 211–222

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

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

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

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

[52] 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)

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

[54] 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)

[55] Inoue, K., Hypothetical reasoning in logic programs. Technical Report ICOT 607, Tokyo (1991)

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

[57] 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)

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

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

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

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

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

[63] 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

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

[65] 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)

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

[67] 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)

[68] Kakas, A. C., Mancarella, P., Preferred extensions are partial stable models. to appear in *Journal of Logic Programming*

[69] 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)

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

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

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

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

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

[75] Kowalski, R.A., Logic programming in artificial intelligence. *Proc. 12th International Joint Conference on Artificial Intelligence*, Sidney (1991) 596–603

[76] Kowalski, R.A., Sadri, F., 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

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

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

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

[80] 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

[81] 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)

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

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

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

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

[86] 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

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

[88] 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

[89] 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

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

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

[92] 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

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

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

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

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

[97] 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)

[98] 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

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

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

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

[102] 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

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

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

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

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

[107] 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

[108] 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

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

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

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

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

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

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

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

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

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

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

[119] 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)

[120] 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

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

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

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

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

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

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

[127] 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

[128] 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

[129] 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)

[130] Wallace, M., Negation by constraints: a sound and efficient implementation of negation in deductive databases. *Proc. 4th Symposium on Logic Programming*, San Francisco (1987)