Three-Valued Completion for Abductive Logic Programs

F.J.M. Teusink

Computer Science/Department of Algorithmics and Architecture

CS-R9474 1994

Three-Valued Completion for Abductive Logic Programs

Frank Teusink

CWI
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
email: frankt@cwi.nl

Abstract

There is a growing interest in ways to represent incomplete information in logic programs. It has been shown that limited forms of abduction can be used quite elegantly for this purpose. In this paper, we propose a three-valued completion semantics for abductive logic programs, which solves some problems associated with Console et al's 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.

AMS Subject Classification (1991): 68N17.

CR Subject Classification (1991): D.1.6, F.4.1, I.2.3.

Keywords & Phrases: Logic Programming, Abduction, Completion semantics.

Note: This report is an extended version of [Teu94].

1. Introduction

In [DS93], Denecker and De Schreye propose to use abduction as a means to represent incomplete information in logic programs, and present a translation from $\mathcal{A}$ (a language for stating problems in event calculus, proposed by M. Gelfond and V. Lifschitz in [GL92]) to abductive logic programs (or *incomplete logic programs*, as they call them). As a proof procedure, they propose SLDNFA-resolution (see [DS92]); 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 [CDT91].

In the last few years, various forms of constructive negation have been proposed (see for instance [Cha88, Stu91, Dra93b, Dra93a, Fag94]), to deal with the problem of *floundering* in SLDNF-resolution. In [Dra93b], W. 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 paper, we generalize SLDFA-resolution and use it as a proof procedure for abductive logic programs. The proposed 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.

Moreover, by generalizing a proof procedure from general logic programming in a straightforward way to abductive logic programming, we show that adding (limited forms of) abduction to logic programs is not too involving. We prove that this proof procedure is (under some restrictions) sound with respect to the two-valued completion semantics of Console et al.

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 [Fit85], M. 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 [Kun87], K. 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. It is this second semantics with respect to whom Drabent proved his proof procedure sound and complete.

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 J.C. Shepherdson in [She88], for abductive logic programs. Finally, we prove soundness and completeness of the generalized SLDFA-resolution with respect to Kunen semantics. Again, in generalizing these notions to abductive logic programs, we intend to show that general logic programs can be extended quite naturally to incorporate (some limited forms of) abduction.

The class of abductive logic programs on which we concentrate in this paper, is (almost) the same as the class of 'incomplete logic programs' defined by Denecker and De Schreye. They can be seen as a generalization of ordinary general logic programs, in the sense that they are treated as general logic programs in all but the abducible predicates. The abducible predicates can be seen as 'placeholders' for representing incomplete information; the answer of a query (or the explanation of an observation) is an expression in terms (predicates) of concepts that you know exist, but on which you have no knowledge that enable you to reason with them. The proof procedure we present will reflect this view on this class of programs, by reasoning with the non-abducible predicates as if they were part of a general logic program, while the abducible predicates just 'hang around'.

The paper is organized in four 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, we start with a generalization of SLDFA-resolution to the case of abductive logic programs (Section 5), followed by an example of its use in Section 6. In the third part, which starts with Section 7, we present the immediate consequence operator (Section 8), and use it to characterize Fitting semantics (Section 9) and Kunen semantics (Section 10) for abductive logic programs. Finally, in Sections 11 and 12, we present some soundness and completeness results on SLDFA-resolution.

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 $L_1 \wedge \ldots \wedge L_n$. 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 $(s_1 = t_1) \wedge \ldots \wedge (s_k = t_k)$.

In the remainder of this section, we introduce some basic notions concearning algebras and models. To begin with, an *algebra* (or *pre-interpretation*, as it is called in [Llo87]), is the part of a model that interprets the terms of the language.

Definition 2.1 Let $\mathcal{L}$ be a language and let $\mathcal{F}$ be the set of function symbols in $\mathcal{L}$. An $\mathcal{L}$-*algebra* is a complex $J = \langle D, \mathbf{f}, \ldots \rangle_{f \in \mathcal{F}}$ where D is a non-empty set, the *domain* (or *universe*) of J, and for every n-ary function symbol $f \in \mathcal{F}$, $\mathbf{f}$ is an n-ary function $\mathbf{f} : D^n \rightarrow D$. □

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

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

Definition 2.2 Let $\mathcal{L}$ be a language. Let $\mathcal{F}$ be the set of function symbols in $\mathcal{L}$ and let $\mathcal{R}$ be the set of predicate symbols in $\mathcal{L}$. A *two-valued* $\mathcal{L}$-*model* is a complex $M = \langle D, \mathbf{f}, \ldots \mathbf{r}, \ldots \rangle_{f \in \mathcal{F}, r \in \mathcal{R}}$ where $\langle D, \mathbf{f}, \ldots \rangle_{f \in \mathcal{F}}$ is an $\mathcal{L}$-algebra, for every n-ary predicate symbol $r \in \mathcal{R}$, $\mathbf{r}$ is a subset of D^n, and equality (if present) is interpreted as identity. □

Definition 2.3 Let $\mathcal{L}$ be a language. Let $\mathcal{F}$ be the set of function symbols in $\mathcal{L}$ and let $\mathcal{R}$ be the set of predicate symbols in $\mathcal{L}$. A *three-valued* $\mathcal{L}$-*model* is a complex $M = \langle D, \mathbf{f}, \ldots \mathbf{r}, \ldots \rangle_{f \in \mathcal{F}, r \in \mathcal{R}}$ where $\langle D, \mathbf{f}, \ldots \rangle_{f \in \mathcal{F}}$ is an $\mathcal{L}$-algebra, for every n-ary predicate symbol $r \in \mathcal{R}$, $\mathbf{r}$ is an n-ary function $\mathbf{r} : D^n \rightarrow \{\mathbf{t}, \mathbf{f}, \bot\}$, and equality (if present) is interpreted as two-valued identity. □

Following [Doe93], 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) $f(x_1, \ldots, x_n) = f(y_1, \ldots, y_n) \rightarrow (x_1 = y_1) \ \wedge \ \ldots \ \wedge \ (x_n = y_n)$ ($\forall$ 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, $FreeVar(\phi)$ denotes the set of free variables in ϕ. A *sentence* is a closed formula (i.e. $FreeVar(\phi)$ 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 $\mathcal{L}$ 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 $\mathcal{L}$ and a domain D, the D-*language* $\mathcal{L}_D$ is obtained by extending $\mathcal{L}$ with a fresh constant for every domain element in D. When working in some language $\mathcal{L}$ and referring to D-sentences or D-formulas, we intend sentences or formulas in the language $\mathcal{L}_D$. We can extend an $\mathcal{L}$-algebra J to an $\mathcal{L}_D$-algebra J_D by interpreting each new constant in $\mathcal{L}_D$ '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 $\mathcal{L}$ and a formula ϕ, a *D-ground* instance of ϕ is a ground instance of ϕ in the language $\mathcal{L}_D$. Given an algebra J with domain D, we sometimes refer to D-ground formulas as J-ground formulas.

Lemma 2.4 *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 $\mathcal{L}_{\mathcal{U}}$, which has a (countably) infinite number of constant and function symbols of any arity. 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 $\langle \mathcal{A}_P, \mathcal{R}_P, \mathcal{I}_P\rangle$, where

- $\mathcal{A}_P$ is a set of *abducible predicates*,
- $\mathcal{R}_P$ 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
- $\mathcal{I}_P$ is a finite set of first-order integrity constraints.

An *abducible formula* (with respect to 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.

In the remainder of this paper, no integrity constraints are used, i.e. $\mathcal{I}_P$ will always be empty. We can make this restriction, because there exist techniques to translate integrity constrains to some set $\mathcal{IR}_P$ of program rules with head *False* (this is 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 $\langle P', \phi \wedge \neg \mathit{False}\rangle$, where P' is the program $\langle \mathcal{A}_P, \mathcal{R}_P \cup \mathcal{IR}_P, \emptyset\rangle$. We use this technique in the example of Section 6.

If we compare our definition of abductive logic programs with the definitions given by Console et al, and by Denecker and DeSchreye, 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.

4. Completion semantics for abductive logic programs

In [Cla78], K. L. 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 4.1 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 $i \in [1..m]$, $y_i = \mathit{FreeVar}(\theta_i, \underline{L}_i) - \mathit{FreeVar}(p(\underline{s}_i))$. The *completed definition of p (with respect to 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 [CDT91], 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 4.2 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. □

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 4.3 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 [CDT91]). 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 4.4 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).

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

$\leftrightarrow$	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**

Figure 0.1: Kleene equivalence and strong equivalence

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 [AB91] 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 4.5 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. □

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

Lemma 4.6 *Let $\langle P, \phi \rangle$ be an abductive problem, 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 that $comp(P) \models \phi \rightarrow \delta$. 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 Section 7, 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 Kunen 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 figure 0.1). The operator $\cong$ cannot be constructed using Kleene's operators, and therefore has to be introduced separately. Its use will 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.

Definition 4.7 Let $\langle P, \phi\rangle$ be an abductive problem. An abducible formula δ is a *three-valued explanation for $\langle P, \phi\rangle$ (in Fitting semantics)*, if ϕ is true in all Herbrand models of $comp(P) \cup \{\delta\}$. δ is consistent. □

Definition 4.8 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.

Example 4.9 *Let P be the program:*

$$\left\langle \begin{array}{l} number(0) \leftarrow a \\ number(s(x)) \leftarrow number(x), a \end{array}, \{a\}, \emptyset \right\rangle$$

It's completion is:

$$number(x) \cong (x = 0 \ \wedge \ a) \ \vee \ \exists_y (y = s(x) \ \wedge \ number(y) \ \wedge \ a)$$

Let ϕ be the formula $\forall_x number(x)$. Now, consider the abductive problem $\langle P, \phi\rangle$. In Fitting semantics (over the language $\mathcal{L}_P$, a is an explanation for this problem. The reason is, that in Herbrand models, domain elements are isomorphic to terms of the language, and therefore, in this example, isomorphic to a term of the form $s^i(0)$ where $i \geq 0$. On the other hand, if we allow arbitrary three-valued models, we can choose richer models. For instance, consider the model M with domain $\mathbb{N} \cup \{\omega\}$, in which a term $s^i(0)$ is mapped onto i, in which a is true, and in which $number(n)$ is true for all natural numbers n in the domain, but in which $number(\omega)$ is false. Clearly, M is a model of $comp(P) \cup \{a\}$. However, ϕ is not true in M, and therefore a 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 4.10 *Consider the abductive logic program P, with a single abducible predicate a, and the following two clauses:*

$$\begin{array}{l} p \leftarrow \neg p, a \\ q \leftarrow a \end{array}$$

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 $\bot$ to p, we can construct three-valued models of $comp(P) \cup \{a\}$, and therefore a is *a three-valued explanation for $\langle P, q\rangle$.* ○

5. Generalizing SLDFA-resolution

In this section we generalize SLDFA-resolution, as defined by W. Drabent in [Dra93b], to abductive logic programs. The main difference with the definition given in [Dra93b] 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 [Dra93b]. 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 5.1 *Let θ be an equality constraint. Then, θ is satisfiable in $CET_{\mathcal{L}_\mathcal{U}}$ iff $CET_{\mathcal{L}_\mathcal{U}} \cup \{\theta\}$ is consistent.*

Proof: The lemma follows directly from the fact that $CET_{\mathcal{L}_\mathcal{U}}$ 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 5.2 Let P be a program. A *goal* (with respect to P) is a formula $\neg(\theta \wedge L_1 \wedge \ldots \wedge L_k)$, 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 5.3 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

- $\mathit{FreeVar}(G) \cap \mathit{FreeVar}(p(\underline{s}) \leftarrow \sigma, \underline{L}) = \emptyset$ 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 5.4 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
- $\mathit{FreeVar}(\sigma) \subseteq \mathit{FreeVar}(A)$.

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

Definition 5.5 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 5.6 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

1. each node of the tree is an s-goal and the goal part of the root node is G,
2. the tree is finite,
3. 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
4. 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

   $$\mathit{CET} \models \theta \rightarrow \delta_1 \ \vee \ \ldots \ \vee \ \delta_n \ \vee \ \sigma_1 \ \vee \ \ldots \ \vee \ \sigma_m$$

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

Definition 5.7 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 $\mathit{FreeVar}(C) \cap \mathit{FreeVar}(G_0, \ldots, G_{i-1}) = \emptyset$, or
- G_i is rank $k-1$ negatively derived from G_{i-1}.

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

6. An example: the Murder Mystery Domain

In [DS93], M. Denecker and D. de Schreye present a translation of the so-called Murder Mystery Domain into an abductive logic program. The Murder Mystery domain is described by the following action system:

Init *Alive*.
$\neg$*Alive* **after** *Shoot*; *Wait*.
Load **causes** *Loaded*.
Shoot **causes** $\neg$*Alive* **if** *Loaded*.
Shoot **causes** $\neg$*Loaded*.

This domain is translated into the following abductive logic program P_{MMD} (we use lists here, instead of *Result/2* terms):

$\mathit{Holds}(f, []) \leftarrow \mathit{Init}(f).$
$\mathit{Holds}(f, [a|s]) \leftarrow \mathit{Holds}(f, s), \neg \mathit{NonInert}(f, a, s).$
$\mathit{Holds}(\mathit{Loaded}, [\mathit{Load}|s]).$
$\mathit{NonInert}(\mathit{Loaded}, \mathit{Load}, s).$
$\mathit{NonInert}(\mathit{Loaded}, \mathit{Shoot}, s).$
$\mathit{NonInert}(\mathit{Alive}, \mathit{Shoot}, s) \leftarrow \mathit{Holds}(\mathit{Loaded}, s).$
$\mathit{False} \leftarrow \neg \mathit{Holds}(\mathit{Alive}, []).$
$\mathit{False} \leftarrow \mathit{Holds}(\mathit{Alive}, [\mathit{Wait}, \mathit{Shoot}]).$

In this program, *Init*/1, which models the initial situation, is the only abducible predicate. The predicate *NonInert*/3 describes which actions under which situations can influence which fluents. Then, the *Holds*/2 predicate uses *Init*/1 and *NonInert*/3 to describe which fluents hold in which situations. Note, that the first two clauses for *Holds*/2 are standard; the first one uses *Init*/1 to state that there is incomplete information on the initial situation, while the second one defines the law of inertia (whenever a fluent is inert, it doesn't change state). Finally, the clauses with head *False* implement the integrity constraints. They are used to model the *v-propositions* (i.e. the **init** and **after** clauses). Note, that in order to enforce the integrity constraints, the conjunct $\neg$*False* should be added to any goal we would like to answer.

First, consider the abductive logic program P_{MMD} and the goal $\leftarrow \neg \mathit{False}$. To find an answer for this goal, we need to construct three SLDFA-refutations and one SLDFA-tree. They are represented in figure 0.2. In this figure, a label $\mathit{fail}(T_i)$ is used in negative derivation steps to indicate that one can construct a finitely failed SLDFA-tree from the SLDFA-tree T_i, by adding the constraint in the

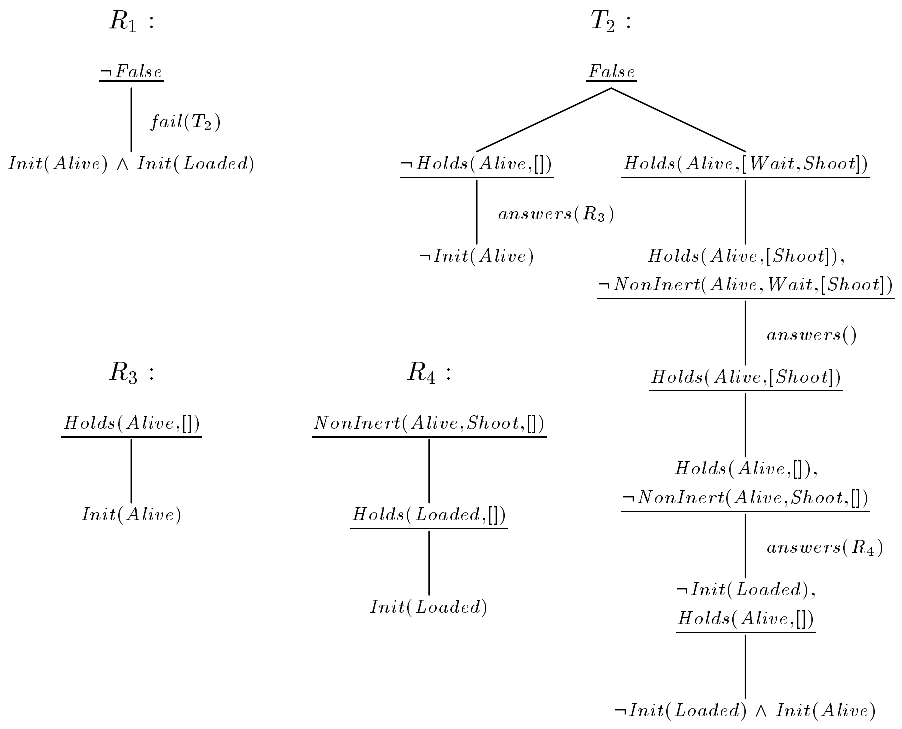

Figure 0.2: Refutations and trees for answering $\leftarrow \neg \mathit{False}$.

derivant to every goal in T_i. A label $\mathit{answers}(R_1, \ldots, R_n)$ indicates the SLDFA-refutations used in the constructing of the sons of a node in an SLDFA-tree with negative literal selected. From figure 0.2 (in particular SLDFA-refutation R_1) it follows that

$$\mathit{Init}(\mathit{Alive}) \ \wedge \ \mathit{Init}(\mathit{Loaded})$$

is an SLDFA-computed answer for $\leftarrow \neg \mathit{False}$.

Now, consider the goal $\leftarrow \neg \mathit{Holds}(\mathit{Alive}, [x_1, x_0]) \ \wedge \ \neg \mathit{False}$. In Figure 0.3 we have a SLDFA-refutation for this goal, together with some subsidiary refutations and trees (refutation R_5 refers to SLDFA-tree T_2 in figure 0.2). The SLDFA-computed answer for this goal is

$$x_1 = \mathit{Shoot} \ \wedge \ x_0 \neq \mathit{Shoot} \ \wedge \ \mathit{Init}(\mathit{Loaded}) \ \wedge \ \mathit{Init}(\mathit{Alive})$$

Note, that in this refutation we both use abducible predicates (in this case only $\mathit{Init}/1$) and constructive negation.

In Section 11, we prove that it follows that this abducible formula is a three-valued explanation for

$$\langle P_{MMD}, \neg \mathit{Holds}(\mathit{Alive}, [x_1, x_0]) \ \wedge \ \neg \mathit{False} \rangle$$

7. Three-valued completion semantics

In definition 4.8 (Section 4), we generalize Kunen semantics to abductive logic programs. The definition as given there is, however, very succinct. For one thing, it doesn't express the intention behind

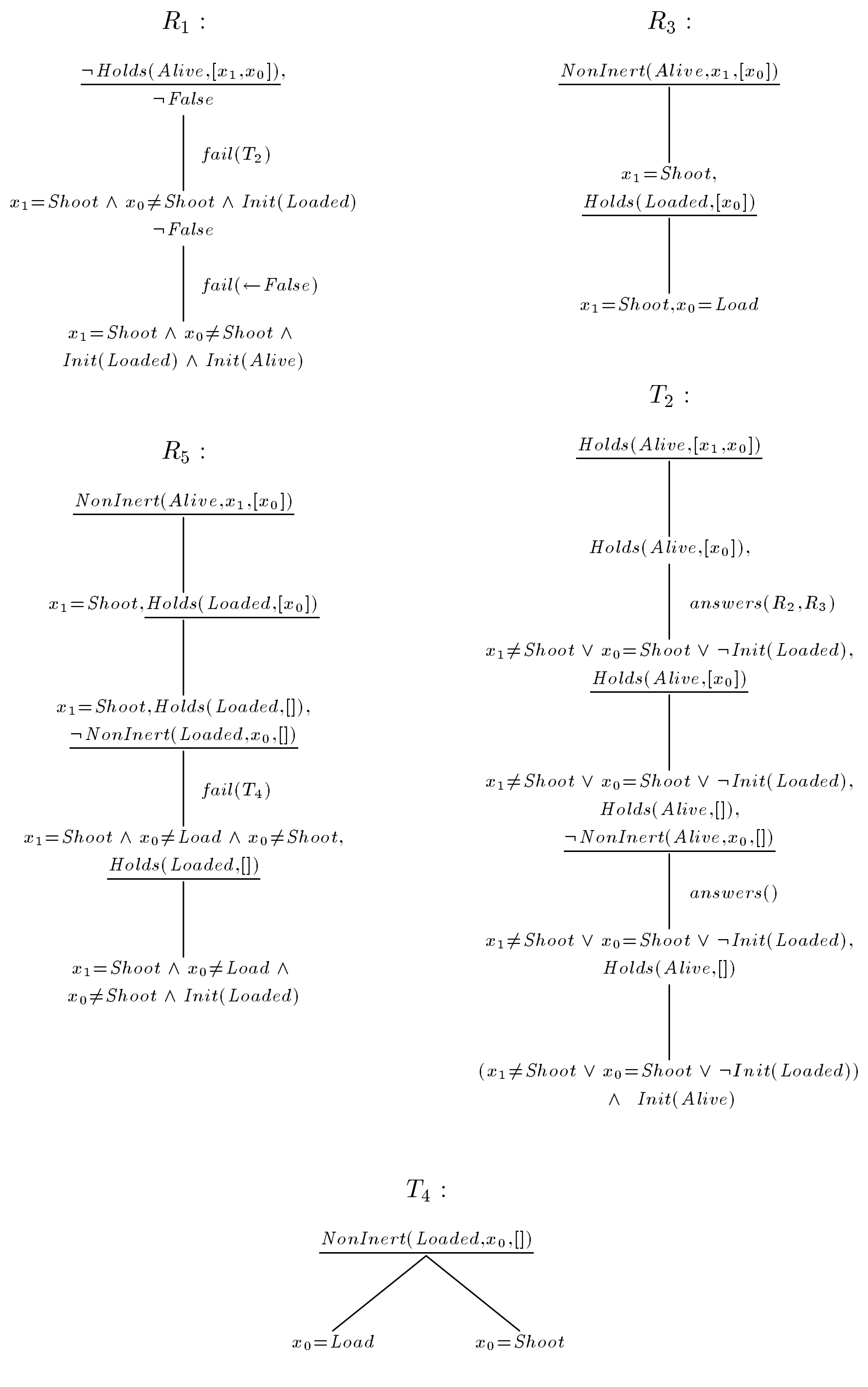
R_1 :
$\underline{\neg Holds(Alive,[x_1,x_0])}$, $\neg False$
$fail(T_2)$
$x_1{=}Shoot \wedge x_0{\neq}Shoot \wedge Init(Loaded)$ $\underline{\neg False}$
$fail(\leftarrow False)$
$x_1{=}Shoot \wedge x_0{\neq}Shoot \wedge Init(Loaded) \wedge Init(Alive)$
R_3 :
$\underline{NonInert(Alive,x_1,[x_0])}$
$x_1{=}Shoot$, $\underline{Holds(Loaded,[x_0])}$
$x_1{=}Shoot, x_0{=}Load$
R_5 :
$\underline{NonInert(Alive,x_1,[x_0])}$
$x_1{=}Shoot, \underline{Holds(Loaded,[x_0])}$
$x_1{=}Shoot, Holds(Loaded,[])$, $\underline{\neg NonInert(Loaded,x_0,[])}$
$fail(T_4)$
$x_1{=}Shoot \wedge x_0{\neq}Load \wedge x_0{\neq}Shoot$, $\underline{Holds(Loaded,[])}$
$x_1{=}Shoot \wedge x_0{\neq}Load \wedge x_0{\neq}Shoot \wedge Init(Loaded)$
T_2 :
$\underline{Holds(Alive,[x_1,x_0])}$
$\underline{Holds(Alive,[x_0])}$,
$answers(R_2,R_3)$
$x_1{\neq}Shoot \vee x_0{=}Shoot \vee \neg Init(Loaded)$, $\underline{Holds(Alive,[x_0])}$
$x_1{\neq}Shoot \vee x_0{=}Shoot \vee \neg Init(Loaded)$, $Holds(Alive,[])$, $\underline{\neg NonInert(Alive,x_0,[])}$
$answers()$
$x_1{\neq}Shoot \vee x_0{=}Shoot \vee \neg Init(Loaded)$, $Holds(Alive,[])$
$(x_1{\neq}Shoot \vee x_0{=}Shoot \vee \neg Init(Loaded)) \wedge Init(Alive)$
T_4 :
$\underline{NonInert(Loaded,x_0,[])}$
$x_0{=}Load$
$x_0{=}Shoot$

Figure 0.3: Answering $\langle P_{MMD}, \neg Holds(Alive,[x_1,x_0]) \wedge \neg False\rangle$.

both Fitting and Kunen semantics. That is, that the third truth-value stands for something like 'truth-value not determined'.

In [Fit85], M. Fitting proposes the use of three-valued semantics for general logic programs, using the third truth-value ($\bot$) 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 [Kun87], K. 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 [She88]).

8. 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 $\mathbf{t}$). 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 [Fit85] and [Kun87], Φ_P operates on Herbrand models. We however follow K.Doets [Doe93], and define the operators on J-models, given an algebra J.

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 $\Phi_{P,\Delta}$, 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 8.1 Let P be a program. A model M is an *abducible model* (with respect to P), if all non-abducible atoms in P are mapped to $\bot$ in M. □

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 $\mathbf{t}$ (resp. $\mathbf{f}$) in $\Phi_{P,\Delta}(M)$ iff it is $\mathbf{t}$ (resp. $\mathbf{f}$) in Δ.

Definition 8.2 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 $\Delta \models_3 A \vee$
  $\exists A \leftarrow \theta, \underline{L} \in J\text{-}ground(\mathrm{P}) : \Delta \models_3 \theta \ \wedge \ M \models_3 \underline{L}$

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

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.

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 8.3 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 [She88], J.C. 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 8.4 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

$$T_n(\phi) \overset{def}{=} \phi \qquad F_n(\phi) \overset{def}{=} \neg\phi$$

- 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 $FreeVars(\psi) = \underline{x}$. We define

$$T_0(\phi) \overset{def}{=} \mathbf{f} \qquad F_0(\phi) \overset{def}{=} \mathbf{f}$$

  and

$$T_n(\phi) \overset{def}{=} T_{n-1}(\underline{x} = \underline{s} \ \wedge \ \psi) \qquad F_n(\phi) \overset{def}{=} F_{n-1}(\underline{x} = \underline{s} \ \wedge \ \psi)$$

- If ϕ is a complex formula, we define

$$\begin{array}{rcl} T_n(\neg\phi) \overset{def}{=} F_n(\phi) & \qquad & F_n(\neg\phi) \overset{def}{=} T_n(\phi) \\ T_n(\phi \wedge \psi) \overset{def}{=} T_n(\phi) \wedge T_n(\psi) & & F_n(\phi \wedge \psi) \overset{def}{=} F_n(\phi) \vee F_n(\psi) \\ T_n(\phi \vee \psi) \overset{def}{=} T_n(\phi) \vee T_n(\psi) & & F_n(\phi \vee \psi) \overset{def}{=} F_n(\phi) \wedge F_n(\psi) \\ T_n(\phi \rightarrow \psi) \overset{def}{=} F_n(\phi) \vee T_n(\psi) & & F_n(\phi \rightarrow \psi) \overset{def}{=} T_n(\phi) \wedge F_n(\psi) \\ T_n(\forall\underline{x}\phi) \overset{def}{=} \forall\underline{x}T_n(\phi) & & F_n(\forall\underline{x}\phi) \overset{def}{=} \exists\underline{x}F_n(\phi) \\ T_n(\exists\underline{x}\phi) \overset{def}{=} \exists\underline{x}T_n(\phi) & & F_n(\exists\underline{x}\phi) \overset{def}{=} \forall\underline{x}F_n(\phi) \end{array}$$

□

The following lemma is a generalization of Lemma 4.1 in [She88] to abductive logic programs.

Lemma 8.5 *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,*

1. $\Phi^J_{P,\Delta} \uparrow n \models_3 \phi$ *iff* $\Delta \models_3 T_n(\phi)$
2. $\Phi^J_{P,\Delta} \uparrow n \models_3 \neg\phi$ *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 $\perp$ 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,

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

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 8.6 *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,*

1. $\Phi^J_{P,\delta} \uparrow n \models_3 \phi$ *iff* $J \cup \{\delta\} \models_3 T_n(\phi)$
2. $\Phi^J_{P,\delta} \uparrow n \models_3 \neg\phi$ *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 $\{\delta\}$. □

9. 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 9.1 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 $\mathcal{M} \models_3 \phi$. $\square$

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 9.2 *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,

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

and

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

$\square$

Corollary 9.3 *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 9.4 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. $\square$

Lemma 9.5 *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

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

and

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

□

Theorem 9.6 *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 9.2: 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 9.5, 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 $comp(P) \cup \{\delta\}$. □

10. Kunen semantics for abductive logic programs

In this section, we propose a Kunen semantics for abductive logic programs. In [Kun87], 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 10.1 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 7. The remainder of this section is dedicated to proving that these two definitions give rise to the same semantics (Theorem 10.10). In his proof of Theorem 6.3 in [Kun87], Kunen makes heavy use of ultra-products. We base our proofs on an alternative proof given by K. Doets in [Doe93].

The larger part of the work is done in the proof of Theorem 10.2, 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 10.10 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 10.2 *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 [Doe93]. It is organized as follows. In Lemma 10.3 we show that we can replace J with an elementary extension of J. Then, in Lemma 10.7 we show that for certain elementary extensions J of HA, $\Phi^{J}_{P,\Delta}$ is *continuous*. In Lemma 10.8 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 10.5 and 10.6 stated below (see [CK73]), we know these desired elementary extensions of HA exist, we can prove Theorem 10.2.

Lemma 10.3 *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 8.5, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$ 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 8.5, $\Delta' \models_3 T_n(\phi)$ iff $\Phi^{J}_{P,\Delta'} \uparrow n \models_3 \phi$. □

For Lemmas 10.7 and 10.8, we need the following definitions and results from model theory, concerning recursively saturated models.

Definition 10.4 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 10.5 *Every countable model has a countable recursively saturated elementary extension*

Property 10.6 *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,*

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

Lemma 10.7 *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 $\mathbf{t}$ (resp. $\mathbf{f}$) in $\Phi^J_{P,\Delta} \uparrow \omega$, then, for some natural number n, ϕ is $\mathbf{t}$ (resp. $\mathbf{f}$) 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 $\mathbf{t}$ in $\Phi^J_{P,\Delta} \uparrow \omega$. Then, for all $a \in A$, $\psi(a)$ is $\mathbf{t}$ in $\Phi^J_{P,\Delta} \uparrow \omega$. By induction hypothesis, for all $a \in A$, there exists an n such that $\psi(a)$ is $\mathbf{t}$ in $\Phi^J_{P,\Delta} \uparrow n$. But then, by Lemma 8.5, for all $a \in A$, there exists an n such that $T_n(\psi)(a)$ is $\mathbf{t}$ in Δ. Because J is recursively saturated, by Lemma 10.6 there exists an n such that for all $a \in A$ $T_n(\psi)(a)$ is $\mathbf{t}$ in Δ. But then, $T_n(\forall y\psi)$ is $\mathbf{t}$ in Δ and therefore by Lemma 8.5, $\forall y\psi$ is $\mathbf{t}$ in $\Phi^J_{P,\Delta} \uparrow n$.

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

Lemma 10.8 *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. $\mathbf{f}$) 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 $\mathbf{t}$ 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 10.7, there exists a natural number n such that $\Phi^J_{P,\Delta} \uparrow n \models_3 \underline{L}$, and therefore $p(\underline{s})$ is $\mathbf{t}$ in $\Phi^J_{P,\Delta} \uparrow n + 1$. Thus, $p(\underline{s})$ is $\mathbf{t}$ in $\Phi^J_{P,\Delta} \uparrow \omega$.
- Suppose $p(\underline{s})$ is $\mathbf{f}$ 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 10.7 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 $\mathbf{f}$ in $\Phi^J_{P,\Delta} \uparrow n + 1$ and therefore, $p(\underline{s})$ is $\mathbf{f}$ in $\Phi^J_{P,\Delta} \uparrow \omega$. □

Before proving Theorem 10.2, we combine the preceding two lemmas in the following corollary.

Corollary 10.9 *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 Lemma 9.2 and Lemma 10.8, $\Phi^J_{P,\Delta} \uparrow \omega$ is a three-valued model of $comp(P) \cup \{\delta\}$, and therefore $\Phi^J_{P,\Delta} \uparrow \omega \models_3 \phi$. Therefore, by Lemma 10.7 there exists a finite n such that $\Phi^J_{P,\Delta} \uparrow n \models_3 \phi$. □

Proof: (of Theorem 10.2)
Suppose that $comp(P) \cup \{\delta\} \models_3 \phi$. By property 10.5, 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 10.5, there exists an elementary J-extension Δ' of Δ. By Corollary 10.9 there exists a finite n such that $\Phi^J_{P,\Delta'} \uparrow n \models_3 \phi$. Finally, by Lemma 10.3, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$. □

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 10.10 *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 10.11 *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 10.12 *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 10.9, 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 10.11 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 10.10)

($\Rightarrow$) Suppose that $comp(P) \cup \{\delta\} \models_3 \phi$. By property 10.5, there exists a recursively saturated elementary extension J of HA. Because J is an extension of HA, it is a CET-algebra. By Lemma 10.12 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 10.5, 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 10.3, $\Phi^{HA}_{P,\Delta} \uparrow n \models_3 \phi$. 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

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

and

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

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

11. Soundness of generalized SLDFA-resolution

In this section we present some soundness 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 11.1 *Let P be a program and let G be the goal $\leftarrow \theta, \underline{L}$.*

1. *If δ is an SLDFA-computed answer for G then $comp(P) \models_3 \delta \to \theta \wedge \underline{L}$.*
2. *If G finitely fails then $comp(P) \models_3 \theta \to \neg\underline{L}$.*

The proof of this theorem closely resembles the proof of Theorem 4.2 in in [Dra93b]. 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. Before giving the proof of the theorem, we first prove the following lemma.

Lemma 11.2 *Let P be a program and let G be a goal with a positive literal selected. Let $G'_1, \ldots, G'_n$ ($n \geq 0$) be the set of all goals positively derived from G in P. Then,*

$$comp(P) \models_3 G \cong G'_1 \wedge \ldots \wedge G'_n$$

Proof: Let G be of the form $\leftarrow \theta, L_1, \ldots, L_k$. Let us assume, without loss of generality, that the left-most literal (i.e. L_1) is selected. Let L_1 be the atom $p(\underline{s})$ and let $\underline{L'}$ denote the sequence $L_2, \ldots, L_k$. Let

$$R^1 : p(\underline{t}^1) \leftarrow \sigma^1, \underline{M}^1 \quad \ldots \quad R^m : p(\underline{t}^m) \leftarrow \sigma^m, \underline{M}^m$$

contain a variant for each clause in P with head p. Assume that these clauses are standardized apart from each other and from $\underline{L}$. Finally, let, for $i \in [1..m]$, $\underline{y}^i = \mathit{FreeVar}(\sigma^i, \underline{M}^i) - \mathit{FreeVar}(p(\underline{t}^i))$.

By definition of $comp(P)$, we have that

$$comp(P) \models_3 \; p(\underline{x}) \;\cong \bigvee_{i\in[1..m]} \exists_{\underline{y}^i}((\underline{x} = \underline{t}^i), \sigma^i, \underline{M}^i)$$

(where the $\underline{x}$ do not appear in $R_1, \ldots, R_m, \theta, \underline{L}$). But then, we also have that

$$comp(P) \models_3 \; \theta, \underline{L} \;\cong \bigvee_{i\in[1..m]} \exists_{\underline{y}^i}(\theta, (\underline{s} = \underline{t}^i), \sigma^i, \underline{M}^i, \underline{L'})$$

and therefore

$$comp(P) \models_3 \neg(\theta, \underline{L}) \cong \neg\left(\bigvee_{i\in[1..m]} \exists_{\underline{y}^i}(\theta, (\underline{s} = \underline{t}^i), \sigma^i, \underline{M}^i, \underline{L'})\right)$$

which is equivalent to

$$comp(P) \models_3 \neg(\theta, \underline{L}) \cong \bigwedge_{i\in[1..m]} \forall_{\underline{y}^i} \neg(\theta, (\underline{s} = \underline{t}^i), \sigma^i, \underline{M}^i, \underline{L'})$$

Because all clauses where standardized apart, we can remove the universal quantifier:

$$comp(P) \models_3 \neg(\theta, \underline{L}) \cong \bigwedge_{i\in[1..m]} \neg(\theta, (\underline{s} = \underline{t}^i), \sigma^i, \underline{M}^i, \underline{L'})$$

Now, by definition of positively derived goal, we have for all $j \in [1..n]$ that G'_j is a variant of $\neg(\theta, (\underline{s} = \underline{t}^i), \sigma^i, \underline{M}^i, \underline{L'})$, for some $i \in [1..m]$. Moreover, for each R^i for which there is not a goal G'_j such that G'_j is derived from G using a variant of R^i, $\theta, (\underline{s} = \underline{t}^i), \sigma^i$ is inconsistent (with respect to CET), and therefore

$$comp(P) \models_3 \neg(\theta, (\underline{s} = \underline{t}^i), \sigma^i, \underline{M}^i, \underline{L'})$$

But then,

$$comp(P) \models_3 G \cong G'_1 \;\wedge\; \ldots \;\wedge\; G'_n$$

$\Box$

Proof: (of Theorem 11.1)
We prove by complete induction over k that for all natural numbers k and all goals $G : \leftarrow \theta, \underline{L}$,

1. If δ is a computed answer of a rank k SLDFA-refutation for G, then $comp(P) \models_3 \delta \rightarrow \theta \ \wedge \ \underline{L}$.
2. If G is a rank k finitely failed goal, then $comp(P) \models_3 \theta \rightarrow \neg \underline{L}$.

Assume that 1. and 2. hold for all ranks smaller than k. We first prove 1. for rank k, using the induction hypothesis, and then prove 2. for rank k, using the fact that we already have proven 1. for rank k.

(1.) Suppose that G has a rank k SLDFA-refutation $G'_1, \ldots, G'_n$ with computed answer δ. For a moment, let us assume that, for all $i \in [1..n-1]$, $comp(P) \models_3 G'_i \rightarrow G'_{i+1}$. Then, it follows by straightforward induction that

$$comp(P) \models_3 G'_0 \rightarrow \ldots \rightarrow G'_n$$

Because $G'_1, \ldots, G'_n$ is a refutation of G, we have that G'_0 is of the form $\leftarrow \theta, \underline{L}$ and G'_n is of the form $\leftarrow \tau$. But then, it follows that

$$comp(P) \models_3 \neg(\theta \ \wedge \ \underline{L}) \rightarrow \neg \tau$$

which can be rewritten as

$$comp(P) \models_3 \tau \rightarrow \theta \ \wedge \ \underline{L}$$

Now, δ is of the form $\exists \underline{y} \tau$, where $\underline{y} = FreeVar(\tau) - FreeVar(G)$. But then, the free variables of τ that are existentially quantified in δ, do not occur in $\theta \ \wedge \ \underline{L}$, and therefore,

$$comp(P) \models_3 \delta \rightarrow \theta, \underline{L}$$

So, to prove 1. for rank k, it is sufficient to prove that, for all $i \in [1..n-1]$, $comp(P) \models_3 G'_i \rightarrow G'_{i+1}$. Because every G'_{i+1} is either positively or negatively derived from G'_i, there are two cases:

- Suppose that G'_{i+1} is positively derived from G'_i. Then, by Lemma 11.2,

  $$comp(P) \models_3 G'_i \cong G''_1 \ \wedge \ \ldots \ \wedge \ G''_m$$

  where $G''_1, \ldots, G''_m$ contains all goals that are positively derivable from G'_i. Because G'_{i+1} is positively derivable from G'_i, it is a member of $G''_1, \ldots, G''_m$, and therefore

  $$comp(P) \models_3 G'_i \rightarrow G'_{i+1}$$

- Suppose that G'_{i+1} is negatively derivable from G'_i. Then, G'_i is of the form $\leftarrow \rho, \neg A, \underline{L'}$ (we assume without loss of generality that the left-most literal is selected) and G'_{i+1} is of the form $\leftarrow \rho, \sigma, \underline{L'}$, such that $\leftarrow \rho, \sigma, A$ has a rank $k-1$ finitely failed SLDFA-tree. By induction hypothesis (part 2.), it follows that

  $$comp(P) \models_3 \rho \ \wedge \ \sigma \rightarrow \neg A$$

  But then, we also have that

  $$comp(P) \models_3 \rho \ \wedge \ \sigma \ \wedge \ \underline{L'} \rightarrow \rho \ \wedge \ \neg A \ \wedge \ \underline{L'}$$

  which can be rewritten as

$$comp(P) \models_3 \neg(\rho \wedge \neg A \wedge \underline{L'}) \rightarrow \neg(\rho \wedge \sigma \wedge \underline{L'})$$

and is equivalent to

$$comp(P) \models_3 G'_i \rightarrow G'_{i+1}$$

Thus, we have proven 1. for arbitrary goals with rank k SLDFA-computed answers.

(2.) Suppose that $\leftarrow \theta, \underline{L}$ is the root of a rank k finitely failed SLDFA-tree. We have to prove that $comp(P) \models_3 \theta \rightarrow \neg\underline{L}$. We prove this by complete induction over the depth l of rank k finitely failed SLDFA-trees. Assume that the claim holds for all rank k finitely failed SLDFA-trees with a depth smaller than l. Now, suppose we have a rank k finitely failed SLDFA-tree of depth l, where the root G has the form $\leftarrow \theta, L_1, \ldots, L_k$. Let us assume, without loss of generality, that the left-most literal (i.e. L_1) is selected. Let the sons of G be

$$G_1 : \leftarrow \sigma_1, \underline{M}_1 \quad \ldots \quad G_n : \leftarrow \sigma_n, \underline{M}_n$$

Each G_i is the root of a rank k finitely failed SLDFA-tree of depth $l - 1$. Therefore, by induction hypothesis, for all $i \in [1..n]$,

$$comp(P) \models_3 \sigma_i \rightarrow \neg\underline{M}_i$$

which can be rewritten as

$$comp(P) \models_3 G_i$$

Now, there are two cases:

- Suppose that L_1 is the positive literal A. Then, by definition, for every clause R in P that is applicable to A, G has exactly one son which is positively derived from G using a variant of R. By Lemma 11.2 it follows that

  $$comp(P) \models_3 G \cong G_1 \wedge \ldots \wedge G_n$$

  Because for all $i \in [1..n]$, $comp(P) \models_3 G_i$, we have that $comp(P) \models_3 G$. Which can be rewritten as $comp(P) \models_3 \theta \rightarrow \neg\underline{L}$.

- Suppose that L_1 is the negative literal $\neg A$. Then, by definition, there exist rank k SLDFA-computed answers $\delta_1, \ldots, \delta_m$ for $\leftarrow \theta, A$ such that

  $$CET \models_3 \theta \rightarrow \delta_1 \vee \ldots \vee \delta_m \vee \sigma_1 \vee \ldots \vee \sigma_n$$

  We can make the following two observations.

  - For all $i \in [1..m]$, it follows from having proven 1 for rank k, that $comp(P) \models_3 \delta_i \rightarrow A$ and therefore $comp(P) \models_3 \delta_i \rightarrow \neg\underline{L}$.
  - For all $i \in [1..n]$, $comp(P) \models_3 \sigma_i \rightarrow \neg\underline{M}_i$, and, by definition, $\underline{M}_i$ is of the form $L_2, \ldots, L_k$. Therefore, $comp(P) \models_3 \sigma_i \rightarrow \neg\underline{L}$.

  But then, it follows that $comp(P) \models_3 \theta \rightarrow \neg\underline{L}$. □

The following corollary proves soundness of SLDFA-resolution with respect to the three-valued completion semantics for abductive logic programs, as stated in definition 7.

Corollary 11.3 (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 11.1, $comp(P) \models_3 \delta \rightarrow \theta \wedge \underline{L}$. Moreover, δ has a 3-valued model, which implies that $comp(P) \cup \{\delta\}$ is consistent. But then, it follows that $comp(P) \cup \{\delta\} \models_3 \theta \wedge \underline{L}$. 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 11.4 *Let P be a program and let $G : \leftarrow \theta, \underline{L}$ be a goal.*

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

Proof:

1. Suppose that δ is an SLDFA-computed answer for G. Then, by Theorem 11.1, we have that $comp(P) \models_3 \delta \rightarrow \theta \wedge \underline{L}$. But every two-valued model for $comp(P)$ is also a three-valued model for $comp(P)$, and therefore $comp(P) \models \delta \rightarrow \theta \wedge \underline{L}$.
2. Suppose that G finitely fails. Then, by Theorem 11.1, $comp(P) \models_3 \theta \rightarrow \neg\underline{L}$. But every two-valued model for $comp(P)$ is also a three-valued model for $comp(P)$. Therefore, we have that $comp(P) \models \theta \rightarrow \neg\underline{L}$. □

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

Corollary 11.5 (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 11.4, $comp(P) \models \delta \rightarrow \theta \wedge \underline{L}$. But then, because $comp(P) \cup \{\delta\}$ is consistent, $comp(P) \cup \{\delta\} \models \theta \wedge \underline{L}$. Thus, δ is an explanation for $\langle P, \theta \wedge \underline{L}\rangle$. □

12. Completeness of generalized SLDFA-resolution

In this section, we prove completeness of the generalized SLDFA-resolution with respect to three-valued completion semantics.

Theorem 12.1 *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,*

1. *if $comp(P) \cup \{\delta\} \models_3 \theta \wedge \underline{L}$, then there exist SLDFA-computed answers $\delta_1, \ldots, \delta_n$ for G such that $CET \models_3 \delta \rightarrow \delta_1 \vee \ldots \vee \delta_n$, and*
2. *if $comp(P) \models_3 \theta \rightarrow \neg\underline{L}$ then G fails finitely.*

As was the case with Theorem 11.1, the proof of this theorem is (almost) identical to the proof of the corresponding theorem in [Dra93b] (Theorem 5.1). The only difference is, that we use results from Section 7, where Drabent used results from [Kun87].

Before giving the proof of the theorem, we present three (technical) lemmas.

Lemma 12.2 *Let P be a program, let $\leftarrow \theta, \underline{L}$ be a rank k finitely failed goal and let σ be a consistent abducible formula. If $CET \models_3 \sigma \rightarrow \theta$, then $\leftarrow \sigma, \underline{L}$ is a rank k finitely failed goal.*

Proof: Suppose that $\leftarrow \theta, \underline{L}$ has a rank k finitely failed SLDFA-tree. Then, there exists a rank k finitely failed SLDFA-tree for $\leftarrow \theta, \underline{L}$ such that, for all variables x occurring in σ but not in $\leftarrow \theta, \underline{L}$, x does not occur in that SLDFA-tree. From this SLDFA-tree, we can construct a rank k finitely failed SLDFA-tree for $\leftarrow \theta, \sigma, \underline{L}$, by adding σ to every node in the tree and then pruning subtrees whose roots contain an inconsistent abducible formula. Because $CET \models_3 \sigma \rightarrow \theta$, the resulting tree is also a rank k finitely failed SLDFA-tree for $\leftarrow \sigma, \underline{L}$. □

Lemma 12.3 *Let P be a program. Let δ be a computed answer for $\leftarrow \theta, \underline{L}$. Then, for any abducible formula σ such that σ, δ is consistent, σ, δ is (equivalent to) an SLDFA-computed answer for $\leftarrow \sigma, \theta, \underline{L}$.*

Proof: Suppose that $\leftarrow \theta, \underline{L}$ has an SLDFA-refutation. Then, it also has an SLDFA-refutation $G'_1, \ldots, G'_n$ such that, for all variables x occurring in σ but not in $\leftarrow \theta, \underline{L}$, x does not occur in$G'_1, \ldots, G'_n$. Now let, for $i \in [1..n]$, G'_i be of the form $\leftarrow \theta_i, \underline{L}_i$ and G''_i be of the form $\leftarrow \sigma, \theta_i, \underline{L}_i$. Then, δ is of the form $\exists_{\underline{y'}}\theta_n$, where $\underline{y'} = FreeVar(\theta_n) - FreeVar(\theta, \underline{L})$. We prove that $G''_1, \ldots, G''_n$ is an SLDFA-refutation of $\leftarrow \sigma, \theta, \underline{L}$, and that its computed answer is equivalent to σ, δ.

To prove that $G''_1, \ldots, G''_n$ is an SLDFA-refutation, it suffices to prove that, for all $i \in [1..n]$, σ, θ_i is consistent. We know that $\sigma \wedge \exists_{\underline{y}}\theta_n$ is consistent and that θ_n is consistent. Because $\underline{y'}$ only quantifies variables that do not occur in σ, it follows that σ, θ_n is consistent. Now, assume that σ, θ_i is consistent. Because G'_i is (positively or negatively) derived from G'_{i-1}, we have by the definition of positively and negatively derived goals that $CET \models_3 \theta_i \rightarrow \theta_{i-1}$. From this, and the fact that σ, θ_i is consistent, it follows that σ, θ_{i-1} is consistent. Thus, $G''_1, \ldots, G''_n$ is an SLDFA-refutation.

Now, G''_n is of the form $\leftarrow \sigma, \theta_n$ and therefore, for $\underline{y''} = FreeVar(\sigma, \theta_n) - FreeVar(\leftarrow \sigma, \theta, \underline{L})$, the formula $\exists_{\underline{y''}}(\sigma, \theta_n)$ is a computed answer for $\leftarrow \sigma, \theta, \underline{L}$. Because σ occurs in $\leftarrow \sigma, \theta, \underline{L}$, the variables in $\underline{y''}$ do not occur in σ, and therefore

$$\exists_{\underline{y''}}(\sigma, \theta_n) \cong \sigma \wedge \exists_{\underline{y''}}\theta_n \cong \sigma, \delta$$

□

The following lemma will form the core of the proof of Theorem 12.1. In the lemma, and in the proof of the theorem, we use the following notation.

Definition 12.4 Let $L_1, \ldots, L_m$ be a sequence of literals, and let $n_1, \ldots, n_m$ be a sequence of natural numbers. Then

$$T_{n_1}, \ldots, n_m(L_1, \ldots, L_m) \overset{def}{=} T_{n_1}(L_1) \wedge \ldots \wedge T_{n_m}(L_m)$$

□

Note, that by definition of T_n and F_n, for a sequence $L_1, \ldots, L_k$ of literals, $T_n(\underline{L}) \cong T_{\underline{n}}(\underline{L})$ and $F_n(\underline{L}) \cong F_{\underline{n}}(\underline{L})$, where $\underline{n}$ is the sequence $n, \ldots, n$ of length k.

Lemma 12.5 *Let $L_1, \ldots, L_k$ be a sequence of non-abducible literals and let $n_1, \ldots, n_k$ be a sequence of natural numbers. Then, for arbitrary fair selection rules,*

1. *There exist computed answers $\delta_1, \ldots, \delta_l$ for $\leftarrow \underline{L}$ such that*

$$CET \models_3 T_{\underline{n}}(\underline{L}) \rightarrow \delta_1 \vee \ldots \vee \delta_l$$

2. *For any sequence $\underline{M}$ of literals, $\leftarrow F_{\underline{n}}(\underline{L}), \underline{L}, \underline{M}$ either fails, or $F_{\underline{n}}(\underline{L})$ is inconsistent.*

Proof: The proof of the two claims is by induction on $n_1, \ldots, n_k$, using the multiset ordering. For the base case, where $\underline{n} = 0$ ($k = 1$), the two claims are trivially true, because $T_0(L) = F_0(L) = \mathbf{f}$ holds for arbitrary non-abducible literals L. Assume that we have proven the two claims for all $\underline{L'}$ and $\underline{n'}$ such that $\underline{n'}$ is smaller than $\underline{n}$ (in the multiset order).

Now, we prove the two claims for $\underline{n}$ and $\underline{L}$.

(1.) We have to prove that there exist computed answers $\delta_1, \ldots, \delta_l$ for $\leftarrow \underline{L}$ such that

$$CET \models_3 T_{\underline{n}}(\underline{L}) \rightarrow \delta_1 \vee \ldots \vee \delta_l$$

Suppose that $n_1 = 0$. Then, $T_{n_1}(L_1) = \mathbf{f}$ and therefore the claim is trivially true. So, assume that $n_1 > 0$. Without loss of generality, let us assume that the selected literal in $\leftarrow L_1, \ldots, L_k$ is L_1. Let $\underline{L'}$ be the sequence $L_2, \ldots, L_k$ and let $\underline{n'}$ be the sequence $n_2, \ldots, n_k$. There are two cases:

1. L_1 is of the form $p(\underline{s})$. Let

$$p(\underline{t}^1) \leftarrow \sigma^1, \underline{M}^1 \quad \ldots \quad p(\underline{t}^m) \leftarrow \sigma^m, \underline{M}^m$$

contain a variant for each clause in P with head p. Assume that these clauses are standardized apart from each other and from $\underline{L}$. consider the following clauses:

$$\leftarrow (\underline{s} = \underline{t}^1), \sigma^1, \underline{M}^1, \underline{L'} \quad \ldots \quad \leftarrow (\underline{s} = \underline{t}^m), \sigma^m, \underline{M}^m, \underline{L'}$$

Let, for $i \in [1..m]$, $\underline{n^i}$ be the sequence $(n_1 - 1), \ldots, (n_1 - 1)$, where the length of $\underline{n^i}$ is equal to the length of $\underline{M^i}$, and let $\underline{y}^i = \mathit{FreeVar}(\sigma^i, \underline{M}^i) - \mathit{FreeVar}(p(\underline{t}^i))$. By applying the induction hypothesis for each $i \in [1..m]$, we have that there exist computed answers $\delta^i{}_1, \ldots, \delta^i{}_{v^i}$ for $\leftarrow \underline{M^i}, \underline{L'}$ such that

$$CET \models_3 T_{\underline{n^i}.\underline{n'}}(\underline{M^i}, \underline{L'}) \rightarrow \delta^i_1 \vee \ldots \vee \delta^i_{v^i}$$

We proceed by first showing that $CET \models_3 T_{\underline{n}}(\underline{L}) \rightarrow \phi$, where ϕ is a disjunction of abducible formulas, and then proving that each disjunct of ϕ is either inconsistent (with respect to CET) or (equivalent to) a computed answer for $\leftarrow \underline{L}$.

To begin with, we have by definition of $T_n(\phi)$ that

$$\begin{array}{rl} & T_{n_1}(p(\underline{s})) \\ \cong & T_{n_1-1}\left(\bigvee_{i\in[1..m]} \exists_{\underline{y^i}}((\underline{s}=\underline{t}^i),\sigma^i,\underline{M}^i)\right) \\ \cong & \bigvee_{i\in[1..m]} \exists_{\underline{y^i}}((\underline{s}=\underline{t}^i),\sigma^i,T_{n_1-1}(\underline{M}^i)) \\ \cong & \bigvee_{i\in[1..m]} \exists_{\underline{y^i}}((\underline{s}=\underline{t}^i),\sigma^i,T_{\underline{n^i}}(\underline{M}^i)) \end{array}$$

But then,

$$\begin{array}{rl} & T_{\underline{n}}(\underline{L}) \\ \cong & (\bigvee_{i\in[1..m]} \exists_{\underline{y^i}}((\underline{s}=\underline{t}^i),\sigma^i,T_{\underline{n^i}}(\underline{M}^i))) \ \wedge \ T_{\underline{n'}}(\underline{L'}) \\ \cong & \bigvee_{i\in[1..n]} \exists_{\underline{y^i}}((\underline{s}=\underline{t}^i),\sigma^i,T_{\underline{n^i},\underline{n'}}(\underline{M}^i,\underline{L'})) \end{array}$$

and therefore

$$CET \models_3 T_{\underline{n}}(\underline{L}) \rightarrow \bigvee_{i\in[1..m]} \exists_{\underline{y^i}}((\underline{s}=\underline{t}^i),\sigma^i,(\delta^i_1 \ \vee \ \ldots \ \vee \ \delta^i_{v^i}))$$

This formula can be rewritten as

$$CET \models_3 T_{\underline{n}}(\underline{L}) \rightarrow \bigvee_{i\in[1..m]} \bigvee_{j\in[1..v^i]} \exists_{\underline{y^i}}((\underline{s}=\underline{t}^i),\sigma^i,\delta^i_j)$$

For this formula we prove that each disjunct on the right-hand side of the implication is either inconsistent with CET or equivalent to a computed answer for $\leftarrow \underline{L}$.

Recall that, for $i \in [1..m]$ and $j \in [1..v^i]$, δ^i_j is a computed answer for $\leftarrow \underline{M}^i, \underline{L'}$. But then, by Lemma 12.3, $(\underline{s}=\underline{t}^i),\sigma^i,\delta^i_j$ is either inconsistent (with respect to CET) or a computed answer for $\leftarrow (\underline{s}=\underline{t}^i),\sigma^i,\underline{M}^i,\underline{L'}$. If $(\underline{s}=\underline{t}^i),\sigma^i,\delta^i_j$ is a computed answer, we may assume that it is obtained from a refutation that does not use variables that occur in $\underline{L}$. From this refutation, we can construct a refutation of $\leftarrow \underline{L}$, by putting the goal $\leftarrow \underline{L}$ in front. But then, for all $i \in [1..m]$ and $j \in [1..v^i]$, $\exists_{\underline{y^i}}((\underline{s}=\underline{t}^i)\sigma^i,\delta^i_j)$ is either inconsistent or (equivalent to) a computed answer for $\leftarrow \underline{L}$.

2. Suppose that L_1 is the negative literal $\neg A$. We have that

$$T_{n_1}(\neg A) \overset{def}{=} F_{n_1}(A)$$

But then,

$$T_{\underline{n}}(\underline{L}) \cong F_{n_1}(A) \ \wedge \ T_{\underline{n'}}(\underline{L'})$$

By the inductive hypothesis for A and n_1, $\leftarrow F_{n_1}(A),\underline{L'}$ is negatively derived from $\leftarrow \underline{L}$. By the inductive assumption 2. for $\underline{L'}$ and $\underline{n'}$, we obtain computed answers for $\underline{L}$.

(2.) If $n_1 = 0$, then $F_{\underline{n}}(\underline{L}) \cong F_{\underline{n'}}(\underline{L'})$ and $\leftarrow F_{\underline{n'}}(\underline{L'}),\underline{L'}$ fails by induction hypothesis, which implies that $\leftarrow F_{\underline{n'}}(\underline{L'}),\underline{L'},\underline{M}$ fails finitely. So, assume that $n_1 > 0$.

Consider the tree T, which is constructed as follows:

- The root node of T is $\leftarrow F_{\underline{n}}(\underline{L}),\underline{L},\underline{M}$.
- All nodes in T are of the form $\leftarrow F_{\underline{n}}(\underline{L}),\rho,\underline{L},\underline{N}$ (for some ρ and $\underline{N}$).

- For all nodes $\leftarrow F_{\underline{n}}(\underline{L}), \rho, \underline{L}, \underline{N}$ in T,
  - if the selected literal is a member of $\underline{L}$, then the node is a leaf, and
  - if the selected literal is a member of $\underline{N}$, then the children of the node are defined according to definition 5.6.

Clearly, T is the 'top part' of some SLDFA-tree for $\leftarrow F_{\underline{n}}(\underline{L}), \underline{L}, \underline{M}$.

We prove that T can be extended to a finitely failed SLDFA-tree for $\leftarrow F_{\underline{n}}(\underline{L}), \underline{L}, \underline{M}$. For this, we have to prove that we can extend all leaves $\leftarrow F_{\underline{n}}(\underline{L}), \rho, \underline{L}, \underline{N}$ in which a literal from $\underline{L}$ is selected. It is sufficient to prove that, for all ρ and $\underline{N}$, $\leftarrow F_{\underline{n}}(\underline{L}), \rho, \underline{L}, \underline{N}$ (where the selected literal is a member from $\underline{L}$) fails finitely. Without loss of generality, let us assume that the selected literal from $\underline{L}$ is L_1. Let $\underline{L'}$ be the sequence $L_2, \ldots, L_k$ and let $\underline{n'}$ be the sequence $n_2, \ldots, n_k$. There are two cases:

- L_1 is of the form $p(\underline{s})$. Let

$$p(\underline{t}^1) \leftarrow \sigma^1, \underline{M}^1 \quad \ldots \quad p(\underline{t}^m) \leftarrow \sigma^m, \underline{M}^m$$

  contain a variant for each clause in P with head p. Assume that these clauses are standardized apart from each other and from $\underline{L}$. Let, for $i \in [1..m]$, $\underline{n^i}$ be the sequence $(n_1 - 1), \ldots, (n_1 - 1)$, where the length of $\underline{n^i}$ is equal to the length of $\underline{M^i}$ and let $\underline{y}^i = \mathit{FreeVar}(\sigma^i, \underline{M}^i) - \mathit{FreeVar}(p(\underline{t}^i))$. We have, by definition of $F_n(\phi)$, that

$$\mathit{comp}(P) \models_3 F_{n_1}(p(\underline{s})) \cong \bigwedge_{i \in [1..m]} \forall_{\underline{y}^i}((\underline{s} = \underline{t}^i), \sigma^i \rightarrow F_{\underline{n}^i}(\underline{M}^i))$$

  But then, for all $i \in [1..m]$,

$$\mathit{comp}(P) \models_3 F_{n_1}(p(\underline{s})), (\underline{s} = \underline{t}^i), \sigma^i \rightarrow F_{\underline{n}^i}(\underline{M}^i)$$

  and therefore

$$\mathit{comp}(P) \models_3 F_{\underline{n}}(\underline{L}), (\underline{s} = \underline{t}^i), \sigma^i \rightarrow F_{\underline{n}^i}(\underline{M}^i) \ \vee \ F_{\underline{n'}}(\underline{L'})$$

  which can be rewritten as

$$\mathit{comp}(P) \models_3 F_{\underline{n}}(\underline{L}), (\underline{s} = \underline{t}^i), \sigma^i \rightarrow F_{\underline{n}^i, \underline{n'}}(\underline{M}^i, \underline{L'})$$

  By induction hypothesis, for all $i \in [1..m]$, we have that either $F_{\underline{n}^i, \underline{n'}}(\underline{M}^i, \underline{L'})$ is inconsistent or $\leftarrow F_{\underline{n}^i, \underline{n'}}(\underline{M}^i, \underline{L'}), \underline{M}^i, \underline{L}, \underline{N}$ fails finitely for any fair computation rule. But then, by Lemma 12.2, for all $i \in [1..m]$, $F_{\underline{n}}(\underline{L}), \rho, (\underline{s} = \underline{t}^i), \sigma^i$ is inconsistent or $\leftarrow F_{\underline{n}}(\underline{L}), \rho, (\underline{s} = \underline{t}^i), \sigma^i, \underline{M}^i, \underline{L'}, \underline{N}$ fails finitely for any fair computation rule. But then, there exists a finitely failed SLDFA-tree for $\leftarrow F_{\underline{n}}(\underline{L}), \rho, \underline{L}, \underline{N}$

- Suppose that L_1 is of the form $\neg A$. By definition, $F_{n_1}(\neg A)$ is equivalent to $T_{n_1}(A)$. But then,

$$F_{\underline{n}}(\underline{L}) \cong T_{n_1}(A) \ \vee \ F_{\underline{n'}}(\underline{L'})$$

  By induction hypothesis, it follows that $\leftarrow F_{\underline{n'}}(\underline{L'}), \underline{L'}, \underline{N}$ has a finitely failed SLDFA-tree. If we extend this tree by making the node $\leftarrow F_{\underline{n}}(\underline{L}), \underline{L}, \underline{N}$ the parent of $\leftarrow F_{\underline{n'}}(\underline{L'}), \underline{L'}, \underline{N}$, we have build a finitely failed SLDFA-tree for $\leftarrow F_{\underline{n}}(\underline{L}), \underline{L}, \underline{N}$. But then, by Lemma 12.2, there exists a finitely failed SLDFA-tree for $\leftarrow F_{\underline{n}}(\underline{L}), \rho, \underline{L}, \underline{N}$ □

Proof:(of Theorem 12.1)
We have to prove that

1. If $comp(P) \cup \{\delta\} \models_3 \theta, \underline{L}$, then there exist SLDFA-computed answers $\delta_1, \ldots, \delta_n$ for $\leftarrow \theta, \underline{L}$ such that $CET \models_3 \delta \rightarrow \delta_1 \ \vee \ \ldots \ \vee \ \delta_n$.
2. If $comp(P) \models_3 comp(P) \models_3 \theta \rightarrow \neg\underline{L}$ then G fails finitely.

(1.) From $comp(P) \cup \{\delta\} \models_3 \theta, \underline{L}$, it follows that $comp(P) \models_3 \delta \rightarrow \theta, \underline{L}$. By Corollary 8.6 and Theorem 10.10, for some n, $CET \models_3 T_n(\forall(\delta \rightarrow \theta \ \wedge \ \underline{L}))$, which, by definition of T_n, is equivalent to $CET \models_3 \delta \rightarrow \theta \ \wedge \ T_n(\underline{L})$. Let $\underline{n}$ be the sequence $n, \ldots, n$, whose length is the same as the length of $\underline{L}$. By Lemma 12.5, there exist SLDFA-computed answers $\delta'_1, \ldots, \delta'_l$ for $\leftarrow \underline{L}$ such that

$$CET \models_3 T_{\underline{n}}(\underline{L}) \rightarrow \delta'_1 \ \vee \ \ldots \ \vee \ \delta'_l$$

But then, we also have that

$$CET \models_3 delta \rightarrow (\theta, \delta'_1) \ \vee \ \ldots \ \vee \ (\theta, \delta'_l)$$

Moreover by Lemma 12.3, for each $i \in [1..l]$, θ, δ'_i is either inconsistent, or a SLDFA-computed answer for $\leftarrow \theta, \underline{L}$. Let $\delta_1, \ldots, \delta_k$ contain all θ, δ'_i that are consistent. Then, because for those $i \in [1..l]$, for which θ, δ'_i is inconsistent, we have that $CET \not\models_3 \theta, \delta'_i$, it follows that

$$CET \models_3 \delta \rightarrow \delta_1 \ \vee \ \ldots \ \vee \ \delta_k$$

(2.) Suppose that $comp(P) \models_3 \theta \rightarrow \neg\underline{L}$. Then, by Corollary 8.6 and Theorem 10.10, for some n, $CET \models_3 T_n(\forall\theta \rightarrow \neg\underline{L}))$ and therefore, by definition of T_n and F_n, $CET \models_3 \theta \rightarrow F_n(\underline{L})$. Let $\underline{n}$ be the sequence $n, \ldots, n$, whose length is the same as the length of $\underline{L}$. By Lemma 12.5, $F_{\underline{n}}(\underline{L}), \underline{L}$ fails finitely. But then, because $CET \models_3 \theta \rightarrow F_{\underline{n}}(\underline{L})$, by Lemma 12.2, $\leftarrow \theta, \underline{L}$ fails finitely. □

Corollary 12.6 (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 $CET \models_3 \delta \rightarrow \delta_1 \ \vee \ \ldots \ \vee \ \delta_k$.*

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 12.1, there exist SLDFA-computed answers $\delta_1, \ldots, \delta_k$ for $\leftarrow \theta, \underline{L}$ such that $CET \models_3 \delta \rightarrow \delta_1 \ \vee \ \ldots \ \vee \ \delta_k$. □

13. Conclusions

In 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. 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 should use three-valued completion, because then inconsistencies are the result of flaws in the program.

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 [DS93]. The 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 [Den93]). 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).

Acknowledgement

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.

References

[AB91] K.R. Apt and M. Bezem. Acyclic programs. *New Generation Computing*, 9:335–363, 1991.

[CDT91] L. Console, D.T. Dupre, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[Cha88] D. Chan. Constructive negation based on the completed database. In *Proceedings of the International Conference on Logic Programming*, pages 111–125. MIT Press, 1988.

[CK73] C.C. Chang and H.J. Keisler. *Model Theory.* Number 73 in Studies in Logic and the Foundations of Mathematics. North-Holland Publishing Company, 1973.

[Cla78] K.L. Clark. Negation as failure. In H. Gallaire and G. Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, 1978.

[Den93] M. Denecker. *Knowledge Representation and Reasoning in Incomplete Logic Programming.* PhD thesis, Katholieke Universiteit Leuven, Leuven, Belgium, September 1993.

[Doe93] K. Doets. *From Logic to Logic Programming.* The MIT Press series in Foundations of Computing. MIT Press, 1993.

[Dra93a] W. Drabent. SLS-resolution without floundering. In *Proceedings of the workshop on Logic Programming and Non-Monotonic Reasoning*, 1993.

[Dra93b] W. Drabent. What is failure? an approach to constructive negation. Updated version of a Technical Report LITH-IDA-R-91-23 at Linkoping University, 1993.

[DS92] M. Denecker and D. De Schreye. SLDNFA: an abductive procedure for normal abductive programs. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 686–700, 1992.

[DS93] M. Denecker and D. De Schreye. Representing incomplete knowledge in abductive logic programming. In *Proceedings of the International Logic Programming Symposium*, 1993.

[Fag94] F. Fages. Constructive negation by pruning. Working paper, May 1994.

[Fit85] M. Fitting. A Kripke-Kleene semantics for logic programs. *Journal of Logic Programming*, 2(4):295–312, 1985.

[GL90] M. Gelfond and V. Lifschitz. Logic programs with classical negation. In *Proceedings of the International Conference on Logic Programming*, pages 579–597, 1990.

[GL92] M. Gelfond and V. Lifschitz. Representing actions in extended logic programming. In *Proceedings of the Joint International Conference and Symposium on Logic Programming*, pages 559–573, 1992.

[KKT92] A.C. Kakas, R.A. Kowalski, and F. Toni. Abductive logic programming. *Journal of Logic and Computation*, 2:719–770, 1992.

[Kun87] K. Kunen. Negation in logic programming. *Journal of Logic Programming*, 4:289–308, 1987.

[Llo87] J.W. Lloyd. *Foundations of Logic Programming.* Symbolic Computation – Artificial Intelligence. Springer-Verlag, 1987. Second, extended edition.

[She88] J.C. Shepherdson. Language and equality theory in logic programming. Technical Report PM-88-08, School of Mathematics, University Walk, Bristol, BS8 1 TW, England, 1988.

[Stu91] P.J. Stuckey. Constructive negation for constraint logic programming. In *Proceedings of the IEEE Symposium on Logic in Computer Science*, pages 328–339. IEEE Computer Society Press, July 1991.

[Teu94] F.J.M. Teusink. Three-valued completion for abductive logic programs. In Giorgio Levi and Mario Rodríguez-Artalejo, editors, *Proceedings of the International Conference on Algebraic and Logic Programming*, number 850 in Lecture Notes in Computer Science, pages 150–167. Springer-Verlag, 1994.