ELSEVIER

Theoretical Computer Science 165 (1996) 171–200

Theoretical Computer Science

Three-valued completion for abductive logic programs

Frank Teusink*

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

Abstract

In this paper, we propose a three-valued completion semantics for abductive logic programs, which solves some problems associated with the Console et al. two-valued completion semantics. The semantics is a generalization of Kunen's completion semantics for general logic programs, which is known to correspond very well to a class of effective proof procedures for general logic programs. Secondly, we propose a proof procedure for abductive logic programs, which is a generalization of a proof procedure for general logic programs based on constructive negation. This proof procedure is sound and complete with respect to the proposed semantics. By generalizing a number of results on general logic programs to the class of abductive logic programs, we present further evidence for the idea that limited forms of abduction can be added quite naturally to general logic programs.

1. Introduction

Abduction is a form of inference where one, given some rules and an observation, tries to find an explanation of that observation using these rules. For instance, given a rule

shoes_are_wet ← *it_is_raining*

the observation *shoes_are_wet* would be explained by *it_is_raining*. As such, abduction is quite the reverse of deduction, where facts and rules are used to derive conclusions. In the above example, from the fact *shoes_are_wet* one cannot derive anything. But from the fact *it_is_raining* one can derive *shoes_are_wet*. One can find abduction in many fields within the realm of artificial intelligence and knowledge engineering, including diagnosis, planning, computer vision, natural language understanding default reasoning, and knowledge assimilation.

Abductive logic programming (first proposed in [14]) is a crossover between logic programming and abduction. The idea is to represent the rules as a logic program and the observation as a query. Then, abduction is used to infer an explanation

* E-mail: frankt@cwi.nl.

SSDI 0304-3975(96)00044-8

using program and query. The best-known semantics for abductive logic programs are those based upon (generalized) stable models [12, 22, 18] and argumentation semantics [12, 19]. Proof procedures for these semantics were proposed by Eshghi and Kowalski [14] and extended by Satoh and Iwayama [22] and Kakas and Mancarella [18]. In [5], Console, Dupre and Torasso propose a different kind of semantics, based on the two-valued completion of a program. The aim of their paper was to investigate the relation between abduction and deduction. In [7], Denecker and DeSchreye propose a proof procedure for such a two-valued completion semantics, which is based on SLDNF-resolution. For a thorough overview on abductive logic programming and their semantics, we refer to the excellent survey by Kakas, Kowalski and Toni [17].

In logic programming, completion semantics was developed as a semantics for describing what can be computed using SLDNF-resolution. By giving a completion semantics for abductive logic programming, Console et al. showed that abduction is closely related to deduction. Denecker and DeSchreye added to this by proposing SLDNFA-resolution, a proof procedure for abductive logic programming based on SLDNF-resolution. However, a disadvantage of the two-valued completion approach is, that it is not defined for arbitrary programs: for many interesting programs there do not exist two-valued models of their completion. In general logic programming, it has been shown that three-valued semantics are better suited to characterize proof procedures based on SLD-resolution than two-valued semantics. In [16], Fitting proposes a three-valued immediate consequence operator, on which he bases a semantics (*Fitting semantics*). Basically, it states that a formula is true in a program iff it is true in all three-valued Herbrand models of the completion of that program. In [20], Kunen proposes an alternative to this semantics (*Kunen semantics*), in which a formula is true in a program iff it is true in all three-valued models of the completion of that program.

In this paper, we generalize Fitting semantics and Kunen semantics to abductive logic programs. In the process, we also propose a three-valued immediate consequence operator, and truth- and falseness formulas as presented by Shepherdson in [23], for abductive logic programs. With this, we provide abduction with a semantics which gives a good characterization of the answers that can be actually computed by effective proof procedures. This in contrast with semantics based on well-founded semantics, whose proof procedures involve expensive loop checking, and those based on stable model semantics, which become intractable as soon as function symbols are used. Such complex semantics are interesting from the point of view of knowledge representation, and are definitely of use in specific problem domains, but are not viable candidate semantics for general purpose (abductive) logic programming systems. This in contrast to Kunen semantics, which is commonly used in the verification of general logic programs. By providing a Kunen semantics for abductive logic programs, we present further evidence for the idea that limited forms of abduction can be added quite naturally to general logic programs.

The obtained results are also interesting within the context of *modular logic programming*, where one reasons with predicates which are (partially) undefined, or defined in other modules, and of *constraint logic programming*, where some of the predicates

represent constraints that are to be handled by a constraint solver. In each of these cases there is a distinction between the 'logic programming part' of the program and some other part, which is either abducible, handled by an other (unknown) logic program, or handled by a constraint solver. In these contexts, it is interesting to see if we can find a semantics of the logic programming part, which is parametric with respect to the 'abducible', 'open' or 'constraint' part.

Moreover, we present an alternative proof procedure, based upon SLDFA-resolution: a proof procedure proposed by Drabent [11]. This proof procedure solves some problems associated with SLDNFA-resolution. First of all, by using constructive negation instead of negation as failure, we remove the problem of *floundering*. Secondly, instead of skolemizing non-ground queries, which introduces some technical problems, we use equality in our language, which allows a natural treatment of non-ground queries.

The paper is organized in three more or less separate parts. In the first part, we give an introduction to abductive logic programming (Section 3), and present two- and three-valued completion semantics (Section 4). Then, in the second part, which starts with Section 5, we present the immediate consequence operator (Section 6), and use it to characterize Fitting semantics (Section 7) and Kunen semantics (Section 8) for abductive logic programs. In the third part, we generalize SLDFA-resolution to the case of abductive logic programs (Section 9), and present some soundness and completeness results on SLDFA-resolution in Section 10.

2. Preliminaries and notation

In this paper, we use k, l, m and n to denote natural numbers, f, g and h to denote functions (constants are treated as 0-ary functions), x, y and z to denote variables, s, t and u to denote terms, p, q and r to denote predicate symbols, A, B and C to denote atoms, L, M and N to denote literals, G, H and I to denote goals, θ, δ, σ, τ and ρ to denote abducible formulas (they will be defined later) and ϕ and ψ to denote formulas.

In general, we use underlining to denote finite sequences of objects. Thus, $\underline{L}$ denotes a sequence $L_1,\ldots,L_n$ of literals and $\underline{s}$ denotes a sequence $s_1,\ldots,s_n$ of terms. Moreover, in formulas we identify the comma with conjunction. Thus, $\underline{L}$ (also) denotes a conjunction $L_1 \wedge \cdots \wedge L_n$. Finally, for two sequences $s_1,\ldots,s_k$ and $t_1,\ldots,t_k$ of terms, we use $(\underline{s} = \underline{t})$ to denote the formula $(s_1 = t_1) \wedge \cdots \wedge (s_k = t_k)$.

In the remainder of this section, we introduce some basic notions concerning algebras and models. For a more thorough treatment of these notions, we refer to [9]. To begin with, an *algebra* (or *pre-interpretation*, as it is called in [21]), is the part of a model that interprets the terms of the language.

Definition 1. Let $\mathscr{L}$ be a language and let $\mathscr{F}$ be the set of function symbols in $\mathscr{L}$. An *$\mathscr{L}$-algebra* is a complex $J = \langle D, \mathbf{f}, \ldots\rangle_{f\in\mathscr{F}}$ where D is a non-empty set, *the domain* (or *universe*) of J, and for every n-ary function symbol $f \in \mathscr{F}$, $\mathbf{f}$ is an n-ary function $\mathbf{f} : D^n \to D$.

Note, that constant symbols are treated as 0-ary functions. Interpretation of terms of $\mathscr{L}$ in a $\mathscr{L}$-algebra J is defined as usual.

We now define the notion of two- and three-valued *models*.

Definition 2. Let $\mathscr{L}$ be a language. Let $\mathscr{F}$ be the set of function symbols in $\mathscr{L}$ and let $\mathscr{R}$ be the set of predicate symbols in $\mathscr{L}$. A *two-valued $\mathscr{L}$-model* is a complex $M = \langle D, \mathbf{f}, \ldots \mathbf{r}, \ldots \rangle_{f \in \mathscr{F}, r \in \mathscr{R}}$ where $\langle D, \mathbf{f}, \ldots \rangle_{f \in \mathscr{F}}$ is an $\mathscr{L}$-algebra, for every n-ary predicate symbol $r \in \mathscr{R}$, $\mathbf{r}$ is a subset of D^n, and equality (if present) is interpreted as identity.

Definition 3. Let $\mathscr{L}$ be a language. Let $\mathscr{F}$ be the set of function symbols in $\mathscr{L}$ and let $\mathscr{R}$ be the set of predicate symbols in $\mathscr{L}$. A *three-valued $\mathscr{L}$-model* is a complex $M = \langle D, \mathbf{f}, \ldots \mathbf{r}, \ldots \rangle_{f \in \mathscr{F}, r \in \mathscr{R}}$ where $\langle D, \mathbf{f}, \ldots \rangle_{f \in \mathscr{F}}$ is an $\mathscr{L}$-algebra, for every n-ary predicate symbol $r \in \mathscr{R}$, $\mathbf{r}$ is an n-ary function $\mathbf{r} : D^n \to \{\mathbf{t}, \mathbf{f}, \perp\}$, and equality (if present) is interpreted as two-valued identity.

Following [9], we treat equality as a special predicate with a fixed (two-valued) interpretation.

For two-valued models, the interpretation of (complex) formulas is defined as usual. For three-valued models, the interpretation of (complex) formulas is defined by the use of Kleene's truth-tables for three-valued logic. We use $\models$ to denote ordinary two-valued logical consequences, while $\models_3$ is used for three-valued logical consequences ($T \models_3 \phi$ iff ϕ is true in all three-valued models of T).

In this paper, we always use equality in the context of Clark's Equality Theory (*CET*), which consists of the following *Free Equality Axioms*:

(i) $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \to (x_1 = y_1) \wedge \cdots \wedge (x_n = y_n)(\forall f)$,

(ii) $f(x_1, \ldots, x_n) \neq g(y_1, \ldots, y_m)$ ($\forall$ distinct f and g),

(iii) $x \neq t$ (for all x and t where x is a proper sub-term of t).

Note, that the fixed interpretation of equality replaces the usual equality axioms, which are normally part of *CET*.

One important algebra is the *Herbrand Algebra HA*. It is the algebra that has the set of all closed terms as domain, and maps each closed term on 'itself'. Given an algebra J, a *J-model* is a model with algebra J. For instance, the set of all *HA*-models is the set of all Herbrand models. A *CET-algebra* is an algebra that satisfies *CET*. Note that every *CET*-algebra extends *HA*.

For a formula ϕ, *FreeVar* (ϕ) denotes the set of free variables in ϕ. A *sentence* is a closed formula (i.e. *FreeVar* (ϕ) is empty). A *ground* formula is a quantifier-free sentence. A *ground instance* of a formula ϕ is a formula ϕ' such that ϕ' is the result of substituting all variables in ϕ (free *and* local ones) by ground terms. When working with some language $\mathscr{L}$ and models over some domain D, it will sometimes be useful to work with the domain elements of D as if they were constants. This can be done using the following definitions. Given a language $\mathscr{L}$ and a domain D, the *D-language* $\mathscr{L}_D$ is obtained by extending $\mathscr{L}$ with a fresh constant for every domain element in D. When working in some language $\mathscr{L}$ and referring to D-sentences or D-formulas, we

intend sentences or formulas in the language $\mathscr{L}_D$. We can extend an $\mathscr{L}$-algebra J to an $\mathscr{L}_D$-algebra J_D by interpreting each new constant in $\mathscr{L}_D$ 'as itself', and extend a J-model M to a J_D-model M_D by replacing the algebra J by the algebra J_D. Given a domain D, a language $\mathscr{L}$ and a formula ϕ, a *D-ground* instance of ϕ is a ground instance of ϕ in the language $\mathscr{L}_D$. Given an algebra J with domain D, we sometimes refer to D-ground formulas as J-ground formulas.

Lemma 4. *Let J be an algebra with domain D and let M be a J-model. Let ϕ be a quantifier-free formula. Then, $M \models \phi$ iff for all J-ground instances ϕ' of ϕ $M_D \models \phi'$.*

In the following, given a model M with domain D and a D-ground formula ϕ, we write $M \models \phi$ whenever we intend $M_D \models \phi$.

In the remainder of this paper, we will not always specify the language. When no language is given, we assume a fixed 'universal' language $\mathscr{L}_{\mathscr{U}}$, which has a (countably) infinite number of constant and function symbols of all arities. The advantage of using such a universal language is, among others, that for that language *CET* is complete.

3. Abductive logic programming

Abduction is the process of generating an explanation E, given a theory T and an observation Ψ. More formally, E is an explanation for an abductive problem $\langle T, \Psi \rangle$, if $T \cup E$ is consistent, Ψ is a consequence of $T \cup E$, and E satisfies 'some properties that make it interesting'.

In this paper, we limit ourselves to the context of abductive logic programs, in which T is an *abductive logic program*, Ψ is a formula and E is an *abducible formula*.

An *abductive logic program* P is a triple $\langle \mathscr{A}_P, \mathscr{R}_P, \mathscr{I}_P \rangle$, where

- $\mathscr{A}_P$ is a set of *abducible predicates*,
- $\mathscr{R}_P$ is finite set of clauses $A \leftarrow \theta, \underline{L}$, where A is a non-abducible atom, θ is an abducible formula and $\underline{L}$ is a sequence of non-abducible literals, and
- $\mathscr{I}_P$ is a finite set of first-order integrity constraints.

An *abducible formula* (w.r.t. to a program P) is a first-order formula build out of the equality predicate '=' and the abducible predicates. An abducible formula δ is said to be *(in)consistent*, if $CET \cup \{\delta\}$ is (in)consistent.

Example 5. Here is an example abductive logic program P_{Tweety}.

- $\mathscr{A}_{P_{Tweety}} = \{penguin, ostrich\}$
- $\mathscr{R}_{P_{Tweety}} = \left\{ \begin{array}{l} flies(x) \leftarrow bird(x) \wedge \neg ab(x) \\ ab(x) \leftarrow penguin(x) \\ ab(x) \leftarrow ostrich(x) \\ bird(tweety). \end{array} \right\}$
- $\mathscr{I}_{P_{Tweety}} = \emptyset$.

In the remainder of this paper, no integrity constraints are used, i.e. $\mathscr{I}_P$ is always empty. Integrity constraints are used to restrict the explanations for a given observation to a smaller class of 'legal' explanations. As such, they can be seen completely separate from the 'program part' of the abductive logic program, which specifies the explanations for a given observation. In this paper, we want to concentrate on the 'program part' of abductive logic programs. That is, we want to give a semantics for it, and develop a proof procedure for it. Also, on a more practical level, a reason is that the proof procedure we propose has no way of dealing with integrity constraints in full generality. One should note however, that there exist techniques that, under certain conditions, can translate integrity constrains to some set $\mathscr{IR}_P$ of program rules with head *False* (a propositional variable). Instead of testing whether a candidate explanation δ of a problem $\langle P, \phi\rangle$ satisfies the integrity constraints, one can find an explanation of the problem $\langle P', \phi \wedge \neg False\rangle$, where P' is the program $\langle \mathscr{A}_P, \mathscr{R}_P \cup \mathscr{IR}_P, \emptyset\rangle$.

If we compare our definition of abductive logic programs with the definitions given in [17], the main difference is, that we add equality to our abducible formulas. Of course, equality is not abducible, in the sense that one can assume two terms to be equal, in order to explain an observation; we use equality in context of *CET*, which is complete when a universal language is used. However, when one thinks of the class of abducible formulas as the class of formulas that can be used to explain a given observation, it makes perfect sense to include equality. Note that also Eshghi in [13] uses a kind of equality in its abducible formulas. However, it is a restricted notion of equality, consisting of only the identity and transitivity axioms, and inequality between distinct skolem constants.

4. Completion semantics for abductive logic programs

In [4], Clark introduces the notion of *completion* of a general logic program, and proposes the (two-valued) completion semantics for general logic programs. The central notion in the definition of the completion of a program, is the notion of the *completed definition* of a predicate.

Definition 6. Let P be a program and let p be a predicate symbol in the language of P. Let n be the arity of p and let $x_1, \ldots, x_n$ be a sequence of fresh variables. Let $p(\underline{s}_1) \leftarrow \theta_1, \underline{L}_1 \ \ldots \ p(\underline{s}_m) \leftarrow \theta_m, \underline{L}_m$ be the clauses in P with head p, and let, for $i \in [1..m]$, $y_i = FreeVar(\theta_i, \underline{L}_i) - FreeVar(p(\underline{s}_i))$. The *completed definition of* p (w.r.t. P) is the formula

$$p(\underline{x}) \cong \bigvee_{i \in [1..m]} \exists_{\underline{y}_i}((\underline{x} = \underline{s}_i), \theta_i, \underline{L}_i).$$

Intuitively, the completed definition of a predicate states that 'p is true *iff* there exists a rule for p whose body is true'.

The *completion* (*comp*(P)) of a general logic program consists of the completed definitions of its predicates, plus *CET* to interpret equality correctly. In the (two-valued)

completion semantics for general logic programs, a formula is true in a program iff it is true in all (two-valued) models of the completion of that program.

In [5], Console et al. propose a two-valued completion semantics for abductive logic programs. The idea is, that the completion of an abductive logic program only contains completed definitions of non-abducible predicates. As a result, the theory $comp(P)$ contains no information on the abducible predicates (i.e. the abducible predicates can be freely interpreted).

Definition 7. Let P be an abductive logic program. The *completion of* P (denoted by $comp(P)$) is the theory that consists of CET and, for every non-abducible predicate p in P, the completed definition of p.

Example 8. Given the program P_{Tweety} of Example 5, the completed program $comp(P_{Tweety})$ consists of the following formulas

$$flies(x) \cong bird(x) \wedge \neg ab(x)$$

$$ab(x) \cong penguin(x) \vee ostrich(x)$$

$$bird(tweety) \cong \mathbf{t}.$$

plus CET.

Using this notion of completion for abductive logic programs, Console et al. give an object level characterization of the explanation of an abductive problem $\langle P, \phi \rangle$. Intuitively, it is the formula (unique up to logical equivalence) that represents all possible ways of explaining the observation in that abductive problem. Before we can give its definition, we have to introduce the notion of *most specific* abducible formula.

Definition 9. For abducible formulas θ and σ, θ is *more specific* than σ if $CET \models \theta \rightarrow \sigma$. θ is *most specific* if there does not exist a σ (different from θ, modulo logical equivalence) such that σ is more specific than θ.

We now give the definition of explanation, as proposed by Console et al. (i.e. the object level characterization of Definition 2 in [5]). As we want to reserve the term 'explanation' for an alternative notion of explanation we define later on, we use the term 'full explanation' here.

Definition 10. Let $\langle P, \phi \rangle$ be an abductive problem. Let δ be an abducible formula. Then, δ is *the full explanation of* $\langle P, \phi \rangle$, if δ is the most specific abducible formula such that $comp(P) \cup \{\phi\} \models \delta$, and $comp(P) \cup \{\delta\}$ is consistent.

Note, that in this definition ϕ and δ switched positions with respect to the ordinary characterization of abduction. The advantage of this definition is, that for a given abductive problem, the full explanation is unique (up to logical equivalence).

Example 11. Consider the program P_{Tweety} and observation $\neg flies(tweety)$. The full explanation for this observation is $penguin(tweety) \vee ostrich(tweety)$. With this simple program, this can be easily checked by using the equivalence formulas in $comp(P_{Tweety})$:

$$\begin{aligned}\neg flies(tweety) &\cong \neg(bird(tweety) \wedge \neg ab(tweety)) \\ &\cong \neg bird(tweety) \vee ab(tweety)) \\ &\cong \neg \mathbf{t} \vee penguin(tweety) \vee ostrich(tweety) \\ &\cong penguin(tweety) \vee ostrich(tweety).\end{aligned}$$

In their paper, Console et al. restrict their abductive logic programs to the class of *hierarchical programs*. As a reason for this, they argue that 'it is useless to explain a fact in terms of itself'. Practical reasons for this restriction seem to be twofold: it ensures consistency of $comp(P)$, and soundness and completeness of their 'abstract' proof procedure ABDUCE. Although we agree that, as is the case with general logic programs, a large class of naturally arising programs will turn out to be hierarchical, we do not want to restrict ourselves to hierarchical programs. Moreover, the problem of checking whether a given program is hierarchical is not always easy (see [1] for some techniques). Thus, instead of restricting ourselves to hierarchical programs, in the definition of full explanation, we added the condition that $comp(P) \cup \{\delta\}$ has to be consistent.

We now define an alternative notion of 'explanation'. This second definition is more in line with the normal characterization of abduction. However, it is also weaker, in the sense that there can exist more than one explanation for a given abductive problem.

Definition 12. Let $\langle P, \phi \rangle$ be an abductive problem. An abducible formula δ is *an explanation for* $\langle P, \phi \rangle$, if $comp(P) \cup \{\delta\} \models \phi$ and $comp(P) \cup \{\delta\}$ is consistent.

Example 13. Consider again program P_{Tweety} and *observation* $\neg$ $flies(tweety)$. Then, $penguin(tweety)$ is an explanation, and so is $ostrich(tweety)$.

Note, that in both δ and ϕ the free variables are implicitly universally quantified. Thus, there is no 'communication' between free variables in δ and ϕ. As a result, the observation $flies(x)$ does not stand for the generic 'given a hypothetical individual x, what can you tell me (about x) when I observe that x flies'. Instead, it just states that you observe that 'all x fly'.

The following lemma shows that the full explanation of a given abductive problem is less specific than any explanation for that abductive problem.

Lemma 14. *Let $\langle P, \phi \rangle$ be an abductive problem such that ϕ is ground, let δ be the full explanation of $\langle P, \phi \rangle$, and let θ be an explanation for $\langle P, \phi \rangle$. Then, CET $\models \theta \rightarrow \delta$.*

Proof. δ is the full explanation of $\langle P, \phi \rangle$, and therefore $comp(P) \cup \{\phi\} \models \delta$, which implies $comp(P) \models \phi \rightarrow \delta$. Moreover, θ is an explanation for $\langle P, \phi \rangle$, and therefore $comp(P) \cup \{\theta\} \models \phi$, which implies $comp(P) \models \theta \rightarrow \phi$. But then, it follows that $comp(P) \models \theta \rightarrow \delta$. But $\theta \rightarrow \delta$ is an abducible formula and therefore $CET \models \theta \rightarrow \delta$. □

Thus, the difference between the two kinds of explanations is, that the full explanation incorporates all possible ways of explaining a given observation, while an (ordinary) explanation is a formula that is just sufficient to explain that given observation.

In the above, we used two-valued completion as a semantics. In general logic programming, there also exists a three-valued completion semantics. In this semantics, the third truth-value models the fact that effective proof procedures cannot determine truth or falsity for all formulas. Thus, the third truth-value ($\bot$) stands for 'truth-value undetermined'. In the following example, we show how this third truth-value can be useful.

Example 15. Let us construct the program $P_{Tweety'}$ by adding to P_{Tweety} the seemingly irrelevant clause

$$p \leftarrow \neg p.$$

The completion $comp(P_{Tweety'})$ has no two-valued models. As a result, the *observation* $\neg flies(tweety)$ has no (two-valued) explanations. This problem can be solved by assigning to p the third truth-value $\bot$, i.e. by switching to a three-valued logic.

In Section 5, we will characterize Fitting semantics and Kunen semantics for abductive logic programs, using a three-valued immediate consequence operator. In the remainder of this section, we present them semantics using a model-theoretic approach.

Fitting semantics and Kunen semantics are based the same notion of completion as used in the two-valued case, but use it in the setting of three-valued models. In this three-valued setting, special care must be taken to interpret the equivalence operator, used in the completed definition of a predicate, correctly. Intuitively, this equivalence should enforce that the left-hand side and the right-hand side of the completed definition have the same truth-value. However, Kleene's three-valued equivalence ($\leftrightarrow$) stands for something like 'the truth-values of left- and right-hand sides are equal and neither one is unknown'. Therefore, instead of $\leftrightarrow$, another notion of equivalence ($\cong$) is used, which has the required truth-table (see Fig. 1). The operator $\cong$ cannot be constructed using Kleene's operators, and therefore has to be introduced separately. Its use is be restricted: it will only be used in the completed definition of a predicate. Note, that $\leftrightarrow$ and $\cong$ are equivalent when restricted to the truth-values **t** and **f**.

Using a model-theoretic approach, Fitting semantics and Kunen semantics can be stated very succinctly.

$\leftrightarrow$	t	f	$\bot$
t	t	f	$\bot$
f	f	t	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\cong$	t	f	$\bot$
t	t	f	f
f	f	t	f
$\bot$	f	f	t

Fig. 1. Kleene equivalence and strong equivalence

Definition 16. Let $\langle P, \phi \rangle$ be an abductive problem. A consistent abducible formula δ is a *three-valued explanation for* $\langle P, \phi \rangle$ *(in Fitting semantics)*, if ϕ is true in all three-valued Herbrand models of $comp(P) \cup \{\delta\}$.

Definition 17. Let $\langle P, \phi \rangle$ be an abductive problem. A consistent abducible formula δ is a *three-valued explanation for* $\langle P, \phi \rangle$ *(in Kunen semantics)*, if $comp(P) \cup \{\delta\} \models_3 \phi$.

Note, that in these definitions only consistency of δ (with respect to CET) is required. The reason is, that in three-valued completion the completed definitions of the program-rules are always consistent. In the following, when we refer to a three-valued explanation, we refer to an explanation in Kunen semantics.

From these definitions, it is easy to see that any Kunen explanation is also a Fitting explanation. The converse, however, does not hold. To get an idea of the difference, consider the following example, involving the *universal query problem* in (abductive) logic programming.

Example 18. Let P be the program:

$$\langle p(a) \leftarrow q, \{q\}, \emptyset \rangle.$$

Let ϕ be the formula $\forall_x p(x)$. Now, consider the abductive problem $\langle P, \phi \rangle$. In Fitting semantics (over the language $\mathscr{L}_P$), q is an explanation for this problem. The reason is, that in Herbrand models, domain elements are isomorphic to terms of the language. On the other hand, if we allow arbitrary (three-valued) models, we can choose richer models. For instance, consider the model M with domain $\{a, b\}$, in which a is mapped onto itself, in which q and $p(a)$ are true, but $p(b)$ is false. Clearly, M is a model of $comp(P) \cup \{q\}$. However, ϕ is not true in M, and therefore q is not an explanation for ϕ.

There is a large difference in the handling of inconsistencies between two- and three-valued completion. In the following example, we show how inconsistencies 'disappear' in three-valued completion semantics.

Example 19. Consider the abductive logic program P, with a single abducible predicate a, and the following two clauses:

$$p \leftarrow \neg p, a$$

$$q \leftarrow a.$$

Then, $comp(P) \cup \{a\}$ is obviously inconsistent in two-valued completion, because when a is true, the completed definition of p reduces to $p \cong \neg p$. Thus, among others, a is not an explanation for $\langle P, q \rangle$. However, by assigning $\perp$ to p, we can construct three-valued models of $comp(P)$, and therefore a is a three-valued explanation for $\langle P, q \rangle$.

Thus, the choice between two-valued and three-valued logic. With a three-valued logic, explanations can be inconsistent with respect to some parts of the program. In our opinion, the choice of semantics depends on your view on abductive logic programs, and the relation between abducible and non-abducible predicates. If one assumes that a program, i.e. the definition of the non-abducible predicates, can contain implicit information on the abducible predicates, in the form of potential inconsistencies, one should use two-valued completion. On the other hand, if one thinks of abducible predicates as *completely* undefined (apart from integrity constraints), or thinks that only integrity constraints should be used for constraining the abducible predicates, one can use three-valued completion, because then inconsistencies are the result of flaws in the program. But even if one thinks that the two-valued semantics is the proper one, Kunen's three-valued semantics remains interesting, because it describes the explanations that can actually be computed using a SLD-like proof procedure. One would, however, have to prune those explanations that are inconsistent with respect to the program.

5. Three-valued completion semantics

In Definition 17 of the previous section, we generalized Kunen semantics to abductive logic programs. The definition as given there is, however, very succinct. For one thing, it does not express the intention behind both Fitting and Kunen semantics. That is, that the third truth-value stands for something like 'truth-value not determined'.

In [16], Fitting proposes the use of three-valued semantics for general logic programs, using the third truth-value ($\perp$) to represent the fact that for some formulas, the truth-value cannot be determined. For this purpose, Fitting introduced a three-valued immediate consequence operator Φ_P, to characterize the meaning of a general logic program. He proves that the fixpoints of this operator are three-valued Herbrand models of the completed program. He takes the least fixpoint of this operator as the meaning of a general logic program (*Fitting semantics*). However, as Fitting points out, in general this semantics is highly non-constructive: the closure ordinal for the least fixpoint can be as high as ω_1, the first non-recursive ordinal.

In [20], Kunen proposes a semantics in which the iteration of Fitting's immediate consequence operator is cut-off at ordinal ω. Moreover, he proves that a sentence ϕ is true in his semantics iff ϕ is true in all three-valued models of $comp(P)$.

In the following sections, we define an immediate consequence operator for abductive logic programs, and use it to characterize Fitting semantics and Kunen semantics for

abductive logic programs. In the process, we also generalize Shepherdson's truth- and falseness formulas (see [23]).

6. The immediate consequence operator

Let us now define an three-valued immediate consequence operator for abductive logic programs. For general logic programs, the immediate consequence operator Φ_P operates on models, and $\Phi_P(M)$ denotes the one-step consequences of M, given a program P. If we would use this operator on an abductive logic program, this operator would generate all observations that need no explanation (i.e. are explained by the formula **t**). We however, want to build an operator that generates all observation ϕ that are explained by some observation δ. Therefore, we define an operator $\Phi_{P,\delta}$, such that $\Phi_{P,\delta}(\mathcal{M})$ denotes the one-step consequences of $\mathcal{M}$, given an abductive logic program P and an explanation δ. So, we compute immediate consequences in P, under the assumption that δ holds. One problem is, that for an arbitrary abducible formula δ, δ cannot be characterized by a single model. For instance, if δ is of the form $p(a) \vee p(b)$, it has two minimal models. Therefore, $\Phi_{P,\delta}$ will operate on sets of models. In [16, 20], Φ_P operates on Herbrand models. We however follow Doets [9], and define the operators on arbitrary J-models, given an algebra J. If we add a ^{J} to the operators, they operate on J-models. Without a ^{J}, they operate on Herbrand models (i.e. HA is our 'default' algebra).

Thus, the operator $\Phi_{P,\delta}$ operates on sets of models. To facilitate its definition and various proofs, we define the operator $\Phi_{P,\delta}$ in two steps. First, we define an operator $\Phi_{P,\Delta}$, which operates on models. Then, in the second step, we define $\Phi_{P,\delta}$ in terms of $\Phi_{P,\Delta}$. In $\Phi_{P,\Delta}$, a model Δ models the abducible predicates of P. The idea is that, because Δ is a model instead of an abducible formula, the set of immediate consequences of a model M in P under assumption Δ can be characterized by a single model. Because we want Δ to model the abducible predicates only, we first have to introduce the notion of *abducible models*.

Definition 20. Let P be a program. A model M is an *abducible model* (w.r.t. P), if all non-abducible atoms in P are mapped to $\bot$ in M.

Example 21. Given the program P_{Tweety}, Let us define the Herbrand model M_{Tweety} as follows:
- $penguin(tweety)$ is **t** in M_{Tweety},
- all other ground abducible atoms are **f** in M_{Tweety}, and
- all ground non-abducible atoms are $\bot$ in M_{Tweety}.

Then M_{Tweety} is an abducible model (w.r.t. P_{Tweety}).

Now, the definition of $\Phi_{P,\Delta}$ is a straightforward generalization of the operator Φ_P for general logic programs. For non-abducible atoms, the definition stays the same.

However, for an abducible atom A, A is **t** (resp. **f**) in $\Phi_{P,\Delta}(M)$ iff it is **t** (resp. **f**) in Δ.

Definition 22. Let P be a program. Let J be an algebra and let Δ be a abducible J-model. The three-valued immediate consequence operator $\Phi^J_{P,\Delta}$ is defined as follows:

- $\Phi^J_{P,\Delta}(M)(A) = \mathbf{t}$ iff $\Delta \models_3 A \vee$
  $\exists A \leftarrow \theta, \underline{L} \in J - ground(P) : \Delta \models_3 \theta \wedge M \models_3 \underline{L}$
- $\Phi^J_{P,\Delta}(M)(A) = \mathbf{f}$ iff $\Delta \models_3 \neg A \vee$
  $\forall A \leftarrow \theta, \underline{L} \in J - ground(P) : \Delta \models_3 \neg\theta \vee M \models_3 \neg\underline{L}$.

The powers of $\Phi^J_{P,\Delta}$ are defined as follows:

$$\Phi^J_{P,\Delta} \uparrow \alpha = \begin{cases} \Delta & \text{if } \alpha = 0 \\ \Phi^J_{P,\Delta}(\Phi^J_{P,\Delta} \uparrow n-1) & \text{if } \alpha \text{ is a successor ordinal} \\ \bigcup_{\beta<\alpha} \Phi^J_{P,\Delta} \uparrow \beta & \text{if } \alpha \text{ is a limit ordinal.} \end{cases}$$

Note that this definition is not standard for $\alpha = 0$. We could define $\Phi^J_{P,\delta} \uparrow 0$ to be the empty set, but at the cost of having a special treatment of the base case in some of the lemmas.

Example 23. Given the program P_{Tweety} and abducible model M_{Tweety}, we can observe that:

- *penguin*(*tweety*) is **t** in $\Phi^{HA}_{P_{Tweety},M_{Tweety}} \uparrow 0$, therefore
- *ab*(*tweety*) becomes **t** in $\Phi^{HA}_{P_{Tweety},M_{Tweety}} \uparrow 1$, and therefore
- *flies*(*tweety*) becomes **f** in $\Phi^{HA}_{P_{Tweety},M_{Tweety}} \uparrow 2$.

Now, we can define $\Phi_{P,\delta}$. We will not define $\Phi_{P,\delta}(\mathcal{M})$ for arbitrary sets of models $\mathcal{M}$. Instead, we only define $\Phi_{P,\delta} \uparrow \alpha$, for arbitrary ordinals α.

Definition 24. Let P be a program and let δ be a consistent abducible formula. Let J be an algebra and let $\mathcal{M}$ be the set of abducible J-models of $\{\delta\}$. Then,

$$\Phi^J_{P,\delta} \uparrow \alpha = \{\Phi^J_{P,\Delta} \uparrow \alpha \mid \Delta \in \mathcal{M}\}.$$

In [23], Shepherdson defines the notion of truth- and falseness formulas. These formulas give an elegant alternative characterization of what is computed by the immediate consequence operator. We generalize these formulas to abductive logic programs.

Definition 25. Let P be a program. For a natural number n and a formula ϕ, we define the formulas $T_n(\phi)$ and $F_n(\phi)$ as follows:

- If ϕ is an abducible formula, then for all n

$$T_n(\phi) \stackrel{def}{=} \phi, \quad F_n(\phi) \stackrel{def}{=} \neg\phi.$$

– If ϕ is an atom of the form $p(\underline{s})$, where p is a non-abducible predicate, then $comp(P)$ contains a definition $p(\underline{x}) \cong \psi$, where $Free\ Vars(\psi) = \underline{x}$. We define

$$T_0(\phi) \stackrel{def}{=} \mathbf{f}, \quad F_0(\phi) \stackrel{def}{=} \mathbf{f}.$$

and

$$T_n(\phi) \stackrel{def}{=} T_{n-1}(\underline{x} = \underline{s} \wedge \psi), \quad F_n(\phi) \stackrel{def}{=} F_{n-1}(\underline{x} = \underline{s} \wedge \psi).$$

– If ϕ is a complex formula, we define

$$\begin{aligned}
&T_n(\neg\phi) \stackrel{def}{=} F_n(\phi), && F_n(\neg\phi) \stackrel{def}{=} T_n(\phi)\\
&T_n(\phi \wedge \psi) \stackrel{def}{=} T_n(\phi) \wedge T_n(\psi), && F_n(\phi \wedge \psi) \stackrel{def}{=} F_n(\phi) \vee F_n(\psi)\\
&T_n(\phi \vee \psi) \stackrel{def}{=} T_n(\phi) \vee T_n(\psi), && F_n(\phi \vee \psi) \stackrel{def}{=} F_n(\phi) \wedge F_n(\psi)\\
&T_n(\phi \rightarrow \psi) \stackrel{def}{=} F_n(\phi) \vee T_n(\psi), && F_n(\phi \rightarrow \psi) \stackrel{def}{=} T_n(\phi) \wedge F_n(\psi)\\
&T_n(\forall \underline{x}\phi) \stackrel{def}{=} \forall \underline{x} T_n(\phi), && F_n(\forall \underline{x}\phi) \stackrel{def}{=} \exists \underline{x} F_n(\phi)\\
&T_n(\exists \underline{x}\phi) \stackrel{def}{=} \exists \underline{x} T_n(\phi), && F_n(\exists \underline{x}\phi) \stackrel{def}{=} \forall \underline{x} F_n(\phi).
\end{aligned}$$

Example 26. Given the program P_{Tweety}, we have that

$$\begin{aligned}
T_2(\neg flies(tweety)) &= F_2(flies(tweety))\\
&= F_1(bird(tweety) \wedge \neg ab(tweety))\\
&= F_1(bird(tweety)) \wedge F_1(\neg ab(tweety))\\
&= F_1(\mathbf{t}) \wedge T_1(ab(tweety))\\
&= \mathbf{f} \wedge T_0(penguin(tweety) \vee ostrich(tweety))\\
&= penguin(tweety) \vee ostrich(tweety).
\end{aligned}$$

The following lemma is a generalization of Lemma 4.1 in [23] to abductive logic programs.

Lemma 27. *Let P be a program. Let J be an algebra with domain D, let Δ be an abducible J-model and let ϕ be a D-sentence. Then, for all natural numbers n,*

(i) $\Phi^J_{P,\Delta} \uparrow n \models_3 \phi$ *iff* $\Delta \models_3 T_n(\phi)$,
(ii) $\Phi^J_{P,\Delta} \uparrow n \models_3 \neg\phi$ *iff* $\Delta \models_3 F_n(\phi)$.

Proof. We prove the lemma by induction on n and formula induction on ϕ.

Suppose ϕ is an abducible formula. Then $T_n(\phi) = \phi$ and $F_n(\phi) = \neg\phi$. So, we only have to prove that $\Phi^J_{P,\Delta} \uparrow n \models_3 \phi$ iff $\Delta \models_3 \phi$. This follows directly from the construction of $\Phi^J_{P,\Delta}$.

Suppose $n = 0$ and ϕ is a non-abducible atom $p(\underline{s})$. Then, by definition, $p(\underline{s})$ is $\bot$ in $\Phi^J_{P,\Delta} \uparrow 0$ and $T_0(p(\underline{s})) = F_0(p(\underline{s})) = \mathbf{f}$. Therefore, the claims hold.

Assume that the lemma holds for all $m < n$. Suppose ϕ is the atom $p(\underline{s})$. Because ϕ is a D-sentence, $p(\underline{s})$ is J-ground. Because p is a non-abducible predicate, $comp(P)$ contains a definition $p(\underline{x}) \cong \psi$. Now,

$\Delta \models_3 T_n(p(\underline{s}))$ by definition of $T_n(p(\underline{s}))$
iff $\Delta \models_3 T_{n-1}(\underline{x} = \underline{s} \wedge \psi)$ by induction hypothesis
iff $\Phi^J_{P,\Delta} \uparrow n-1 \models_3 \underline{x} = \underline{s} \wedge \psi$ by construction of ψ
iff $\exists\ p(\underline{s}) \leftarrow \underline{L} \in J\text{-}ground(P) : \Phi^J_{P,\Delta} \uparrow n-1 \models_3 \underline{L}$
by construction of $\Phi^J_{P,\Delta}$
iff $\Phi^J_{P,\Delta} \uparrow n \models_3 p(\underline{s})$

The reasoning for $F_n(p(\underline{s}))(\underline{a})$ is similar.

If ϕ is of the form $\neg\psi, \psi \wedge \eta, \psi \vee \eta$ or $\psi \rightarrow \eta$, the claim follows from the construction of $T_n(\phi)$ and $F_n(\phi)$.

Suppose ϕ is of the form $\exists x\psi$. Then, $\Phi^J_{P,\Delta} \uparrow n \models_3 \exists x\psi$ iff, for some element a of the domain of J, $\Phi^J_{P,\Delta} \uparrow n \models_3 \psi(a)$. Because $\psi(a)$ is a D-sentence, we have by induction that $\Phi^J_{P,\Delta} \uparrow n \models_3 \psi(a)$ iff $\Delta \models_3 T_n(\psi(a))$. Finally, we have that $\Delta \models_3 T_n(\psi(a))$ iff $\Delta \models_3 T_n(\exists x\psi)$.

The other cases with quantifiers are similar. □

Corollary 28. *Let P be a program and let δ be a consistent abducible formula. Let J be an algebra with domain D and let ϕ be a D-sentence. Then,*

(i) $\Phi^J_{P,\delta} \uparrow n \models_3 \phi$ *iff* $J \cup \{\delta\} \models_3 T_n(\phi)$,
(ii) $\Phi^J_{P,\delta} \uparrow n \models_3 \neg\phi$ *iff* $J \cup \{\delta\} \models_3 F_n(\phi)$.

Proof. The proof follows immediately from the fact that $J \cup \{\delta\} \models_3 \phi$ iff ϕ is true in all abducible J-models of $\{\delta\}$. □

7. Fitting semantics for abductive logic programs

In this section, we use the three-valued consequence operator defined in the previous section to generalize Fitting semantics to abductive logic programs.

Definition 29. Let $\langle P, \phi\rangle$ be an abductive problem. Let δ be a consistent abducible formula. Let $\mathcal{M}$ be the least fixpoint of $\Phi^{HA}_{P,\delta}$. Then, δ is an *explanation* for $\langle P, \phi\rangle$ *in the Fitting semantics*, if $\mathcal{M} \models_3 \phi$.

With Fitting semantics for general logic programs, a formula is true in the Fitting semantics iff it is true in all three-valued Herbrand models. The same holds for Fitting semantics for abductive logic programs. In order to prove this, we first present two lemmas. First of all, the following lemma shows that the fixpoints of $\Phi_{P,\Delta}$ are indeed three-valued models of $comp(P) \cup \{\delta\}$.

Lemma 30. *Let P be a program and let δ be a consistent abducible formula. Let J be an algebra, let Δ be an abducible J-model of $\{\delta\}$ and let M be a J-model. If $\Phi^J_{P,\Delta}(M) = M$ then $M \models_3 comp(P) \cup \{\delta\}$.*

Proof. Suppose that $\Phi^J_{P,\Delta}(M) = M$. The fact that M is a model of $\{\delta\}$ follows trivially from the definition of $\Phi^J_{P,\Delta}$. We have to prove that $M \models_3 comp(P)$.

Let $p(\underline{x}) \cong \psi$ be a formula in $comp(P)$. Let $p(\underline{a})$ be a J-ground atom. Then,

$$
\begin{array}{ll}
M \models_3 \psi(\underline{a}) & \text{by definition of } \psi \\
\text{iff } \exists\ p(\underline{a}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : M \models_3 \underline{L} & \text{by definition of } \Phi^J_{P,\Delta} \\
\text{iff } \Phi^J_{P,\Delta}(M) \models_3 p(\underline{a}) & \text{because } \Phi^J_{P,\Delta}(M) = M \\
\text{iff } M \models_3 p(\underline{a}) &
\end{array}
$$

and

$$
\begin{array}{ll}
M \models_3 \neg\psi(\underline{a}) & \text{by definition of } \psi \\
\text{iff } \forall\ p(\underline{a}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : M \models_3 \neg\underline{L} & \text{by definition of } \Phi^J_{P,\Delta} \\
\text{iff } \Phi^J_{P,\Delta}(M) \models_3 \neg p(\underline{a}) & \text{because } \Phi^J_{P,\Delta}(M) = M \\
\text{iff } M \models_3 \neg p(\underline{a}) \quad \square &
\end{array}
$$

Corollary 31. *Let P be a program and let δ be a consistent abducible formula. Let J be an algebra. If $\mathcal{M}$ is a fixpoint of $\Phi^J_{P,\delta}$, then $\mathcal{M} \models_3 comp(P) \cup \{\delta\}$.*

In the second lemma, we prove the converse. For this, we need the following definition.

Definition 32. Let P be a program and let M be a model. The *abducible projection* of M is the abducible model Δ such that
- $\Delta(A) = M(A)$, *if A is an abducible atom, and*
- $\Delta(A) = \bot$, *otherwise.*

Lemma 33. *Let P be a program and let δ be an abducible formula. Let J be an algebra and let M be a J-model such that $M \models_3 comp(P) \cup \{\delta\}$. Let Δ be the abducible projection of M. Then, M is a fixpoint of $\Phi^J_{P,\Delta}$.*

Proof. Suppose that $M \models_3 comp(P) \cup \{\delta\}$.
We have to prove that $\Phi^J_{P,\Delta}(M) = M$.
- If L is an abducible J-ground literal, $\Delta \models_3 L$ iff $M \models_3 L$, and therefore, by definition of $\Phi^J_{P,\Delta}$, $\Phi^J_{P,\Delta}(M) \models_3 L$ iff $M \models_3 L$.

– If $p(\underline{a})$ is a non-abducible J-ground atom, there exists a J-ground instance $p(\underline{a}) \cong \psi$ of a formula in $comp(P)$ such that

$$\begin{aligned}
&\Phi^J_{P,\Delta}(M) \models_3 p(\underline{a}) && \text{by definition of } \Phi^J_{P,\Delta}\\
&\text{iff } \exists\ p(\underline{a}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : M \models_3 \underline{L} && \text{by definition of completion}\\
&\text{iff } M \models_3 \psi && \text{because } M \models_3 comp(P)\\
&\text{iff } M \models_3 p(\underline{a})
\end{aligned}$$

and

$$\begin{aligned}
&\Phi^J_{P,\Delta}(M) \models_3 \neg p(\underline{a}) && \text{by definition of } \Phi^J_{P,\Delta}\\
&\text{iff } \forall\ p(\underline{a}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : M \models_3 \neg\underline{L} && \text{by definition of completion}\\
&\text{iff } M \models_3 \neg\psi && \text{because } M \models_3 comp(P)\\
&\text{iff } M \models_3 \neg p(\underline{a}) \qquad \square
\end{aligned}$$

Theorem 34. *Let $\langle P, \phi\rangle$ be an abductive problem. A consistent abducible formula δ is an explanation for $\langle P, \phi\rangle$ in the Fitting semantics iff ϕ is true in all three-valued Herbrand models of $comp(P) \cup \{\delta\}$.*

Proof. Let $\mathcal{M}$ be the least fixpoint of $\Phi^{HA}_{P,\delta}$.

($\Leftarrow$) This follows directly from Lemma 30: as a fixpoint of $\Phi^J_{P,\Delta}$ is a J-model, take J to be HA, and we have that the fixpoints of $\Phi^{HA}_{P,\delta}$ are subsets of the set of Herbrand models of $comp(P) \cup \{\delta\}$.

($\Rightarrow$) Let M' be an arbitrary Herbrand model of $comp(P) \cup \{\delta\}$, and let Δ be its abducible projection. By Lemma 33, M' is a fixpoint of $\Phi^{HA}_{P,\Delta}$. Moreover, Δ is an abducible HA-model of $\{\delta\}$. As a result, for some fixpoint $\mathcal{M}'$ of $\Phi^{HA}_{P,\delta}$, $M' \in \mathcal{M}'$. Because $\mathcal{M}$ is the least fixpoint of $\Phi^{HA}_{P,\delta}$, there exists a $M \in \mathcal{M}$ such that $M' \models_3 M$. But then, if ϕ is true in $\mathcal{M}$, it is true in M, and therefore in M', which is what we started with; an arbitrary Herbrand model of $comp(P) \cup \{\delta\}$. $\square$

8. Kunen semantics for abductive logic programs

In this section, we propose a Kunen semantics for abductive logic programs. In [20], Kunen proposes to cut off iteration of the immediate consequence operator at ordinal ω, instead of continuing until the least fixpoint is reached. Generalizing this idea to abductive logic programming, we get the following semantics.

Definition 35. Let $\langle P, \phi\rangle$ be an abductive problem. Let δ be a consistent abducible formula. Then, δ is an explanation for $\langle P, \phi\rangle$ *in the Kunen semantics* if, for some natural number n, $\Phi^{HA}_{P,\delta} \uparrow n \models_3 \phi$.

Note, that this definition differs from Definition 17. The remainder of this section is dedicated to proving that these two definitions give rise to the same se-

mantics (Theorem 42). In his proof of Theorem 6.3 in [20], Kunen makes heavy use of ultra-products. We base our proofs on an alternative proof given by Doets in [9].

The larger part of the work is done in the proof of Theorem 36, which proves one direction of the desired result for the operator $\Phi_{P,\Delta}$. Basically, with this result on $\Phi_{P,\Delta}$, we have proven the result for $\Phi_{P,\delta}$, for the case where δ is a conjunction of abducible literals (i.e. has a minimal model over any algebra). The remainder of the proof of Theorem 42 is concerned with extending this result to the case where δ is an arbitrary abducible sentence, and proving the other direction of the desired result.

Theorem 36. *Let P be a program and let ϕ be a sentence. Let δ be a consistent abducible formula and let Δ be an abducible HA-model of $\{\delta\}$. Then, if $comp(P)\cup\{\delta\} \models_3 \phi$, for some natural number n, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$.*

The proof of this theorem closely resembles the proof of Corollary 8.37 in [9]. It is organized as follows. In Lemma 37 we show that we can replace J with an elementary extension of J. Then, in Lemma 39 we show that for certain elementary extensions J of HA, $\Phi^{J}_{P,\Delta}$ is *continuous*. In Lemma 40 we show that for certain elementary extensions J of HA, $\Phi^{J}_{P,\Delta} \uparrow \omega$ is a least fixpoint. From these lemmas, and from the fact that, by properties 1 and 2 stated below (see [3]), we know these desired elementary extensions of HA exist, we can prove Theorem 36.

Lemma 37. *Let P be a program. Let J be an elementary extension of HA, let Δ be an abducible HA-model and let Δ' be an elementary J-extension of Δ. For every sentence ϕ and natural number n, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$ iff $\Phi^{J}_{P,\Delta'} \uparrow n \models_3 \phi$.*

Proof. By Lemma 27, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$ iff $\Delta \models_3 T_n(\phi)$. Because Δ' is an elementary extension of Δ, and $T_n(\phi)$ is a sentence, $\Delta \models_3 T_n(\phi)$ iff $\Delta' \models_3 T_n(\phi)$. Again, by Lemma 27, $\Delta' \models_3 T_n(\phi)$ iff $\Phi^{J}_{P,\Delta'} \uparrow n \models_3 \phi$. □

For Lemmas 39 and 40, we need the following definitions and results from model theory, concerning recursively saturated models.

Definition 38. Let $\Psi = \{\psi^i \mid i \in \mathbb{N}\}$ be a sequence of formulas ψ^i in finitely many free variables $x_1,\ldots,x_k,y_1,\ldots,y_m$ and let M be a two-valued model. M is called *Ψ-saturated* if, for every sequence $a_1,\ldots,a_m$ of domain elements, either
- $\{\psi^i\{\underline{y}/\underline{a}\} \mid i \in \mathbb{N}\}$ is satisfiable in M, or
- there exist a natural number N such that $\{\psi^i\{\underline{y}/\underline{a}\} \mid i < N\}$ is not satisfiable in M. M is called *saturated* if it is Ψ-saturated for every sequence Ψ. M is called *recursively saturated* if it is Ψ-saturated for every computable sequence Ψ.

Property 1. Every countable model has a countable recursively saturated elementary extension

Property 2. Let $\Psi = \{\psi^i \mid i \in \mathbb{N}\}$ be a sequence of sentences with free variable x. Let M be a recursively saturated model and let A be the domain of M. Then,

$$\forall_{a\in A}\exists_n M \models \psi^i(a) \text{ implies } \exists_n \forall_{a\in A} M \models \psi^i(a)$$

Lemma 39. *Let P be a program. Let J be a recursively saturated algebra with domain D and let Δ be an abducible J-model. Let ϕ be a D-sentence. If ϕ is* **t** *(resp.* **f***) in $\Phi^J_{P,\Delta} \uparrow \omega$, then, for some natural number n, ϕ is* **t** *(resp.* **f***) in $\Phi^J_{P,\Delta} \uparrow n$.*

Proof. The proof is by induction on the complexity of ϕ. Only when ϕ is of the form $\forall y\psi$ or $\exists y\psi$, the proof is non-trivial, and we can write $\exists y\psi$ as $\neg\forall y\neg\psi$. Let A be the domain of J.

Assume that $\forall y\psi$ is **t** in $\Phi^J_{P,\Delta} \uparrow \omega$. Then, for all $a \in A$, $\psi(a)$ is **t** in $\Phi^J_{P,\Delta} \uparrow \omega$. By induction hypothesis, for all $a \in A$, there exists an n such that $\psi(a)$ is **t** in $\Phi^J_{P,\Delta} \uparrow n$. But then, by Lemma 27, for all $a \in A$, there exists an n such that $T_n(\psi)(a)$ is **t** in Δ. Because J is recursively saturated, by Lemma 2 there exists an n such that for all $a \in A$ $T_n(\psi)(a)$ is **t** in Δ. But then, $T_n(\forall y\psi)$ is **t** in Δ and therefore by Lemma 27, $\forall y\psi$ is **t** in $\Phi^J_{P,\Delta} \uparrow n$.

Assume that $\forall y\psi$ is **f** in $\Phi^J_{P,\Delta} \uparrow \omega$. Then, for some $a \in A$, $\psi(a)$ is **f** in $\Phi^J_{P,\Delta} \uparrow \omega$. By induction hypothesis, for some $a \in A$, there exists an n such that $\psi(a)$ is **f** in $\Phi^J_{P,\Delta} \uparrow n$. But then, $\forall y\psi$ is **f** in $\Phi^J_{P,\Delta} \uparrow n$. □

Lemma 40. *Let P be a program. Let J be a recursively saturated CET-algebra and let Δ be an abducible J-model. Then, $lfp(\Phi^J_{P,\Delta}) = \Phi^J_{P,\Delta} \uparrow \omega$.*

Proof. We have to prove for an arbitrary $J-$ground atom A that, whenever $\Phi^J_{P,\Delta} \uparrow \omega + 1(A) = \mathbf{t}$, then $\Phi^J_{P,\Delta} \uparrow \omega(A) = \mathbf{t}$, and if $\Phi^J_{P,\Delta} \uparrow \omega + 1(A) = \mathbf{f}$, then $\Phi^J_{P,\Delta} \uparrow \omega(A) = \mathbf{f}$.

For abducible atoms, the claims hold trivially, because then $\Phi^J_{P,\Delta} \uparrow \alpha(A) = \mathbf{t}$ (resp. **f**) iff $\Delta \models_3 A$ (resp. $\Delta \models_3 \neg A$).

Suppose $p(\underline{s})$ is a non-abducible $J-$ground atom.

- Suppose $p(\underline{s})$ is **t** in $\Phi^J_{P,\Delta} \uparrow \omega + 1$. Then, there exists a J-ground instance $p(\underline{s}) \leftarrow \underline{L}$ of a clause in P such that $\Phi^J_{P,\Delta} \uparrow \omega \models_3 \underline{L}$. But then by Lemma 39, there exists a natural number n such that $\Phi^J_{P,\Delta} \uparrow n \models_3 \underline{L}$, and therefore $p(\underline{s})$ is **t** in $\Phi^J_{P,\Delta} \uparrow n + 1$. Thus, $p(\underline{s})$ is **t** in $\Phi^J_{P,\Delta} \uparrow \omega$.
- Suppose $p(\underline{s})$ is **f** in $\Phi^J_{P,\Delta} \uparrow \omega + 1$. Let $p(\underline{t}_1) \leftarrow \underline{L}_1 \ldots p(\underline{t}_k) \leftarrow \underline{L}_k$ be the clauses in P defining p. Then, for all $i \in [1..k]$, $\Phi^J_{P,\Delta} \uparrow \omega \models_3 \neg(\underline{s} = \underline{t}_i \wedge \underline{L}_i)$. Because $\neg(\underline{s} = \underline{t}_i \wedge \underline{L}_i)$ is quantifier-free, it is equivalent to its universal closure. But for all $i \in [1..k]$, $\forall\neg(\underline{s} = \underline{t}_i \wedge \underline{L}_i)$ is a D-sentence (where D is the domain of J), and therefore by Lemma 39 there exists an n_i such that $\Phi^J_{P,\Delta} \uparrow n_i \models_3 \forall\neg(\underline{s} = \underline{t}_i \wedge \underline{L}_i)$. Because k

is finite, there exists an n such that, for all $i \in [1..k]$, we have that $\Phi^J_{P,\Delta} \uparrow n \models \neg(\underline{s} = \underline{t}_i \wedge \underline{L}_i)$. By construction of $\Phi^J_{P,\Delta}$, we have that $p(\underline{s})$ is **f** in $\Phi^J_{P,\Delta} \uparrow n+1$ and therefore, $p(\underline{s})$ is **f** in $\Phi^J_{P,\Delta} \uparrow \omega$. □

Before proving Theorem 36, we combine the preceding two lemmas in the following corollary.

Corollary 41. *Let P be a program and let δ be a consistent abducible formula. Let J be a recursively saturated CET-algebra and let Δ be an abducible J-model of $\{\delta\}$. Let ϕ be a sentence. If $comp(P) \cup \{\delta\} \models_3 \phi$, then for some n $\Phi^J_{P,\Delta} \uparrow n \models_3 \phi$.*

Proof. By Lemmas 30 and 40, $\Phi^J_{P,\Delta} \uparrow \omega$ is a three-valued model of $comp(P) \cup \{\delta\}$, and therefore $\Phi^J_{P,\Delta} \uparrow \omega \models_3 \phi$. Therefore, by Lemma 39 there exists a finite n such that $\Phi^J_{P,\Delta} \uparrow n \models_3 \phi$. □

Proof of Theorem 36. Suppose that $comp(P) \cup \{\delta\} \models_3 \phi$. By property 8, there exists a recursively saturated elementary extension J of HA. Because J is an extension of HA, it is a CET-algebra. Again, by property 8, there exists an elementary J-extension Δ' of Δ. By Corollary 14, there exists a finite n such that $\Phi^J_{P,\Delta'} \uparrow n \models_3 \phi$. Finally, by Lemma 37, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$. □

Thus, for $\Phi_{P,\Delta}$, we have proven the one direction of the desired result. In the following theorem, we prove that the desired correspondence holds for $\Phi_{P,\delta}$.

Theorem 42. *Let P be a program and let δ be a consistent abducible sentence. Let ϕ be a sentence. Then, $comp(P) \cup \{\delta\} \models_3 \phi$ iff, for some finite n, $\Phi^{HA}_{P,\delta} \uparrow n \models_3 \phi$.*

Before proving the theorem, we first need to prove two lemmas. The first one states that, in some sense, the operator $\Phi_{P,\delta}$ behaves 'monotonically' with respect to the assumption δ.

Lemma 43. *Let P be a program and let δ and σ be consistent abducible formulas. Let J be an algebra. If $J \models_3 \delta \rightarrow \sigma$ then, for all natural numbers n, $\Phi^J_{P,\delta} \uparrow n \models_3 \Phi^J_{P,\sigma} \uparrow n$.*

Proof. It suffices to prove that, for all natural numbers n, $M \in \Phi^J_{P,\delta} \uparrow n$ implies $M \in \Phi^J_{P,\sigma} \uparrow n$.

Suppose that $M \in \Phi^J_{P,\delta} \uparrow n$. Then, for some abducible J-model Δ of $\{\delta\}$, $M = \Phi^J_{P,\Delta} \uparrow n$. But because $J \models_3 \delta \rightarrow \sigma$, Δ is also an abducible J-model of $\{\sigma\}$. Therefore, $M \in \Phi^J_{P,\sigma} \uparrow n$. □

Lemma 44. *Let P be a program and let δ be a consistent abducible formula. Let ϕ be a sentence and let J be a recursively saturated CET-algebra. Then, $comp(P) \cup \{\delta\} \models_3 \phi$ implies that, for some finite n, $\Phi^J_{P,\delta} \uparrow n \models_3 \phi$.*

Proof. $comp(P) \cup \{\delta\} \models_3 \phi$ implies that $comp(P) \models_3 \delta \rightarrow \phi$. Let σ be an abducible formula which is a tautology, and let Δ be the least abducible J-model of σ. By Corollary 41, there exists a finite n such that $\Phi^J_{P,\Delta} \uparrow n \models_3 \delta \rightarrow \phi$. Because Δ is the least abducible J-model of $\{\sigma\}$, we have that $\Phi^J_{P,\sigma} \uparrow n \models_3 \delta \rightarrow \phi$ iff $\Phi^J_{P,\Delta} \uparrow n \models_3 \delta \rightarrow \phi$. Moreover, because $J \models_3 \delta \rightarrow \sigma$, it follows by Lemma 43 that $\Phi^J_{P,\delta} \uparrow n \models_3 \delta \rightarrow \phi$. Finally, because we know that $\Phi^J_{P,\delta} \uparrow n \models_3 \delta$, it follows that $\Phi^J_{P,\delta} \uparrow n \models_3 \phi$. □

Proof of Theorem 42. ($\Rightarrow$) Suppose that $comp(P) \cup \{\delta\} \models_3 \phi$. By property 8, there exists a recursively saturated elementary extension J of HA. Because J is an extension of HA, it is a CET-algebra. By Lemma 44 there exists an n such that $\Phi^J_{P,\delta} \uparrow n \models_3 \phi$. Let Δ be an arbitrary abducible HA-model of $\{\delta\}$. By property 8, there exists an elementary J-extension Δ' of Δ. Because Δ' is an elementary extension of Δ, δ is a sentence and $\Delta \models_3 \delta$, it follows that $\Delta' \models_3 \delta$. Therefore, it follows from $\Phi^J_{P,\delta} \uparrow n \models_3 \phi$ that $\Phi^J_{P,\Delta'} \uparrow n \models_3 \phi$. But then, by Lemma 37, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$. Thus, for arbitrary Herbrand models Δ of δ, we have that $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$. But then, also $\Phi^{HA}_{P,\delta} \uparrow n \models_3 \phi$.

($\Leftarrow$) The proof is by induction on n. For $n = 0$, we have that $\Phi^{HA}_{P,\Delta} \uparrow 0 \models_3 \phi$ implies that ϕ is an abducible formula and that $HA \cup \{\delta\} \models_3 \phi$. Because δ and ϕ are sentences and every model of CET is an extension of a Herbrand model, $CET \cup \{\delta\} \models_3 \phi$ and therefore $comp(P) \cup \{\delta\} \models_3 \phi$.

Assume that the claim holds for all $m < n$. If $p(\underline{s})$ is a non-abducible J-ground atom, there exists a J-ground instance $p(\underline{a}) \cong \psi$ of a formula in $comp(P)$ such that $\Phi^J_{P,\delta} \uparrow n \models_3 p(\underline{s})$ iff $comp(P) \cup \{\delta\} \models_3 p(\underline{s})$.

$\Phi^J_{P,\delta} \uparrow n \models_3 p(\underline{s})$
by definition of $\Phi^J_{P,\delta}$
iff $\exists\ p(\underline{s}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : \Phi^J_{P,\delta} \uparrow n - 1 \models_3 \underline{L}$
by induction hypothesis
then $\exists\ p(\underline{s}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : comp(P) \cup \{\delta\} \models_3 \underline{L}$
by definition of completion
iff $comp(P) \cup \{\delta\} \models_3 p(\underline{s})$

and

$\Phi^J_{P,\delta} \uparrow n \models_3 \neg p(\underline{s})$
by definition of $\Phi^J_{P,\delta}$
iff $\forall\ p(\underline{s}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : \Phi^J_{P,\delta} \uparrow n - 1 \models_3 \neg \underline{L}$
induction hypothesis
then $\forall\ p(\underline{s}) \leftarrow \underline{L} \in J\text{-}ground(\mathrm{P}) : comp(P) \cup \{\delta\} \models_3 \neg \underline{L}$
definition of completion
iff $comp(P) \cup \{\delta\} \models_3 \neg p(\underline{s})$

For complex sentences, the proof is by structural induction. □

Thus, we have proven that Definitions 17 and 35 give rise to the same semantics. In the following section, we present a proof procedure for this semantics.

9. Generalizing SLDFA-resolution

In [7], Denecker and DeSchreye propose a proof procedure which is sound with respect to the two-valued completion semantics of [5]: SLDNFA-resolution (a proof procedure for abductive logic programs based on SLDNF-resolution). The semantics they use, is the two-valued completion semantics for abductive logic programs, proposed by Console et al. in [5]. In this paper, we propose an alternative proof procedure, which is based upon SLDFA-resolution; a proof procedure for general logic programs proposed by Drabent [11]. This proof procedure solves some problems associated with SLDNFA-resolution. First of all, by using constructive negation instead of negation as failure, we remove the problem of *floundering*. Secondly, instead of skolemizing non-ground queries, which introduces some technical problems, we use equality in our language, which allows a natural treatment of non-ground queries.

In the last few years, various forms of constructive negation have been proposed (see for instance [2, 24, 11, 10, 15]), to deal with the problem of *floundering* in SLDNF-resolution. [11], Drabent introduces SLDFA-resolution, a proof procedure for general logic programs based on SLD-resolution and constructive negation, proves that it is sound and complete with respect to Kunen's three-valued completion semantics, and sound with respect to two-valued completion semantics.

In this section we generalize SLDFA-resolution to abductive logic programs. The main difference with the definition given in [11] is that the answers we compute are abducible formulas instead of constraints. As a result, most definitions in this section are direct copies of definitions in [11]. Only the definition of *goal* is slightly different.

The basic idea of using constructive negation in proof procedures for general logic programming is, that computed answers to general goals are *equality constraints*, i.e. first-order formulas build out of the equality predicate '='. This notion of computed answer generalizes the notion of computed answer substitutions, because a substitution can be written as a conjunction of primitive equalities. Instead of using equality constraints as computed answers, we use abducible formulas. If we only look at their definition, we see that abducible formulas are a generalization of equality formulas. However, there is a difference in the meaning of an abducible formula when it is used as a computed answer. When using an equality constraint θ as computed answers, one requires it to be *satisfiable* in *CET*, i.e. $CET \models \exists\theta$. However, when the computed answer is an abducible formula, there is no theory with respect to whom one can require it to be satisfiable. The only requirement for such a computed answer is, that it is consistent. Therefore, we require consistency instead of satisfiability. As our abducible formulas can contain equality predicates, we require our computed answers to be consistent with respect to *CET*. This consistency requirement for abducible formulas generalizes the satisfiability requirement for equality constraints, whenever a universal language is used.

Lemma 45. *Let θ be an equality constraint. Then, θ is satisfiable in $CET_{\mathscr{L}_u}$ iff $CET_{\mathscr{L}_u} \cup \{\theta\}$ is consistent.*

Proof. The lemma follows directly from the fact that $CET_{\mathcal{L}_u}$ is a complete theory. □

We will not concern ourselves with reducing abducible formulas to normal forms. We simply assume the existence of normalization procedures that transform a given abducible formula into a format that is intelligible to humans.

SLDFA-resolution is defined by two basic notions: *SLDFA-refutations* and (*finitely failed*) *SLDFA-trees*. An *SLDFA-refutation* is a sequence of goals, ending in a goal without non-abducible atoms, such that each goal in the sequence is obtained from the previous goal by a *positive* or *negative derivation step*. A *positive derivation step* is the usual one used in SLD-resolution, with the difference that the resolved atom has to be a non-abducible atom. A *negative derivation* step is the replacement of a negative non-abducible literal $\neg A$ in the goal by an abducible formula σ such that $\leftarrow \sigma, A$ is guaranteed to fail finitely. A *finitely failed SLDFA-tree* for a goal G is a proof for the fact that G fails finitely; it is an approximation that is 'save' with respect to finite failure; if a finitely failed SLDFA-tree for G exists, it is guaranteed that G fails finitely, but the fact that that there exists an SLDFA-tree for G that is not finitely failed, does not imply that G is not finitely failed.
Before we can define SLDFA-resolution, we have to define the notion of a *goal*.

Definition 46. Let P be a program. A *goal* (w.r.t. P) is a formula of the form $\neg(\theta \wedge L_1 \wedge \ldots \wedge L_k)$, usually written as $\leftarrow \theta, L_1, \ldots, L_k$, such that
- θ is a consistent abducible formula, and
- L_i (for $i \in [1..k]$) is a non-abducible literal.

An *s-goal* is a goal in which one of the literals is marked as *selected*.

We begin the definition of SLDFA-resolution with the definition of *positively derived goals*.

Definition 47. Let P be a program, let G be the s-goal $\leftarrow \theta, \underline{N}, p(\underline{t}), \underline{M}$ (with $p(\underline{t})$ selected) and let $p(\underline{s}) \leftarrow \sigma, \underline{L}$ be a variant of a clause in P. A goal G' is *positively derived* from G using $p(\underline{s}) \leftarrow \sigma, \underline{L}$ if
- $\mathrm{FreeVar}G \cap \mathrm{FreeVar}\, p(\underline{s}) \leftarrow \sigma, \underline{L} = \emptyset$ and
- G' is of the form $\leftarrow \theta, (\underline{t} = \underline{s}), \sigma, \underline{N}, \underline{L}, \underline{M}$.

If G' is positively derived from G using a variant of a clause R, we call R *applicable* to G.

Note that the abducible formula in G' is (by definition) consistent because G' is (by definition) a goal, and by definition the abducible formula in a goal is consistent.

We now give the definitions of *negatively derived goals*, *finitely failed goals*, (*finitely failed*) *SLDFA-trees*, and *SLDFA-refutations*. These definitions are mutually recursive. Therefore, we define them inductively, using the notion of *rank*.

Definition 48. Let P be a program and let G be the s-goal $\leftarrow \theta, \underline{N}, \neg A, \underline{M}$ (with $\neg A$ selected). Let the notion of *rank k finitely failed goals* be defined. A goal G' is *rank k negatively derived* from G if
- G' is of the form $\leftarrow \theta, \sigma, \underline{N}, \underline{M}$,
- $\leftarrow \theta, \sigma, A$ is a rank k finitely failed goal, and
- $FreeVar(\sigma) \subseteq FreeVar(A)$.

We call θ, σ a (*rank k*) *fail answer* for $\leftarrow \theta, A$.

Definition 49. Let P be a program and let G be a goal. Let the notion of *rank k finitely failed SLDFA-tree* be defined. G is a *rank k finitely failed goal* if there exists a rank k finitely failed SLDFA-tree for G.

Definition 50. Let P be a program and let G be a goal. Let the notion of *rank k SLDFA-refutation* be defined. A *rank k SLDFA-tree* for G is a tree such that

(i) each node of the tree is an s-goal and the goal part of the root node is G,

(ii) the tree is finite,

(iii) if $H : \leftarrow \theta, \underline{L}_1, A, \underline{L}_2$ (with A selected) is a node in the tree then, for every clause R in P applicable to H, there exists exactly one son of H that is positively derived from H using a variant of R, and

(iv) if $H : \leftarrow \theta, \underline{L}_1, \neg A, \underline{L}_2$ (with $\neg A$ selected) is a node in the tree, then it has sons

$$\leftarrow \sigma_1, \underline{L}_1, \underline{L}_2 \ , \ \ldots \ , \ \leftarrow \sigma_m, \underline{L}_1, \underline{L}_2$$

provided there exist $\delta_1, \ldots, \delta_n$ that are SLDFA-computed answers obtained by rank k SLDFA-refutations of $\leftarrow \theta, A$, such that

$$CET \models \theta \rightarrow \delta_1 \vee \cdots \vee \delta_n \vee \sigma_1 \vee \cdots \vee \sigma_m$$

If no node in an SLDFA-tree is of the form $\leftarrow \theta$, then that tree is called *finitely failed.*

Definition 51. Let P be a program and let G be a goal. Let the notion of *rank* $k-1$ *negatively derived s-goal* be defined. A *rank k SLDFA-refutation* of G is a sequence of s-goals $G_0, G_1, \ldots, G_n$ such that G is the goal part of G_0, G_n is of the form $\leftarrow \theta$ and, for $i \in [1..n]$,
- G_i is positively derived from G_{i-1} using a variant C of a clause in P such that $FreeVar(C) \cap FreeVar(G_0, \ldots, G_{i-1}) = \emptyset$, or
- G_i is rank $k-1$ negatively derived from G_{i-1}.

The abducible formula $\exists \underline{y} \theta$, where $\underline{y} = FreeVar(\theta) - FreeVar(G)$, is a *SLDFA-computed answer* for G.

To get some insight in the construction of SLDFA-refutations, let us conclude with an example

Example 52. Consider program P_{Tweety} and the *observation* $\neg flies(tweety)$. In Fig. 2 we show the SLDFA-refutation for this query in R_1. Let us see how this refutation is

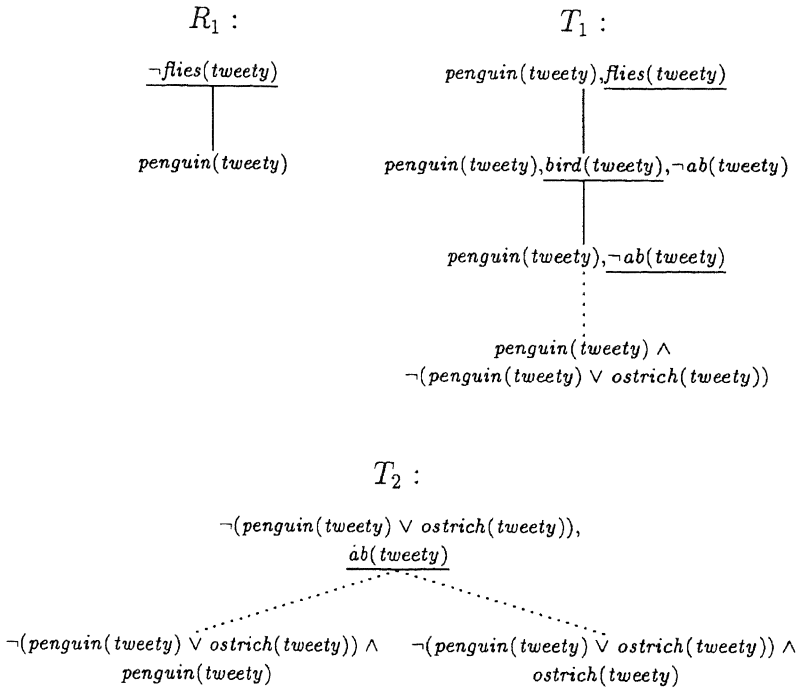

Fig. 2. An SLDFA-refutation for $\neg flies(tweety)$

constructed. First of all, consider T_2 which is a finitely failed SLDFA-tree for

$$\leftarrow \neg(penguin(tweety) \vee ostrich(tweety)), ab(tweety)$$

The two dotted lines in the tree lead to two 'goals' which are not valid resolvents, because their constraint part is inconsistent. Thus, the root of T_2 is a finitely failed goal. Secondly, we have T_1 is a finitely failed SLDFA-tree for

$$\leftarrow penguin(tweety), flies(tweety)$$

This tree is finitely failed, because applying the finite fail answer obtained by T_2 in a (negative) resolution step with $\neg ab(tweety)$ results in a 'goal' with an inconsistent constraint (see the dotted line). Because, the finitely failed goal in T_2 is most general, it follows that the third goal in T_1 has no resolvents. Thus, the root of T_1 is a finitely failed goal. This fact is used in the construction of the SLDFA-refutation R_1.

10. Soundness and completeness of generalized SLDFA-resolution

In this section we present some soundness and completeness results on SLDFA-resolution for abductive logic programs. We start by proving soundness with respect to three-valued completion semantics for abductive logic programs.

Theorem 53. *Let P be a program and let G be the goal $\leftarrow \theta, \underline{L}$.*

(i) *If δ is an SLDFA-computed answer for G then $comp(P) \models_3 \delta \rightarrow \theta \wedge \underline{L}$.*

(ii) *If G finitely fails then $comp(P) \models_3 \theta \rightarrow \neg \underline{L}$.*

The proof of this theorem closely resembles the proof of Theorem 4.2 in [11]. The differences between the two proofs are, that here we prove soundness with respect to three-valued completion semantics, while Drabent's proof proves soundness with respect to two-valued completion, and that we work with abductive formulas instead of constraints. We omitted the proof, because it is rather lengthy and technical. It can be found in [25].

The following corollary proves soundness of SLDFA-resolution with respect to the three-valued completion semantics for abductive logic programs, as stated in Definition 17.

Corollary 54 (Three-valued soundness). *Let P be a program and let G be the goal $\leftarrow \theta, \underline{L}$. If δ is an SLDFA-computed answer for G, then δ is a three-valued explanation for $\langle P, \theta \wedge \underline{L} \rangle$.*

Proof. Because δ is an SLDFA-computed answer for G, by Theorem 53, $comp(P) \models_3 \delta \rightarrow \theta \wedge \underline{L}$. Moreover, δ has a 3-valued model, which implies that $comp(P) \cup \{\delta\}$ is consistent. But then, $comp(P) \cup \{\delta\} \models_3 \theta \wedge \underline{L}$. Thus, δ is a three-valued explanation for $\langle P, \theta \wedge \underline{L} \rangle$. □

Now that we have proven soundness with respect to three-valued completion semantics, the following result is straightforward.

Theorem 55. *Let P be a program and let $G : \leftarrow \theta, \underline{L}$ be a goal.*

(i) *If δ is an SLDFA-computed answer for G then $comp(P) \models \delta \rightarrow \theta \wedge \underline{L}$.*

(ii) *If G finitely fails then $comp(P) \models \theta \rightarrow \neg \underline{L}$.*

Proof.

(i) Suppose that δ is an SLDFA-computed answer for G. Then, by Theorem 53, $comp(P) \models_3 \delta \rightarrow \theta \wedge \underline{L}$. But we know that every two-valued model for $comp(P)$ is also a three-valued model for $comp(P)$, and therefore $comp(P) \models \delta \rightarrow \theta \wedge \underline{L}$.

(ii) Suppose that G finitely fails. Then, by Theorem 53, $comp(P) \models_3 \theta \rightarrow \neg \underline{L}$. But every two-valued model for $comp(P)$ is also a three-valued model for $comp(P)$, and therefore $comp(P) \models \theta \rightarrow \neg \underline{L}$. □

Using this theorem, we can prove the following soundness result with respect to two-valued completion semantics.

Corollary 56 (Two-valued soundness). *Let P be a program and let G be the goal $\leftarrow \theta, \underline{L}$. If δ is an SLDFA-computed answer for G and $comp(P) \cup \{\delta\}$ is consistent, then δ is an explanation for $\langle P, \theta \wedge \underline{L} \rangle$.*

Proof. Because δ is an SLDFA-computed answer for G, by Theorem 55 $comp(P) \models \delta \rightarrow \theta \wedge \underline{L}$. But then, because $comp(P) \cup \{\delta\}$ is consistent, we have that $comp(P) \cup \{\delta\} \models \theta \wedge \underline{L}$. Thus, δ is an explanation for $\langle P, \theta \wedge \underline{L} \rangle$. □

We now turn prove completeness of the generalized SLDFA-resolution with respect to three-valued completion semantics.

Theorem 57. *Let P be a program and let $G : \leftarrow \theta, \underline{L}$ be a goal. Let δ be an abducible sentence. Then, for an arbitrary fair selection rule,*

(i) *if $comp(P) \cup \{\delta\} \models_3 \theta \wedge \underline{L}$, then there exist computed answers $\delta_1, \ldots, \delta_n$ for G such that $CET \models_3 \delta \rightarrow \delta_1 \vee \cdots \vee \delta_n$, and*

(ii) *if $comp(P) \models_3 \theta \rightarrow \neg \underline{L}$ then G fails finitely.*

As was the case with Theorem 53, the proof of this theorem is (almost) identical to the proof of the corresponding theorem in [11] (Theorem 5.1). The only difference is, that we use results from Section 5, where Drabent used results from [20]. We omit the proof here, because it is rather lengthy and technical. It can be found in [25].

Corollary 58 (Three-valued completeness). *Let P be a program, let G be the goal $\leftarrow \theta, \underline{L}$ and let δ be an abducible sentence. If δ is a three-valued explanation for $\langle P, \theta \wedge \underline{L} \rangle$, then there exist SLDFA-computed answers $\delta_1, \ldots, \delta_k$ for G such that $CET \models_3 \delta \rightarrow \delta_1 \vee \cdots \vee \delta_k$.*

Proof. By definition, δ is a three-valued explanation for $\langle P, \theta \wedge \underline{L} \rangle$, iff $comp(P) \cup \{\delta\} \models_3 \theta \wedge \underline{L}$. But then, by Theorem 57, there exist SLDFA-computed answers $\delta_1, \ldots, \delta_k$ for $\leftarrow \theta, \underline{L}$ such that $CET \models_3 \delta \rightarrow \delta_1 \vee \cdots \vee \delta_k$. □

11. Conclusions

In this paper we generalize Kunen semantics and Fitting semantics to the setting of abductive logic programming. This is, we think, the main contribution of this paper. We think that, as is the case with logic programming, also with abductive logic programming these semantics are of interest, especially when considering SLD-like proof procedures, as an alternative to the more informative but also computationally more expensive semantics like the argumentation semantics. Also, by providing these semantics, we underline the fact that deduction and (limited forms of) abduction are closely related.

Also, we show that it is not necessary to restrict explanations to ground formulas, as is often done when presenting semantics or proof procedures for abductive logic programs. However, by allowing variables in explanations, we have to take care with free variables in observations and explanations. In our definition of explanation, we chose to implicitly universally quantify the free variables in both observation and explanation. By doing so, we do not allow any 'communication' between observation

and explanation. As a result, we cannot handle situations where the observation and explanation both are to be seen as 'generic' in some set of free variables, i.e. where, given observation ϕ and explanation δ, both with free variables $\underline{x}$, and a substitution θ with domain $\{\underline{x}\}$, it is understood that $\delta\theta$ is an explanation for $\phi\delta$. We could define the notion of explanation differently, by having $comp(P) \models \delta \rightarrow \phi$ in its definition, instead of $comp(P) \models \delta \rightarrow \phi$. With such a definition, there would be 'communication' between free variables in δ and ϕ. Our reasons for not doing so are mostly of a technical nature, concerning the definition of the immediate consequence operator. We think that for this alternative notion of explanation, also a Kunen semantics can be established, and that the proof procedure would also be sound with respect to this alternative notion of explanation.

In the second part of this paper we present a generalization of Drabent's SLDFA-resolution, and use it as a proof procedure for abductive logic programming. We show that the proof procedure is sound with respect to two-valued completion semantics – provided the union of completed program and answer is consistent – and that it is sound and complete with respect to three-valued completion semantics. There is quite a difference between SLDFA-resolution for abductive logic programming, and Denecker and De Schreye's SLDNFA-resolution. For one thing, Denecker and De Schreye want the explanations to be ground conjunctions of atoms. For this, they skolemize non-ground goals, and use 'skolemizing substitutions' in the resolution steps. Instead, we allow our explanations to be arbitrary non-ground abducible formulas. These differences would make a close comparison between the two proof procedures a rather technical exercise. However, we are quite confident that, for any answer given by SLDNFA-resolution, there is an 'equivalent' SLDFA-computed answer. We expect this not to hold the other way around, simply because our proof procedure is based on constructive negation, while SLDNFA-resolution is based on negation as failure.

The great similarity between SLDFA-resolution and SLDNFA-resolution is that they both use deduction, and both do not concern themselves with the consistency of the obtained answers with respect to the completed program. As a result, they cannot be compared with ordinary proof procedures for abductive logic programming, whose main concern *is* consistency of the obtained answers. In this context, choice between two- and three-valued completion semantics is an important one; if we use two-valued completion semantics, in addition to SLDFA-resolution we do need a procedure to check whether the obtained SLDFA-computed answer is consistent with respect to the completed program. We think that this will mean a considerable increase in computation costs. On the other hand, if we use three-valued completion semantics, the need for this consistency check disappears. However, one can argue that this is a 'fake' solution; in some sense we just disregard inconsistencies, by weakening the notion of a model. In our opinion, the choice of semantics depends on your view on abductive logic programs, and the relation between abducible and non-abducible predicates. A second reason why it is interesting to look at proof procedures for abductive logic programming that do not check for consistency, is the case where you can guarantee that the union of computed answer and completed program is consistent. An example of this is the

translation proposed by Denecker and De Schreye in [8]. The abductive logic programs resulting from this translation are acyclic (Proposition 3.1), which implies that the union of their completion with a consistent abducible formula is consistent (a corollary of Proposition C.2 in [6]). There might be more of these examples, and it might be interesting to define classes of programs for which this property holds (among others, the above conjecture on acyclic programs should be proven).

Acknowledgements

This paper was supported by a grant from SION, a department of NWO, the Netherlands Organization for Scientific Research. I would like to thank Krzysztof Apt for his support, and Kees Doets for his help on three-valued completion semantics. Also, I am grateful to the referees, for some valuable comments. I especially would like to thank the referee who persisted in his opinion that his version of Example 18 was better than mine, until I saw his point.

References

[1] K.R. Apt and M. Bezem, Acyclic programs, *New Generation Comput.* **9** (1991) 335–363.
[2] D. Chan, Constructive negation based on the completed database, in: *Proc. Internat. Conf. on Logic Programming* MIT Press, Cambridge, 1988). 111–125.
[3] C.C. Chang and H.J. Keisler, *Model Theory*, Studies in Logic and the Foundations of Mathematics, Vol. 73 (North-Holland, Amsterdam, 1973).
[4] K.L. Clark, Negation as failure, in: H. Gallaire and G. Minker, eds., *Logic and Data Bases* (Plenum Press, Oxford, 1978). 293–322.
[5] L. Console, D.T. Dupre and P. Torasso, On the relationship between abduction and deduction, *J. Logic Comput.* **1**(1991) 661–690.
[6] M. Denecker, Knowledge representation and reasoning in incomplete logic programming, Ph.D Thesis, Katholieke Universiteit Leuven, Leuven, Belgium, September 1993.
[7] M. Denecker and D. De Schreye, SLDNFA: an abductive procedure for normal abductive programs, in: *Proc. Joint Internat. Conf. and Symp. on Logic Programming* (1992) 686–700.
[8] M. Denecker and D. De Schreye, Representing incomplete knowledge in abductive logic programming, in: *Proc. Internat. Logic Programming Symp.* (1993).
[9] K. Doets, *From Logic to Logic Programming*, The MIT Press series in Foundations of Computing (MIT Press, 1993).
[10] W. Drabent, SLS-resolution without floundering, in: *Proc. Workshop on Logic Programming and Non-Monotonic Reasoning* (1993).
[11] W. Drabent, What is failure? an approach to constructive negation. Updated version of a Tech. Report LITH-IDA-R-91-23 at Linkoping University, 1993.
[12] P.M. Dung, Negation as hypotheses: an abductive foundation for logic programming, in: *Proc. Internat. Conf. on Logic Programming* (1991) 3–17.
[13] K. Eshghi, Abductive planning with the event calculus, in: K.A. Bowen and R.A. Kowalski, eds., *Proc. Internat. Conf. on Logic Programming* (1988) 562–579.
[14] K. Eshghi and R.A. Kowalski, Abduction compares with negation by failure, in: G. Levi and M. Martelli, eds., *Proc. Internat. Conf. on Logic Programming* (1989) 234–254.
[15] F. Fages, Constructive negation by pruning, Working Paper, May 1994.
[16] M. Fitting, A Kripke–Kleene semantics for logic programs, *J. Logic Programming* **2** (1985) 295–312.
[17] A.C. Kakas, R.A. Kowalski and F. Toni, Abductive logic programming. *J. Logic Comput.* **2** (1993) 719–770.

[18] A.C. Kakas and P. Mancarella, Generalized stable models: a semantics for abduction, in: *Proc. 9th European Conf. on Artificial Intelligence*, Stockholm (1990) 385–391.

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

[20] K. Kunen, Negation in logic programming, *J. Logic Programming* **4** (1987) 289–308.

[21] J.W. Lloyd, *Foundations of Logic Programming*, Symbolic Computation – Artificial Intelligence (Springer, Berlin, 2nd extended edn., 1987).

[22] K. Satoh and N. Iwayama, A query evaluation method for abductive logic programming, in: K.R. Apt, ed., *Proc. J. Internat. Conf. and Symp. on Logic Programming* (1992) 671–685.

[23] J.C. Shepherdson, Language and equality theory in logic programming. Tech. Report PM-88-08, School of Mathematics, University Walk, Bristol, BS8 1 TW, England, 1988.

[24] P.J. Stuckey, Constructive negation for constraint logic programming, in: *Proc. IEEE Symp. on Logic in Computer Science* (IEEE Computer Society Press, New York, July 1991) 328–339.

[25] F.J.M. Teusink, Three-valued completion for abductive logic programs. Tech. Report CS-R9474, Center for Mathematics and Computer Science, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands, December 1994.