

SLDNFA: an abductive procedure for abductive logic programs

Marc Denecker Danny De Schreye*

Department of Computer Science, K.U.Leuven,
Celestijnenlaan 200A, B-3001 Heverlee, Belgium.
e-mail : {marcd, dannyd}@cs.kuleuven.ac.be

Abstract

We present SLDNFA, an extension of SLDNF-resolution for abductive reasoning on abductive logic programs. SLDNFA solves the floundering abduction problem: non-ground abductive atoms can be selected. SLDNFA provides also a partial solution for the floundering negation problem. Different abductive answers can be derived from an SLDNFA-refutation; these answers provide different compromises between generality and comprehensibility. Two extensions of SLDNFA are proposed which satisfy stronger completeness results. The soundness of SLDNFA and its extensions is proven. Their completeness for minimal solutions with respect to implication, cardinality and set inclusion is investigated. The formalisation of SLDNFA presented here is an update of an older version presented in [13] and does not rely on skolemisation of abductive atoms.

1 Introduction

The role of abduction [48] as a reasoning paradigm in AI is widely accepted. Abduction has been used for fault diagnosis [6], natural language understanding [6], default reasoning [22], [50]. In the context of logic programming, abductive procedures have been used for planning [21], [52], [46, 44], knowledge assimilation and belief revision [32], [30], database updating [31]. [18] showed the role of an abductive system for forms of reasoning, different from planning, in the context of temporal domains with uncertainty. In [15, 17], the role of abductive logic programming for representing uncertainty in a logic programming formalism and the role of abductive procedures for abductive and deductive reasoning and satisfiability checking on incomplete knowledge has been shown in the context of a translation from a temporal language \mathcal{A} [27] to abductive logic programming.

In the past, a number of abductive extensions of SLDNF-resolution have been proposed for abductive logic programs with negation [21], [52], [46, 45, 44],[31], [8], [51], [29], [13], [54]. Anticipating the discussion of these procedures in section 11, we can say that either these procedures have not been formalised and proven correct

*senior research associate of the Belgian National Fund for Scientific Research

[21], [52], or they can be proven correct only for a restricted class of abductive logic programs [46, 45, 44], or they do not provide a way of checking the consistency of the abductive answers [8], [54], or they do not provide a treatment for *floundering abduction* [31], [51], [29]. The *floundering abduction* problem is an analogue of the well known *floundering negation* problem which arises when a non-ground negative literal is selected; analogously, the problem of *floundering abduction* arises when a non-ground abductive atom is selected. While it seems that for -from a practical point of view- important classes of non-abductive logic programs, the floundering negation problem does not occur or can be avoided (e.g. in allowed programs), there is empirical evidence that floundering abduction cannot be prevented in many applications of abductive logic programming.

The floundering abduction problem appeared already in the early work on abduction for planning in the context of event calculus in [21], [52]. The (simplified) main axiom of event calculus is the inertia law which can be formulated as follows:

$$\begin{aligned} holds_at(F, T) &\leftarrow happens(E), E < T, initiates(E, F), \\ &\quad \neg clipped(E, F, T) \\ clipped(E, F, T) &\leftarrow happens(C), E < C < T, terminates(C, F) \end{aligned}$$

The axioms state that a property F holds at time T iff it is initiated at some earlier time E and has not been terminated between E and T . The interpretation of most symbols is rather straightforward; $clipped(E, F, T)$ represents the condition that some event in the interval $]E, T[$ terminates the property F . In the context of a planning problem, the effects of possible actions are described by a set of rules defining *initiates* and *terminates* which depend on the *act* predicate. For example, one effect of an act of putting an object X on top of an object Y is specified in the following rule:

$$initiates(E, on(X, Y)) \leftarrow act(E, put(X, Y))$$

A planning problem may be formulated by means of a goal $\leftarrow holds(p, t_{end}), \dots$ which gives a description of the properties one requires to hold in the final state (after the plan will be executed). The predicates which describe the actions and their interrelations, i.e. *happens*, $<$ and *act*, are abducible. Hence, an abductive procedure searches for a set Δ of these facts such that $P + \Delta \models holds(p, t_{end}) \wedge \dots$. The set Δ constitutes a plan for the desired final state.

In this setting, floundering abduction occurs very naturally. E.g. consider the desired final state $holds(on(a, b), t_{end})$. By calling the goal $\leftarrow holds(on(a, b), t_{end})$ and resolving first with the inertia law and then with the clause for *initiates*, one obtains the goal:

$$\leftarrow happens(E), E < t_{end}, act(E, put(a, b)), \neg clipped(E, on(a, b), t_{end})$$

In this query, all literals are either non-ground negative or non-ground abducible.

The procedures proposed in [21], [52] and later the one proposed in [46, 45, 44] were developed for planning in the event calculus and provide a treatment for non-ground abducible atoms: non-ground abducible atoms are *skolemised*, i.e. all variables are substituted by fresh constants. As indicated above, these procedures were either not formalised (and hence, not proven correct) or only partially sound. In [13], we

presented SLDNFA-resolution, a sound and -under some conditions- complete abductive extension of SLDNF. In several respects, this procedure builds on ideas in the elder procedures and, in the presentation in [13], uses also skolemisation of non-ground abducible atoms to solve the floundering abduction problem.

In this paper, we present a more complete study of SLDNFA. An SLDNFA computation can be understood as a process of deriving formulas of the form $\forall(Q_0 \leftarrow \Psi)$, with $\leftarrow Q_0$ the initial query and the conjunction Ψ composed of the unsolved goals, from the abductive logic program. The formalisation of SLDNFA here, differs in two important ways of the formalisation in [13]. One difference is that here, the definition of SLDNFA is based on a similar schema as used in [1] and [43] to define SLDNF. A second major difference is that the current formalisation does not longer rely on skolemisation of abducible atoms. Instead, a formalisation is used which distinguishes two types of variables: those universally quantified in front of $\forall(Q_0 \leftarrow \Psi)$ and those universally quantified inside Ψ .

The possibility of getting rid of skolem constants was somehow already clear in the proofs of the correctness of the old version of SLDNFA [12]. The proof of the completeness of SLDNFA in [12] already relied on an explicit substitution of skolem constants by variables. There is a well-known theorem in classical logic which states a strong relationship between skolem constants and universally quantified variables: for a given theory T , formula $F[X]$ and constant sk which appears neither in T nor in F , it holds that $T \models F[sk]$ iff $T \models \forall X.F[X]$. For the old version of SLDNFA, it means that deriving the formula $\forall \overline{X}.Q_0[sk] \leftarrow \Psi[sk]$ containing the skolem constant sk , is equivalent with deriving $\forall Z, \overline{X}.Q_0[Z] \leftarrow \Psi[Z]$. Though from a theoretical point of view, the use of variables rather than skolem constants is not so unexpected, it is a significant and useful improvement. We were not the first one to work out this solution; in the process of deskolemising SLDNFA, we found out that [25] had already seen this possibility and had exploited it in another abductive procedure (published as [26]). In sections 4 and 11.5, we will discuss similarities and differences between his solution and ours.

The use of variables gives considerable advantages over the use of skolem constants. First, it becomes possible to extract much more interesting and more general forms of abductive answers from an SLDNFA-refutation. In [13], the only answer derived from an SLDNFA-refutation was a set of ground abductive atoms and a variable substitution both in which skolem constants appear. [25, 26] derives from one derivation many ground answers by replacing certain variables in abduced atoms by arbitrary constants that satisfy certain disequalities generated during the derivation. In the new version of SLDNFA, this type of answer is further generalised; we extract non-ground FOL formulas as abductive solutions which formulate much more general conditions under which the query holds. Section 6 is devoted to a discussion of the answers that can be derived from an SLDNFA-refutation.

Second, it turns out that our techniques for solving the floundering abduction problem (using variables) allow also to provide a partial solution for the floundering negation problem. In [22], an abductive interpretation of negation in logic programs was proposed. In this view, negative calls are seen as a special kind of abductive calls. As observed in [31], this entails that a solution for the floundering negation problem also provides a solution for the floundering abduction problem. Our work shows that a full solution for the floundering abduction problem can be given which only partially solves

the floundering negation problem.

2 Preliminaries

In this section, we introduce the preliminaries for defining SLDNFA, including the 3-valued completion semantics. We assume some familiarity with concepts of logic programming. We follow mostly [38].

An alphabet Σ is a tuple consisting of the usual set of punctuation symbols and connectives, an infinite set of variables, a set Σ^p of predicate symbols and a set Σ^f of functors (including constants). Σ^p includes $=$. Terms, atoms, literals and formulae based on Σ are defined as usual. As usual, a free variable in a formula is a variable not bound by a quantifier; an open formula contains free variables; a closed or ground formula does not. Terms are denoted by t, s ; variables by capitals X, Y, Z ; domain elements by x, y, z . Tuples of terms, variables are denoted \bar{t}, \bar{X} . A p -atom is an atom of the form $p(\bar{t})$. Given a term t , $var(t)$ denotes the set of variables in t ; $var(F)$ denotes the set of free variables in F . The Herbrand universe of Σ is the set of all ground terms based on Σ . The set is denoted by $HU(\Sigma)$ or HU when Σ is obvious from the context.

An equality set E based on Σ is a finite set $\{s_1 = t_1, \dots, s_n = t_n\}$ where s_i, t_i are terms based on Σ . When E appears in a formula, then it should be interpreted as the conjunction $s_1 = t_1 \wedge \dots \wedge s_n = t_n$ of atoms. An equality set is in solved form iff it is a set of equality atoms of the form $X = t$ such that X is a variable different from t and X occurs only once at the left and not at the right. We define a substitution as a special type of equality set: it is a set of atoms of the form $X = t$ with X a variable appearing only once at the left. According to this definition, an equality set in solved form is an *idempotent* substitution and vice versa. Substitutions are denoted $E, \theta, \sigma, \delta$. The empty substitution is denoted ε . The domain $dom(\theta)$ and range $ran(\theta)$ of a substitution θ are defined as usual. θ is a ground substitution iff $ran(\theta)$ consists of ground terms. Given a set of variables V , $\theta|_V$ denotes the substitution $\{X = t | X = t \in \theta \wedge X \in V\}$. The application of a substitution θ on a term t or atom or sets of these is defined as in [38], but denoted $\theta(t)$. The composition of substitutions θ, σ is defined as in [38] but is denoted as $\sigma\theta$. Recall that $\sigma\theta$ is obtained from $\{X = \sigma(t) | X = t \in \theta\} \cup \{X = t | X = t \in \sigma \wedge X \notin dom(\theta)\}$ by dropping all tuples $X = X$. As proven in [38], for any term t , $\sigma\theta(t) = \sigma(\theta(t))$. Renaming substitutions and variants of terms, atoms and formulas are defined as in [38]. In this paper, substitutions are virtually always idempotent. Normally, the composition of two idempotent substitutions θ, σ is not idempotent. However, $\sigma\theta$ is idempotent if $dom(\theta) \cap dom(\sigma) = \{\}$.

For a pair of terms t, s , we define that t is *more general than* s iff there exists a substitution θ such that $\theta(t) = s$. Concepts as the relation "*.. is more general than ..*" between substitutions, unifier and most general unifier (mgu) of sets of terms and equality sets are defined as in [38]. A unifier θ of a set of equality atoms E is relevant iff all variables in θ appear in E . In the context of the paper, the following property is important: an idempotent mgu θ of an equality set E is relevant wrt E . For a proof we refer to [35].

Given an open formula F , its existential projection on a set of variables S is the formula $\exists \bar{X}. F$ with \bar{X} the free variables of F not appearing in S . This formula will be

denoted $\exists|_S(F)$. Likewise for the universal projection on S . The universal (existential) closure of F is the universal (existential) projection of F on $\{\}$; it is denoted $\forall(F)$ (respectively $\exists(F)$).

A program clause based on an alphabet Σ is a formula of the form:

$$A \leftarrow L_1, \dots, L_n$$

where A is an atom and L_1, \dots, L_n ($n \geq 0$) are literals based on Σ . A query or goal is of the form:

$$\leftarrow L_1, \dots, L_n$$

where L_1, \dots, L_n are literals. Below, we often denote queries of the above form as $\leftarrow Q$. The formula $\leftarrow Q$ is interpreted as the open disjunction $\neg L_1 \vee \dots \vee \neg L_n$. Variables are *not* universally quantified! Q is interpreted as the open conjunction $L_1 \wedge \dots \wedge L_n$. In the sequel, we usually write $\leftarrow L, Q$ to express that the literal L occurs inside the query. This does not imply that L occurs as first literal. Similarly, when B, B' denote conjunctions of literals, we will write $\leftarrow B, B'$, to denote the query consisting of the literals of B and B' .

An abductive logic program P^A based on Σ is a pair of a set P of program clauses based on Σ and a subset A of predicates of Σ^p , called *abducible*. The other predicates in $\Sigma^p \setminus A$ are called *non-abducible*. The *definition* of a non-abducible predicate p in P^A , denoted $Def(p, P^A)$ is the set of program clauses with p in the head. This set may be empty. For reasons of elegance, we assume without loss of generality that *for any non-propositional abductive logic program P^A , $Def(=, P^A) = \{X = X\}$* . Or, $=$ is defined by the reflexivity atom.

The interpretations that are used in the semantics are 3-valued and non-Herbrand. A 3-valued interpretation maps predicate symbols to 3-valued relations. A 3-valued relation on a domain D can be seen as a function from D^n to $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$. These values are ordered as follows $\mathbf{f} < \mathbf{u} < \mathbf{t}$. Each truth value has an inverse truth value: we define $\mathbf{f}^{-1} = \mathbf{t}$, $\mathbf{t}^{-1} = \mathbf{f}$, $\mathbf{u}^{-1} = \mathbf{u}$.

Below, given an alphabet Σ and domain D , a D -term or D -formula is a term or formula based on the alphabet Σ_D , which extends Σ by adding the elements of D to the set of constants.

Definition 2.1 *A Σ -interpretation I is a tuple $(D_I, \mathcal{F}_I, \mathcal{H}_I)$ consisting of a domain D_I , a correspondence \mathcal{F}_I which maps D_I -terms $f(x_1, \dots, x_n)$ to D_I , for any n -ary functor f of Σ^f and tuple (x_1, \dots, x_n) of D_I^n and a truth function \mathcal{H}_I which maps D_I -atoms $p(x_1, \dots, x_n)$ to the set $\{\mathbf{f}, \mathbf{u}, \mathbf{t}\}$, for any n -ary predicate p of Σ^p and tuple (x_1, \dots, x_n) of D_I^n .*

In addition, we require that the "=" predicate is interpreted as the identity relation on D_I .

Given a Σ -interpretation I , a variable assignment for I is a partial function from the variables of Σ to elements of D_I . Variable assignments are denoted by V , or as sets of tuples X/x with X a variable and x a domain element. For any open formula (or term) F , $V(F)$ denotes the D_I -formula (or D_I -term), obtained by substituting $V(X)$ in F for all free occurrences of variables X for which $V(X)$ is defined.

When Σ is clear from the context, we may drop Σ as a prefix. The above definition differs slightly from, but is equivalent with the normal one which defines an interpretation as a tuple of domain, a correspondence between functors and functions and a correspondence between predicate symbols and (2- or 3-valued) relations. We make the difference transparent by defining $I(f)$ as the function $D_I^n \rightarrow D_I : (x_1, \dots, x_n) \rightarrow \mathcal{F}_I(f(x_1, \dots, x_n))$ and by defining $I(p)$ as the relation $D_I^n \rightarrow \{\mathbf{f}, \mathbf{u}, \mathbf{t}\} : (x_1, \dots, x_n) \rightarrow \mathcal{H}_I(p(x_1, \dots, x_n))$.

A Σ -interpretation I is 2-valued on a predicate p if \mathbf{u} is not in the range of $I(p)$. I is 2-valued if I is 2-valued on all predicates. Another helpful concept is the projection of a Σ -interpretation on a subalphabet Σ' . Let I be a Σ -interpretation. The projection $I|_{\Sigma'}$ of I on Σ' is the Σ' -interpretation $(D_I, \mathcal{F}'_I, \mathcal{H}'_I)$ with $\mathcal{F}'_I, \mathcal{H}'_I$ the restriction of $\mathcal{F}_I, \mathcal{H}_I$ to the functors and predicate symbols of Σ' .

Given a Σ -interpretation I , any ground D_I -term denotes a unique domain element. The mapping of ground terms to their denoted domain elements is formalised by extending \mathcal{F}_I to all ground D_I -terms. The extension is defined by induction on the depth of the term:

- for any domain element x : $\mathcal{F}_I(x) = x$
- for any n -ary functor $f \in \Sigma^f$ ($n \geq 0$) and ground D_I -terms t_1, \dots, t_n :

$$\mathcal{F}_I(f(t_1, \dots, t_n)) = \mathcal{F}_I(f(\mathcal{F}_I(t_1), \dots, \mathcal{F}_I(t_n)))$$

A Herbrand interpretation M is defined as usual, as an interpretation with domain $HU(\Sigma)$ and with \mathcal{F}_M the identity function on $HU(\Sigma)$. A Herbrand pre-interpretation of Σ is a Herbrand interpretation of the alphabet consisting of all symbols of Σ except the non-equality predicates.

The truth function \mathcal{H}_I associated to some interpretation I can be extended to all ground D_I -formulas. \mathcal{H}_I is defined by induction on the depth of the formulas, as given in the following table:

$$\begin{aligned} \mathcal{H}_I(p(t_1, \dots, t_n)) &= \mathcal{H}_I(p(\mathcal{F}_I(t_1), \dots, \mathcal{F}_I(t_n))) \\ \mathcal{H}_I(\neg F) &= \mathcal{H}_I(F)^{-1} \\ \mathcal{H}_I(F_1 \vee F_2) &= \max\{\mathcal{H}_I(F_1), \mathcal{H}_I(F_2)\} \\ \mathcal{H}_I(F_1 \wedge F_2) &= \min\{\mathcal{H}_I(F_1), \mathcal{H}_I(F_2)\} \\ \mathcal{H}_I(\forall X : F) &= \min\{\mathcal{H}_I(\{X/x\}(F)) \mid x \in D_I\} \\ \mathcal{H}_I(\exists X : F) &= \max\{\mathcal{H}_I(\{X/x\}(F)) \mid x \in D_I\} \\ \mathcal{H}_I(F_1 \leftarrow F_2) &= (\mathcal{H}_I(F_1) \geq \mathcal{H}_I(F_2)) \\ \mathcal{H}_I(F_1 \leftrightarrow F_2) &= (\mathcal{H}_I(F_1) = \mathcal{H}_I(F_2)) \end{aligned}$$

Here, $(\mathcal{H}_I(F_1) \geq \mathcal{H}_I(F_2))$ denotes \mathbf{t} if $\mathcal{H}_I(F_1) \geq \mathcal{H}_I(F_2)$ and \mathbf{f} otherwise; analogously for $(\mathcal{H}_I(F_1) = \mathcal{H}_I(F_2))$.

The truth tables used for \leftrightarrow and \leftarrow are the ones suggested by Lukasiewicz (see [33]) and are used also in [24] and [34]. Note that $F_1 \leftrightarrow F_2$ is defined true if F_1 and F_2 have the same truth value, otherwise the formula is defined false. Also $F_1 \leftrightarrow F_2$ is equivalent with $F_1 \leftarrow F_2 \wedge F_1 \rightarrow F_2$ but not with $F_1 \wedge F_2 \vee \neg F_1 \wedge \neg F_2$.

Given a ground D_I -formula F , $I \models F$ denotes that $\mathcal{H}_I(F) = \mathbf{t}$. Given a theory \mathcal{T} of ground formulas, $I \models \mathcal{T}$ denotes that $\forall F \in \mathcal{T} : I \models F$. A model theory induces an

entailment relation. Given a theory \mathcal{T} and formula F based on Σ , $\mathcal{T} \models F$ iff for any Σ -interpretation M such that $M \models \mathcal{T}$, $M \models F$.

Using the above concepts, we define the 3-valued completion semantics for abductive logic programs. This semantics was first defined in [14] and extends the 3-valued completion semantics for non-abductive logic programs of [34] to abductive programs and the 2-valued completion semantics for abductive logic programs of [8] with 3-valued models. The semantics of an abductive logic program P^A based on Σ under 3-valued completion semantics is expressed via three theories.

One theory is the Free Equality theory ($FEQ(\Sigma)$), also called Clark's Equality theory [7]. Note that because $=$ is interpreted as the identity, the classical axioms of equality (reflexivity, asymmetry, transitivity, substitution) are always satisfied. $FEQ(\Sigma)$ contains the following axioms:

For each n-ary functor $f \in \Sigma^f$ and for each $i, 1 \leq i \leq n$:

$$\forall \overline{X}, \overline{Y} : f(\overline{X}) = f(\overline{Y}) \rightarrow X_i = Y_i$$

For each pair of different functors $f, g \in \Sigma^f$:

$$\forall \overline{X}, \overline{Y} : \neg g(\overline{X}) = f(\overline{Y})$$

For each term t which contains a variable X (occur check axioms):

$$\forall (\neg t = X)$$

The second theory $Comp(P^A)$ is the set of *completed definitions* of the non-abducible predicates of P^A . The construction of the completed definition of a predicate p is as in [7]. Assume that $Def(p, P^A)$ consists of n program clauses of the form $p(\overline{t}_i) \leftarrow B_i$. Assume that \overline{X}_i is a tuple containing all variables of $p(\overline{t}_i) \leftarrow B_i$. The completed definition $CompDef(p, P^A)$ of p wrt P^A is defined as follows:

$$\forall \overline{X} : p(\overline{X}) \leftrightarrow (\exists \overline{X}_1. \overline{X} = \overline{t}_1 \wedge B_1) \vee \dots \vee (\exists \overline{X}_n. \overline{X} = \overline{t}_n \wedge B_n)$$

where \overline{X} is a tuple of fresh variables. Note that if $Def(p, P^A)$ is empty then $CompDef(p, P^A) = \forall \overline{X}. \neg p(\overline{X})$. Define $Comp(P^A) = \{CompDef(p, P^A) | p \text{ is non-abducible}\}$.

The third theory is the set of axioms $Abd2(P^A) = \{\forall \overline{X}. p(\overline{X}) \vee \neg p(\overline{X}) | p \in A\}$. These axioms formulate the law of excluded middle for the abducible predicates. A Σ -interpretation I satisfies $Abd2(P^A)$ iff I is 2-valued on all abducible predicates.

Using the above definitions, the 3-valued semantics can be defined as follows. Given is an abductive logic program P^A based on Σ .

Definition 2.2 A Σ -interpretation M is a model of P^A (under 3-valued completion semantics) iff $M \models FEQ(\Sigma) \cup Comp(P^A) \cup Abd2(P^A)$.

We write $P^A \models_{\Sigma} F$ to denote that $FEQ(\Sigma) \cup Comp(P^A) \cup Abd2(P^A) \models F$. We write $\models_{\Sigma} F$ to denote that $FEQ(\Sigma) \models F$.

The 3-valued completion semantics was proposed in [14] and [11]. Like most semantics for abductive logic programs, 3-valued completion semantics assigns a 2-valued interpretation to abducible predicates. This is also the case in the 2-valued completion semantics [8], the 2-valued generalised stable semantics [32], and the 3-valued extended well-founded semantics [49]. The reason for this is explained in [11] and can be traced back to a well-known argument formulated by Moore in [47]. He argues that reasoning by 2 cases (i.e. something is either true or false) is crucial for reasoning

on uncertainty, and is one of the crucial advantages of (classical) logic compared to other knowledge representation formalisms. Reasoning by 2 cases on a predicate is only possible when the law of excluded middle holds for it, or equivalently, when it has a 2-valued interpretation. That abductive logic programming was successfully applied for representing uncertainty, has to do with the fact that reasoning by 2 cases on abducible predicates is possible. An important illustration of this thesis is found in the recent experiments on the transformation of the language \mathcal{A} to extensions. In [17], we compare the transformation of \mathcal{A} to Extended Logic Programming (ELP), presented in [27] with a transformation to abductive logic programming. The latter transformation is in all respects superior to the first one. In [17] we show that the reason for the incompleteness of the transformation of [27] is due to the fact that in ELP the law of excluded middle does not hold.

One could argue then that also the non-abducible predicates should have a 2-valued interpretation. However, the use of a third truth value for non-abducible predicates has a very specific role which we explain in [11]. Here, we can only summarize the main argument. [11] proposes to interpret an abductive logic program as a set of definitions for the non-abducible predicates. Due to the generality of the formalism, one can easily construct senseless definitions. Consider the following definition for p :

$$\{ p \leftarrow \neg p \}$$

Under a 2-valued completion semantics, an abductive logic program containing such definition would be inconsistent. The use of \mathbf{u} is a more permissive solution to deal with badly constructed definitions. In 3-valued completion semantics, such a program remains consistent; yet the badly defined facts will have truth value \mathbf{u} in all or a subclass of the models. This shows that \mathbf{u} is an error condition and not a truth value. \mathbf{u} should be interpreted as *badly defined*. It is only on badly defined non-abducible facts, that reasoning by 2 cases is impossible. A 3-valued interpretation for an abducible predicate would not make sense under this interpretation of \mathbf{u} : abducible predicates have no definition and cannot be badly defined.

Two theorems about 3-valued completion semantics are important in the context of this paper. The first is about the consistency of abductive logic programs. It was proven in [14, 10].

Theorem 2.1 *Let P^A be an abductive logic program based on Σ . Let I be any Σ_{abd} -interpretation where Σ_{abd} is obtained from Σ by dropping all non-abducible predicates except $=$. Assume $I \models \text{FEQ}(\Sigma) \cup \text{Abd2}(P^A)$.*

There exists a Σ -model M of P^A such that $M|_{\Sigma_{abd}} = I$.

This proves the consistency of any abductive logic program, since any 2-valued Herbrand Σ_{abd} -interpretation satisfies $\text{FEQ}(\Sigma) \cup \text{Abd2}(P^A)$.

A second theorem about the 3-valued completion semantics is that it is the weakest semantics known for abductive logic programming. In [14, 10], the following theorem is proven:

Theorem 2.2 *If M is a Σ -model of P^A wrt (2-valued completion semantics [8]) (generalised stable semantics [32]) (generalised well-founded semantics [49]) (justification semantics [14, 10]) then M is a Σ -model of P^A wrt 3-valued completion semantics.*

As a consequence, the 3-valued completion semantics induces the weakest entailment relation: if an abductive logic program entails F according to the 3-valued completion semantics then also wrt to the other semantics. In intuitive terms: the declarative meaning of an abductive logic program under the 3-valued completion semantics is a safe approximation of its meaning under these other semantics. With a caveat, it follows that SLDNFA will be a correct reasoning procedure with respect to the other semantics. We come back to this and to the caveat in section 11.2.

3 Abductive answers to queries

In abductive logic programming, an abductive answer is commonly defined as a set of ground atoms which imply the initial goal. The formal definition below extends the classical definition.

Given is an abductive logic program P^A and initial goal $\leftarrow Q_0$ based on Σ .

Definition 3.1 *A ground abductive solution for $\leftarrow Q_0$ is a triple $(\Sigma', \Delta, \theta)$ with Δ a finite set of ground abducible atoms, θ a substitution of the variables of Q_0 both based on the extension Σ' of Σ with constants or functors. Moreover,*

$$P + \Delta \models_{\Sigma'} \forall(\theta(Q_0))$$

$P + \Delta$ denotes the logic program $(P \cup \Delta)^{\exists}$. Note that by theorem 2.1, $P + \Delta$ is consistent.

The reason for allowing ground abductive solutions based on extensions of Σ is to be able to reason abductively on an *open domain*, i.e. to reason in case of uncertainty on the elements of the domain. As a trivial example consider the abductive logic program $P_0^{\{r\}}$ with P_0 the set:

$$\begin{aligned} p &\leftarrow r(X), \neg q(X) \\ q(a) \\ X &= X \end{aligned}$$

In the sequel, the program clause $X = X$ will be implicit.

It is easily verified that the query $\leftarrow p$ has no ground abductive solution based on the alphabet Σ consisting of the symbols of the program. However, $(\Sigma \cup \{b\}, \{r(b)\}, \varepsilon)$ is a ground abductive solution. Clearly, $P_0 + \{r(b)\} \models_{\Sigma \cup \{b\}} p$. An abduced atom represents a hypothesis about the abducible predicate; analogously, an extended alphabet with new constants or functors is an abductive hypothesis that there exist additional elements in the domain of discourse.

An application where reasoning in an open domain is natural is planning in the event calculus. The events which change the initial state in the desired goal state are subject of the search, and therefore a priori unknown. A ground abductive solution for the query representing the desired goal state, is a plan stating the existence of certain events represented by -new- constants, and describing their order and what actions occur at each event.

Often, a goal has an infinite number of ground abductive solutions. In such cases it may be possible and useful to return a more general abductive solution which characterises a class of ground abductive solutions.

Given is again an abductive logic program P^A and initial goal $\leftarrow Q_0$ based on Σ .

Definition 3.2 An abductive solution for $\leftarrow Q_0$ wrt P^A is an open formula Ψ containing only equality and abducible predicates such that:

- $P^A \models_{\Sigma} \forall(Q_0 \leftarrow \Psi)$
- $\exists(\Psi)$ is satisfiable wrt P^A .

Obviously, a ground abductive solution can be considered as a special case of an abductive answer: $P + \Delta \models_{\Sigma'} \forall(\theta(Q_0))$ is equivalent with $P^A \models_{\Sigma} \forall(Q_0 \leftarrow \theta \wedge \text{Comp}(\Delta^D) \wedge \text{FEQ}(\Sigma') \setminus \text{FEQ}(\Sigma))$. Here, D is the set of non-abducible predicates; hence $\text{Comp}(\Delta^D)$ is the set of completed definitions of all abducible predicates in Δ .

The two above forms of solutions provide two different compromises between comprehensibility and generality: an *abductive solution* may be more general but also more complex than a ground abductive solution.

While the next two sections describe how SLDNF can be extended for abductive logic programs, section 6 describes in detail how from an abductive SLDNFA-refutation abductive and ground abductive solutions can be extracted.

4 Basic computation steps in SLDNFA

As was shown by Clark [7], an SLDNF computation can be understood as a process of deriving formulas $\forall(Q_0 \leftarrow \Psi)$, with $\leftarrow Q_0$ the initial query and the conjunction Ψ composed of the unsolved goals. For SLDNFA, a similar interpretation can be made. The formula Ψ contains two types of conjuncts:

- For any goal $\leftarrow Q$ for which a derivation still needs to be computed -in the sequel, a *positive goal*-, Ψ contains the open formula Q (denoting the open conjunction of literals).
- For any goal $\leftarrow Q$ for which a failure tree still needs to be constructed -in the sequel, a *negative goal*-, Ψ contains the open formula $\forall \overline{X}. \leftarrow Q$ with \overline{X} a subset of the variables of Q .

SLDNFA selects *unsolved* positive or negative goals $\leftarrow Q$ and literals in $\leftarrow Q$ and rewrites these goals depending on the sort of selection. These rewrite operations can be interpreted as theorem proving steps on Ψ . Resolution is applied on non-abducible atoms selected in positive and negative goals, as in SLDNF. Negative literals are deleted from positive goals and added as negative goal and vice versa, just as in SLDNF. Abducible atoms in positive goals are never selected: they are treated as residual atoms. They will be called *abduced atoms*. The process of computing an increasing set of residual abducible atoms can be understood as the incremental construction of a definition for the abducible predicates. Abducible atoms selected in negative goals are resolved with the residual atoms in the positive goals. It is the latter operation that creates the most serious complications compared with SLDNF.

Note that Ψ contains two types of variables: free variables (universally quantified in front of $\forall(Q_0 \leftarrow \Psi)$) and variables, universally quantified in a conjunct of Ψ . This distinction will play a crucial role in SLDNFA. Below, we call the variables which

appear free in Ψ *positive* and the variables which appear universally quantified in Ψ *negative*. Whether variables in goals in a derivation are positive or negative depends on the way they are introduced in the derivation. The positive variables (free in Ψ) are either the variables of the initial goal Q_0 or are the variables of input program clauses used for resolution with positive goals. The negative variables (universally quantified in Ψ) are the variables of program clauses used for resolution with negative goals. In SLDNF the same kind of distinction can be made. However, due to the safety of the selection in SLDNF, the two types of variables never occur in the same context, in the same atom. In SLDNFA, due to the resolution of non-ground abducible atoms in negative goals with non-ground abduced atoms, positive variables may be transmitted into negative goals and may occur together with negative variables. The problem is that these two types of variables need to be treated differently. Actually, positive variables in negative goals should be treated in the same way as skolem constants are dealt with in [13].

Compared with SLDNF, two technical problems arise. The first one is that at the moment that an abducible p -atom is selected in a negative goal, then in general, the set of residual abduced p -atoms in positive goals is, as yet, only partially computed. Hence, at the time it is selected, it will be impossible to compute all resolvents needed to obtain a complete failed tree.

This problem is simple to solve: the failure tree below a negative goal in which an abducible p -atom is selected, will be constructed incrementally: when new p -atoms are *abduced* in positive goals, new failure branches will be added to the negative goal. We illustrate this strategy with an example.

Consider the abductive logic program $P_1^{\{r\}}$, with P_1 the set:

$$\begin{aligned} q &\leftarrow r(X), \neg p(X) \\ p(X) &\leftarrow r(b) \end{aligned}$$

The initial query is $\leftarrow \neg q, r(a)$. An SLDNFA-refutation is shown in figure 1. As a notational convenience, we prefix positive goals with $+$ and negative goals with $-$. Positive variables are indexed with $+$, negative variables are indexed with $-$. The arcs are numbered to indicate the sequence of computation steps. In the first step, the selection of the negative literal $\neg q$ results in a positive goal $\leftarrow r(a)$ and a negative goal $\leftarrow q$. In step (2), the negative goal is resolved with the unique clause of q . In step (3), the atom $r(X)$ in the negative goal is resolved with the residual atom $r(a)$ in the positive goal. Step (4) switches the negative goal $\leftarrow \neg p(a)$ into the positive goal $\leftarrow p(a)$, which is then resolved (step(5)). At this point, the atom $r(b)$ is abduced. As a consequence, the failure tree below $\leftarrow r(X), \neg p(X)$ has become incomplete and must be extended. To this end, $r(b)$ is resolved with the negative goal (step (6)). Steps (7) and (8) are analogous to steps (4) and (5). An abductive answer that can be derived from this refutation is the formula $\forall X.r(X) \leftrightarrow X = a \vee X = b$. A ground abductive answer is the tuple $(\Sigma, \{r(a), r(b)\}, \varepsilon)$.

The second technical problem is more serious and is related to the interaction of positive and negative variables in negative goals. The following example illustrates the problem. Consider the abductive logic program $P_2^{\{r\}}$, with P_2 the set:

$$q \leftarrow r(a, V, W), Q[V, W]$$

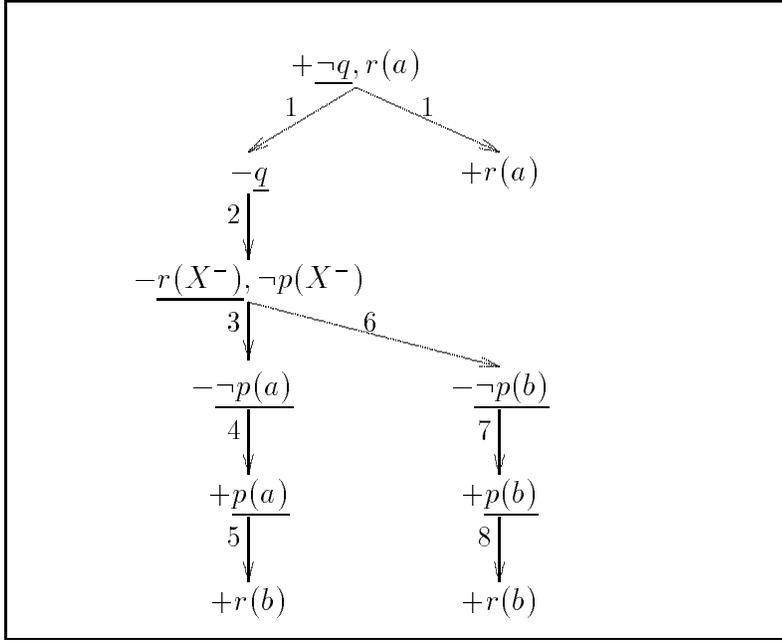


Figure 1: SLDNFA-refutation of $\leftarrow \neg q, r(a)$

where $Q[V, W]$ represents any conjunction of literals with V, W as free variables. A partial SLDNFA-derivation for $\leftarrow \neg q, r(X, b, Y)$ is shown in figure 2. In step (1), selection of $\neg q$ results in an positive goal with residual abduced atom $r(X^+, b, Y^+)$ and the negative goal $\leftarrow q$. The resolution step (2) produces the negative goal $\leftarrow r(a, V^-, W^-), Q[V^-, W^-]$.

The formula $\forall(Q_0 \leftarrow \Psi)$ that has been derived in this derivation is:

$$\forall X^+, Y^+. \neg q \wedge r(X^+, b, Y^+) \leftarrow (\forall V^-, W^-. \leftarrow r(a, V^-, W^-), Q[V^-, W^-]) \wedge r(X^+, b, Y^+)$$

The question now is how to resolve the abduced atom $r(X^+, b, Y^+)$ and the selected atom $r(a, V^-, W^-)$. These atoms have a most general unifier $\theta = \{X^+ = a, V^- = b, W^- = Y^+\}$. If classical resolution would be applied, then X^+ would be assigned the value a and the negative goal $\leftarrow Q[b, Y^+]$ would be derived. Applying θ on the abduced atom would produce $r(a, b, Y^+)$. However, note that in this way, an important class of ground abductive answers is lost, namely those answers where X^+ is different from a and where $\forall V^-, W^-. \neg r(a, V^-, W^-)$ holds.

SLDNFA keeps all solutions, those in which X^+ is not a and those in which X^+ is a and $\leftarrow Q[b, Y^+]$ fails, by adding the atom $X^+ = a$ as a residual equality atom to the negative goal obtained by resolution of $\leftarrow r(a, V^-, W^-), Q[V^-, W^-]$ and $r(X^+, b, Y^+)$. The resulting goal $\leftarrow X^+ = a, Q[b, Y^+]$ is displayed in figure 3. The formula $\forall(Q_0 \leftarrow \Psi)$ corresponding to this derivation is:

$$\forall X, Y. \neg q \wedge r(X, b, Y) \leftarrow r(X, b, Y) \wedge (\forall V, W. \leftarrow r(a, V, W), Q(V, W)) \wedge (\leftarrow X = a, Q[b, Y])$$

From this formulas, a formula $\forall(Q_0 \leftarrow \Psi')$ can be derived such that Ψ' contains only abducible and equality literals:

$$\forall X, Y. \neg q \wedge r(X, b, Y) \leftarrow r(X, b, Y) \wedge (\forall V, W. \neg r(a, V, W)) \wedge X \neq a$$

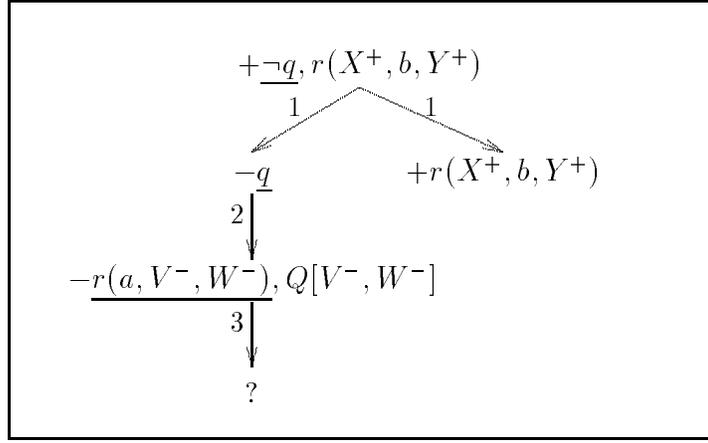


Figure 2: Partial derivation of $\leftarrow \neg q, r(X, b, Y)$

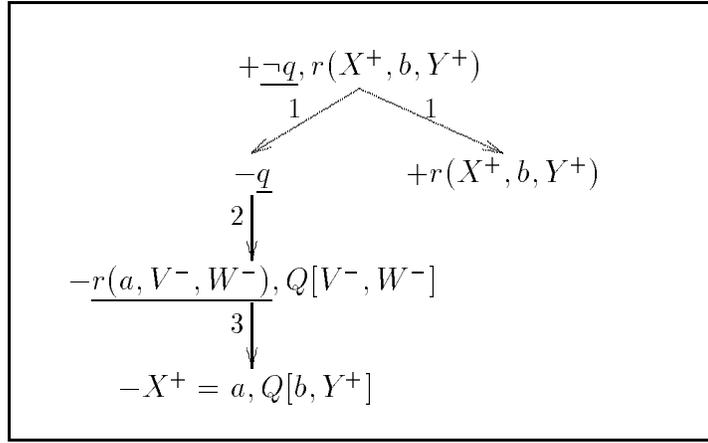


Figure 3: SLDNFA-refutation of $\leftarrow \neg q, r(X, b, Y)$

Ψ' is an abductive answer and represents all ground abductive answers in which $X \neq a$. In general, it is always possible to extract an abductive answer from a derivation of the kind in figure 3, in which positive goals contain only abduced atoms and in which each negative goal has a complete failure tree or contains an equality atom of the kind $X^+ = t$ with t neither X^+ nor a negative variable. Therefore, such a derivation is defined to be an SLDNFA-refutation.

We now give a precise description of how the basic operations of unification and resolution are modified to take the difference between positive and negative variables into account. We assume that both operations get as input terms, equality sets, atoms, goals or program clauses in which all variables are known to be positive (occurring free in Ψ) or negative (universally quantified in Ψ). Below, the concept of marked variables is formally defined.

Definition 4.1 *A marking α is a partial function of the set of variables of an alphabet Σ to the set $\{+, -\}$. Given a marking α , a variable X is marked iff $\alpha(X)$ is defined.*

A marked atom, equality set, goal, program clause, is one in which all variables are marked. The marking α' obtained from α by marking some set of variables V positive is defined as follows:

- $\alpha'(X) = \alpha(X)$ iff $\alpha(X)$ is defined and $X \notin V$
- $\alpha'(X) = +$ iff $X \in V$

The marking α' obtained from α by marking some set of variables V negative is defined analogously.

Hence, the denotation X^+ in a term, means that $\alpha(X) = +$ with respect to the given marking α .

Unification and resolution get as input a marking α of variables, and terms, equality sets, goals, program clauses which are marked under α . We stress that this marking is not an ad hoc feature; it is just a memo to inform unification and resolution in SLDNFA about the logical nature of the variables.

Definition 4.2 *Given a marking α , a marked equality set is in positive solved form iff it is in solved form and it contains no atoms of the form $X^+ = Y^-$.*

Recall from section 2, that equality sets in solved form are idempotent substitutions and vice versa.

Definition 4.3 *An equality set E_s is a (positive) solved form of an equality set E iff E_s is in (positive) solved form and E_s is an mgu of E . An equality set with a solved form is called solvable; an equality set without solved form is called unsolvable. Two atoms $p(\bar{t}), p(\bar{s})$ are said to be unifiable iff $\{\bar{t} = \bar{s}\}$ is solvable.*

Here $\bar{t} = \bar{s}$ denotes the equality set $\{t_i = s_i | 1 \leq i \leq n\}$. Recall from section 2 that E_s is relevant wrt E .

It is straightforward that -given some marking α - a marked equality set has a solved form iff it has a positive solved form. Indeed, a positive solved form is a solved form and vice versa, a solved form can be turned easily into a positive solved form by applying a renaming which maps atoms $X^+ = Y^-$ into $Y^- = X^+$. A formal proof is in proposition A.1. As a consequence, a correct unification algorithm (e.g. [42]) can be extended to an algorithm computing a positive solved form.

Using the notion of positive solved form, we define positive and negative resolution between a marked query $\leftarrow Q$ and a marked program clause or atom C . C may be both a clause of the abductive logic program or an abduced atom. In the rest of this section, we assume the existence of a marking α . The concept below corresponds to classical resolution.

Definition 4.4 (positive resolution) *Let $\leftarrow Q$ be a marked goal $\leftarrow p(\bar{t}), B$ and $C \equiv p(\bar{s}) \leftarrow B'$ a marked clause, possibly atomic (B' the empty conjunction).*

$\leftarrow Q'$ is derived from Q and C by positive resolution on $p(\bar{t})$ using θ if the following holds:

- θ is a solved form of $\bar{t} = \bar{s}$

- $\leftarrow Q'$ is the goal $\theta(\leftarrow B', B)$

We call $\leftarrow Q'$ the positive resolvent.

In SLDNFA, positive resolution can be applied both when p is non-abducible atom and C one of its clauses of the program, and when p is abducible and C is an abduced atom.

If E is a marked equality set in positive solved form, define E_+ , E_- as the subset of E of equalities having a positive variable, respectively negative variable at the left.

Definition 4.5 (negative resolution) Let $\leftarrow Q$ be a marked goal $\leftarrow p(\bar{t}), B$ and $C \equiv p(\bar{s}) \leftarrow B'$ a marked program clause or a marked abducible atom ($B = \{\}$).

$\leftarrow Q'$ is derived from $\leftarrow Q$ and C by negative resolution on $p(\bar{t})$ if the following holds:

- $\bar{t} = \bar{s}$ has a positive solved form E .
- $\leftarrow Q'$ is the goal $E_-(\leftarrow E_+, B', B)$.

We call $\leftarrow Q'$ the negative resolvent.

Definition 4.6 We say that $s = t$ is irreducible when s is a positive variable and t is either a non-variable term or another positive variable.

Negative resolution will be applied to negative goals. In such a negative goal, an irreducible atom formulates a disequality constraint on a positive variable. Negative resolution will never bind a positive variable; instead it generates disequality constraints on them.

Returning to the derivation in figure 3, applying negative resolution on the negative goal $\leftarrow r(a, V^-, W^-), Q[V^-, W^-]$ and the abduced atom $r(X^+, b, Y^+)$ yields the negative goal $\leftarrow X^+ = a, Q[b, Y^+]$. This negative goal formulates the constraint that X and a should differ or else, that $Q[b, Y^+]$ should be false.

Observe also that $Q[b, Y^+]$ contains a positive variable. This shows that by negative resolution on abducible atoms, positive variables can be transmitted to negative goals. Or, even if a safe selection rule is used, positive variables are transmitted to negative goals by negative resolution on abducible atoms. One might expect therefore that if the above techniques can handle floundering abduction, then they must be able also to partially solve the floundering negation problem. This is the case, at least as far as only positive variables are allowed in selected negative literals. The following example illustrates this. Consider the logic program $P_3^{\{\}}$, with P_3 the set:

$$\begin{aligned} q(X) &\leftarrow p1(X) \\ q(X) &\leftarrow \neg p2(X) \\ p1(f(a)) &\leftarrow \\ p2(f(X)) &\leftarrow \end{aligned}$$

The initial query is $\leftarrow \neg q(X)$. An SLDNFA-refutation is given in figure 4. The answer generated by this refutation is $\neg q(X) \leftarrow \exists Y. X = f(Y) \wedge Y \neq a$. This example shows also another feature of SLDNFA. In an SLDNFA-derivation, a positive variable may appear in several branches of the tree. As a consequence, a unifier computed in one

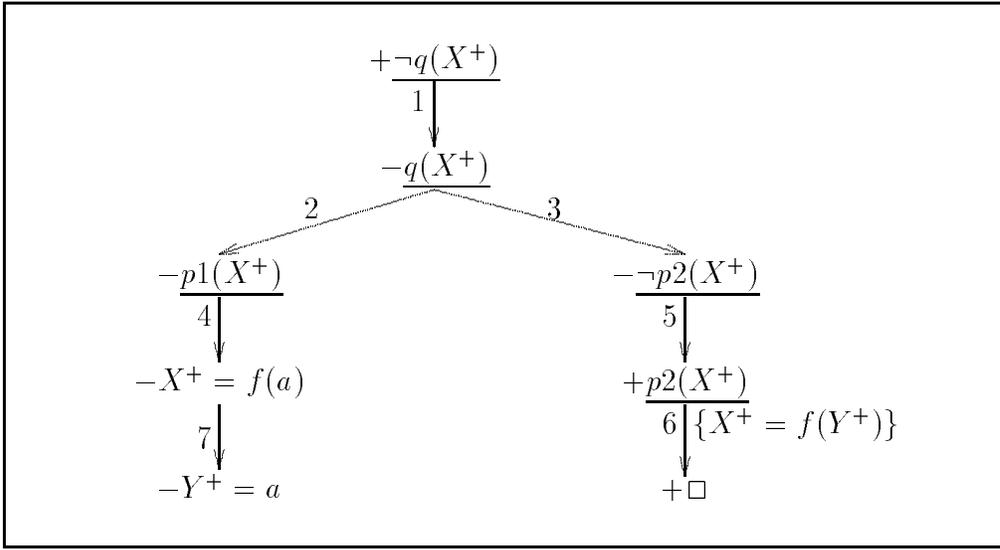


Figure 4: SLDNFA-refutation of $\leftarrow \neg q(X)$

node may have effect on all other nodes. Here this is the case with the variable X^+ . The unification $X^+ = f(Y^+)$ at step 6 turns the negative goal $\leftarrow X^+ = f(a)$ into $\leftarrow f(Y^+) = f(a)$. In step 7, this goal is negatively resolved with the reflexivity atom. This feature that a unifier can affect all nodes in a derivation, cannot occur in SLDNF due to safe selection.

The SLDNFA-refutation in figure 4 shows that the safety condition of SLDNF can be weakened: positive variables may appear in selected negative literals. On the other hand, the following example shows that SLDNFA does not offer a solution for the treatment of negative variables in positive goals. Consider the logic program P_4^{\exists} , with P_4 the set:

$$\begin{array}{l} q \leftarrow \neg p(X) \\ p(a) \end{array}$$

The initial query is $\leftarrow \neg q$. A successful SLDNFA-refutation is given in figure 5. However, P_4^{\exists} does not entail $\neg q$. In section 11, we return to this issue.

SLDNFA does not offer a full solution for floundering negation. In the definition of SLDNFA-derivation, the appearance of negative variables in positive goals is prohibited by imposing a *weak safety* condition: in a selected negative literal, only positive variables may appear.

We conclude the section with a remark on the relationship between positive and negative resolution and classical resolution.

Proposition 4.1 *For goals and program clauses which contain only positive variables, positive resolution and classical resolution coincide. For goals and program clauses which contain only negative variables, negative resolution and classical resolution coincide.*

The proof is trivial. This proposition is of importance for the relationship between SLDNFA and SLDNF in case of non-abducible logic programs P^{\exists} .

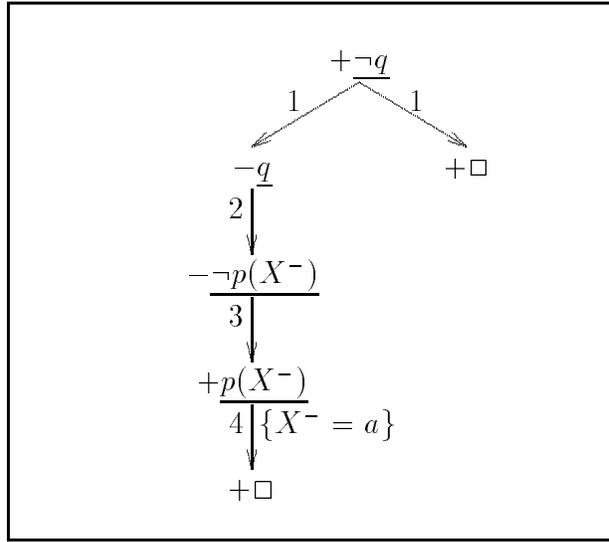


Figure 5: SLDNFA-refutation of $\leftarrow q(X)$

To end this section, we briefly discuss the differences between the basic operators of the new version of SLDNFA, those of the old SLDNFA in [13] and those of the procedure in [25, 26]. As argued in section 1, skolem constants in the old version can be interpreted as positive variables. In the old SLDNFA, there is an explicit abduction step which selects an abducible atom in a positive goal, skolemises its variables and adds the resulting atom to a set Δ . Interpreting skolem constants as positive variables, the skolemisation is a null operation and Δ corresponds to a set of residual abduced atoms. Hence, the abduction step is of no use in the new SLDNFA. Interpreting skolem constants as positive variables, equality reduction and negative resolution of the old and new SLDNFA are equivalent. If in a positive resolution step in the old version, a non-ground term t is assigned to a skolem constant by the equality reduction, then all variables in t are skolemised. This is a null operation if one interpretes skolem constants as positive variables; hence, positive resolution in the old and new SLDNFA are equivalent.

The *iff* procedure in [25, 26] is essentially an equivalence preserving rewrite procedure of FOL formulas, which operates by a generalised resolution step, namely by substituting a non-abducible atom using its completed definition and applying a number of simplification steps. Though the procedure is formalised quite differently than SLDNFA, the atomic computation steps are similar to the ones used in SLDNFA. The procedure also distinguishes between two types of variables. Where SLDNFA defines two forms of resolution both of which rely on equality reduction, the *iff* procedure uses a simple form of resolution (called unfolding) without equality reduction but uses the Martinelli-Montanari rules as rewrite rules in a sort of glassbox constraint solver. The *iff* procedure has two distinct operations in place of negative resolution, unfolding conditions of implications and propagation. In addition, other simplification rules are used to split disjunctions, etc. The combined effect of these atomic operations is very analogous to SLDNFA's operation. In section 11.5 the relationship between SLDNFA and the *iff* procedure is further investigated.

5 The SLDNFA procedure

Throughout this and the following sections, we assume the presence of an abductive logic program P^A based on an alphabet Σ . A number of preliminary concepts are defined.

Definition 5.1 *A pre-derivation K is a tuple $((\theta_1, \dots, \theta_n), T, \alpha)$ with $(\theta_1, \dots, \theta_n)$ a sequence of substitutions, T a tree of labelled goals $\leftarrow Q$ and labelled arcs and α a marking of the variables of Σ . Each goal is labelled positive or negative. A goal $\leftarrow Q$ in the tree may or may not be labeled by a literal in $\leftarrow Q$, called the selected literal.*

An arc from a negative goal $\leftarrow L, Q$ with a selected atom L arrives in a negative goal $\leftarrow Q'$ and is labeled with a program clause $H \leftarrow B$ (or an abducible atom H), called the applied resolvent. $\leftarrow Q'$ is derived from $\leftarrow L, Q$ and $H \leftarrow B$ (respectively H) by negative resolution on L .

The sequence of substitutions in a pre-derivation will be the sequence of substitutions computed at positive resolution steps. The notion of a pre-derivation serves as a kind of skeleton for the notion of an SLDNFA-derivation. In this respect, the role of a pre-derivation is similar to the notion of complex tree in [1]. Using this concept, we can define the notion of selection and standardisation apart.

Given an abductive logic program P^A and a pre-derivation K , a *resolvent* of a node N in K with a non-abducible selected atom L , is any program clause $H \leftarrow B$ of P^A such that L and a variant of H are unifiable. If the selected atom L of N is abducible then a resolvent of N is any abducible atom L' in a positive goal of K such that L and L' are unifiable. For a negative goal $\leftarrow Q$ in K , we distinguish between applied resolvents (those appearing as *applied resolvents* of arcs leaving $\leftarrow Q$ in K) and the other, *non-applied resolvents*.

Definition 5.2 (Selection) *Given is a pre-derivation K .*

A first SLDNFA-selection in K is a tuple (N, L) where N is a positive or negative goal in K without selected literal, L is a literal in N . If N is a positive goal, then L is not an abducible atom.

An SLDNFA-reselection in K is a tuple (N, C) where N is a negative goal in K labelled with a selected atom L and C is a non-applied resolvent of N (wrt K and P^A).

An SLDNFA-selection in K is a first selection or reselection in K .

In this and the following sections up to section 9, we drop the prefix SLDNFA- from SLDNFA-(re)selection.

Note that an abducible atom in a positive goal cannot be selected. The only goals that can be selected more than once are negative goals with a selected atom for which different branches of the failure tree are to be explored.

Given is a pre-derivation K .

Definition 5.3 (Safety) *A selection is safe iff it is a reselection or if it is a first selection (N, L) such that L is not a negative literal containing negative variables. A selection in K is prudent iff it is safe and if N is a positive goal and L a negative literal, then all positive variables in L occur in an abduced atom in a positive goal.*

Definition 5.4 A program clause C' is a standardised apart variant of a program clause C wrt to $K = ((\theta_1, \dots, \theta_n), T, \alpha)$ iff C' is a variant of C and the variables in C' appear neither in C , nor in $\theta_1, \dots, \theta_n$, nor in T .

Definition 5.5 Given is an abductive logic program P^A and a pre-derivation $K = ((\theta_1, \dots, \theta_n), T, \alpha)$.

Let (N, L) be a first selection in K with $N \equiv \leftarrow Q$.

An SLDNFA-extension of K using first selection (N, L) is a pre-derivation $K' = ((\theta_1, \dots, \theta_n, \theta), T', \alpha')$ such that T' is obtained from T by adding a set S with zero, one or two descendants to N , marking N with selected literal L and applying θ on all goals and labels of T . α' , θ and the set of descendants S satisfy one of the following conditions:

- Let N be a positive goal and L an atom $p(\bar{t})$ with p non-abducible.

α' is obtained from α by marking all variables of a standardised apart variant C' of a program clause $C \in P$ positive. S is a singleton containing a positive goal $\leftarrow Q'$ which is derived by positive resolution from $\leftarrow Q$ and C' using θ .

In all other cases $\theta = \varepsilon$ and $\alpha' = \alpha$. Depending on the type of selection, S satisfies the following conditions:

- Let N be a positive goal and $L = \neg A$.

S is a pair consisting of a negative goal $\leftarrow A$ and a positive goal $\leftarrow Q'$ obtained by deleting $\neg A$ in $\leftarrow Q$.

- Let N be a negative goal and $L = \neg A$.

Either S is the singleton containing one positive goal $\leftarrow A$, or S is the singleton consisting of a negative goal $\leftarrow Q'$ obtained by deleting $\neg A$ in $\leftarrow Q$.

- Let N be a negative goal and $L = p(\bar{t})$.

S is empty¹.

Let (N, C) be a reselection in K where $N \equiv \leftarrow Q$.

An SLDNFA-extension of K using reselection (N, C) is the pre-derivation $K' = ((\theta_1, \dots, \theta_n, \varepsilon), T', \alpha')$ such that T' is obtained from T by adding one new descendant N' to N and labelling the arc from N to N' with C as applied resolvent. N' is a negative goal $\leftarrow Q'$ which is derived as follows. Recall that by definition of reselection, N has a selected atom A appearing in $\leftarrow Q$.

- If A is non-abducible then α' is obtained from α by marking the variables of a standardised apart variant C' of C negatively. $\leftarrow Q'$ is derived from $\leftarrow Q$ and C' by negative resolution on A .
- If A is abducible then $\alpha' = \alpha$ and $\leftarrow Q'$ is derived from $\leftarrow Q$ and C by negative resolution on A .

Note that in 2 cases, one SLDNFA-derivation can have different SLDNFA-extensions:

¹ K' and K differ by the fact that N in K has no selected literal whereas N in K' has the selected literal L . Branches of the failure tree below N are added in later stages when N is reselected.

- when a first selection (N, L) is made with N a positive goal and L a non-abducible atom. There are as many SLDNFA-extensions as there are program clauses in P^A with a head unifiable with L .
- when a first selection $(N, \neg A)$ is made with $N \equiv \leftarrow \neg A, Q'$ a negative goal. There are two SLDNFA-extensions: one extension has the positive goal $\leftarrow A$, another extension has the negative goal $\leftarrow Q'$.

In an implementation, these situations correspond to backtracking points. In an SLDNFA-tree (an or-tree to be defined in section 9), they correspond to nodes with more than one descendant.

A difference with SLDNF is that substitutions θ computed in positive resolution steps are applied on all nodes of the tree. This is done because, as shown in figure 4, positive variables may appear in distant leaves of the tree.

Proposition 5.1 *An SLDNFA-extension of a pre-derivation is a pre-derivation.*

A pre-derivation with given selection has only a finite number of SLDNFA-extensions (modulo renaming) and they are easily computable.

Using the notion of an SLDNFA-extension of a pre-derivation, it is easy to define the notion of an SLDNFA-derivation. This concept corresponds to the notion of a pre-SLDNF-tree in [1].

Definition 5.6 *Let P^A be an abductive logic program and $\leftarrow Q_0$ a query, both based on an alphabet Σ . An SLDNFA-derivation is defined by induction on its length.*

- The tuple $((), T_0, \alpha_0)$ with T_0 a tree consisting of a single positive goal $\leftarrow Q_0$ and α_0 the marking which marks the variables of Q_0 positive, is an SLDNFA-derivation of length 0.
- Given an SLDNFA-derivation K of length n , an SLDNFA-extension of K using some safe selection in K is an SLDNFA-derivation of length $n+1$.

Proposition 5.2 *Two different goals of an SLDNFA-derivation K which do not occur in the same branch may share positive but no negative variables. Positive goals contain only positive variables. The substitutions $\theta_1, \dots, \theta_n$ of K contain only positive variables.*

If an SLDNFA-derivation K is obtained using only prudent selections, then in addition, each positive variable in a negative goal occurs in an abduced atom.

This proposition can be proven by a straightforward induction on the length of the SLDNFA-derivation; it relies on the use of safe or prudent selections.

Just as the definition of SLDNF in [43], SLDNFA-derivations are step-wise extended, unlike the definition in [1] which extends all leaves at the same time. In our case, this choice is motivated by the fact that we deliberately want to model the computation process. Our choice gives the possibility to define selection rule and fair selection rule, as in [43]. A difference between our definition and those of [43] and [1] is that failure trees are build node per node. The reason for this choice is clear: this is necessary for the treatment of selected abducible atoms in negative goals. For uniformity, it is done also for selected non-abducible atoms in negative goals.

Definition 5.7 An SLDNFA-derivation K is finitely failed if K contains a positive goal which contains a non-abducible atom without resolvent wrt P^A or if K contains the empty negative goal \square .

Definition 5.8 Given is an SLDNFA-derivation K and a negative goal N in K .

N is completed iff N has a selected literal L and either L is a negative literal or L is an atom such that each resolvent of L wrt P^A and K is an applied resolvent of N .

Definition 5.9 An SLDNFA-refutation K for a goal $\leftarrow Q$ is an SLDNFA-derivation K for $\leftarrow Q$ such that all positive leaves contain only abducible atoms and all negative goals are completed or they have no selected literal and contain an irreducible equality atom.

To illustrate the generic procedure, SLDNFA is applied to a small fault diagnosis problem. A faulty lamp problem is caused by a broken lamp or by a power failure of a circuit without backup, i.e. a loaded battery. The only circuit with a battery is $c1$; its battery is $b1$. A battery is unloaded iff one of its energy cells is dry. This is formalised in P_{lamp} :

$$\begin{aligned} & lamp(l1) \\ & battery(c1, b1) \\ & faulty_lamp \leftarrow lamp(X), broken(X) \\ & faulty_lamp \leftarrow power_failure(X), \neg backup(X) \\ & backup(X) \leftarrow battery(X, Y), \neg unloaded(Y) \\ & unloaded(X) \leftarrow dry_cell(X) \end{aligned}$$

The predicates *broken*, *power failure*, *dry cell* are abducible. An SLDNFA-refutation for the goal $\leftarrow faulty_lamp$ is presented in figure 6. We continue to use the notational conventions used in section 4 for drawing SLDNFA-derivations. The refutation terminates with abduced atom $power_failure(X^+)$ and irreducible equality atom $X^+ = c1$, representing the solution that there is a power failure on a circuit X^+ which is not $c1$. Note that the solution with circuit $c1$ and battery $b1$ with dry energy cell is implicit in this derivation: namely in the negative goal $\leftarrow X^+ = c1, \neg unloaded(b1)$. The extension SLDNFA^o of SLDNFA defined in section 10 will find this solution.

Figure 7 presents a failed SLDNFA-derivation for the query $\leftarrow \neg broken(l1), faulty_lamp$.

6 Extracting answers from an SLDNFA-refutation

A straightforward way to extract an abductive solution from an SLDNFA-refutation is by completing the abduced atoms and adding the irreducible equality atoms as constraints. For example, consider the SLDNFA-refutation for the query $\leftarrow faulty_lamp$ in figure 6. The only abduced atom is $power_failure(X)$. The abductive solution obtained this way would be:

$$\begin{aligned} \neg X = c1 \quad \wedge \quad \forall Y. (power_failure(Y) \leftrightarrow Y = X) \\ \wedge \quad \forall Y. (\neg broken(Y)) \wedge \forall Y. (\neg dry_cell(Y)) \end{aligned}$$

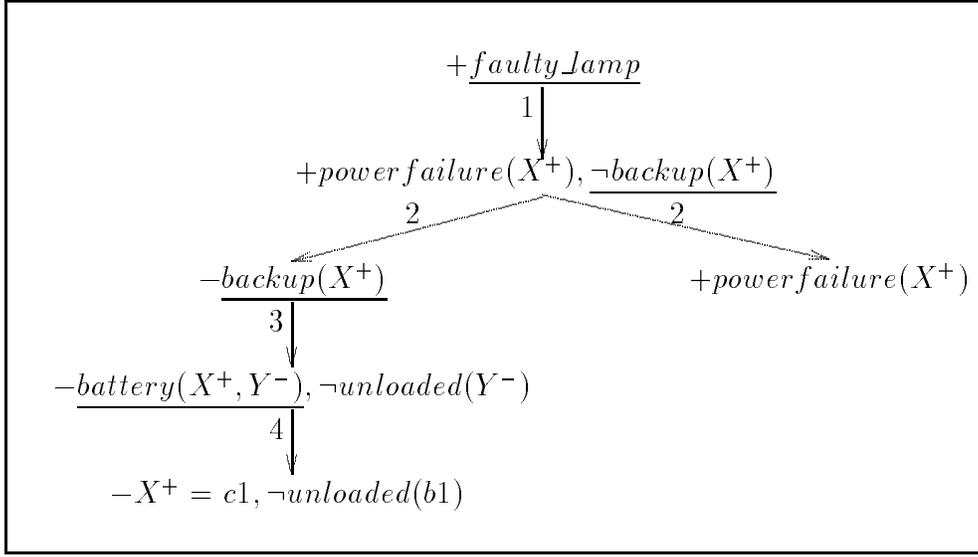


Figure 6: SLDNFA-refutation of $\leftarrow faultyLamp$

This answer states that there is one circuit X with power failure, no lamps are broken, no battery cells are dry and X is different from $c1$.

The completion of the abductive predicates entails that precisely the abduced atoms are true. By allowing more atoms to be true, it might be that some negative goal with selected abducible atom is not longer completed and hence, the conclusion of the SLDNFA-refutation might be falsified. However, often it is unnecessary to complete all abducible predicates. For example, in the SLDNFA-refutation in figure 6, there are no negative goals with selected abducible atoms; hence by adding additional abducible atoms, the conclusion of the refutation cannot be falsified. As a consequence, the following formula is a correct and much more general abductive solution:

$$power failure(X) \wedge \neg X = c1$$

This solution can be derived from the SLDNFA-refutation by taking the completion of all abducible predicates which appear in the selected atom of a negative goal; for the remaining abducible predicates, no completion is applied on the abduced atoms. This strategy of restricted completion can be further generalised. Assume that a negative goal has a selected abducible atom $p(\bar{t})$. For this negative goal to remain completed when other abducible atoms are added, it suffices that these new p -atoms do not unify with $p(\bar{t})$. We illustrate this with an example. Consider the abductive logic program $P_5^{\{r,p\}}$ with P_5 the set:

$$q \leftarrow r(f(Z)), p(Z)$$

A refutation for the query $\leftarrow \neg q, r(f(a)), r(a)$ is shown in figure 8. The negative goal $\leftarrow r(f(Z^-)), p(Z^-)$ is completed and remains completed if additional non-unifiable atoms (like $r(a)$) are abduced. A correct abductive answer that can be derived from this derivation is:

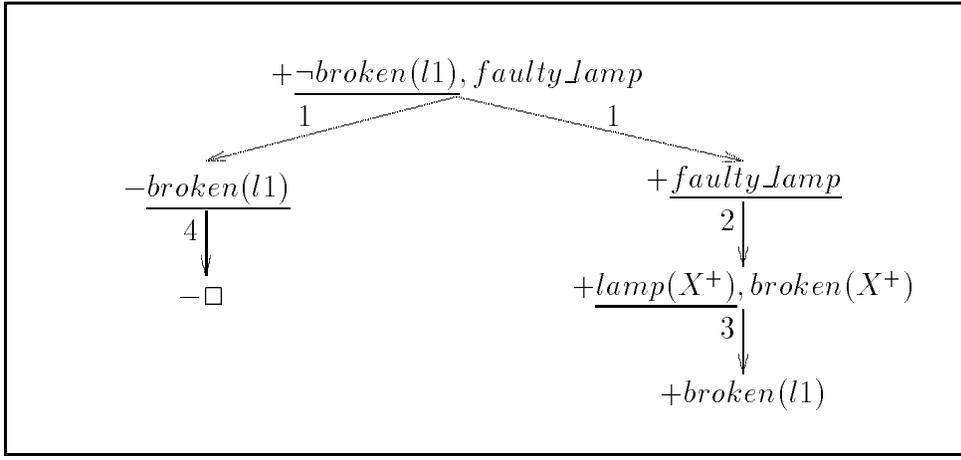


Figure 7: Failed SLDNFA-derivation $\leftarrow \neg broken(l1), faulty_lamp$

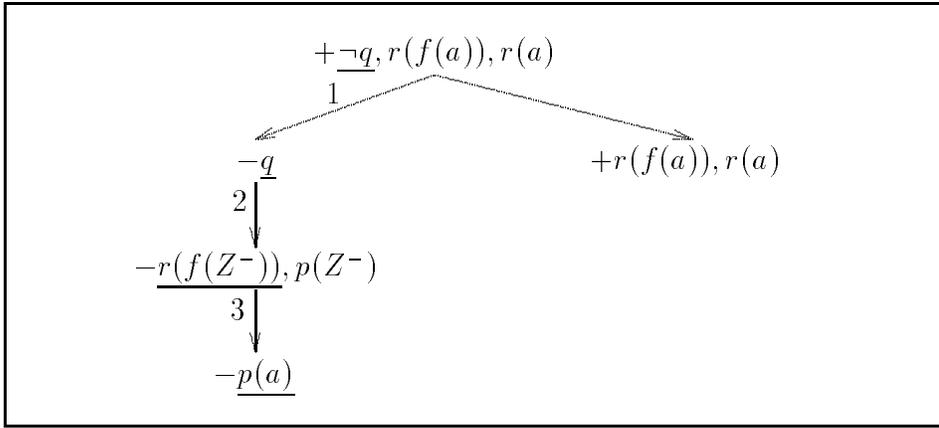


Figure 8: SLDNFA-refutation of $\leftarrow \neg q, r(f(a)), r(a)$

$$r(f(a)) \wedge r(a) \wedge (p(a) \leftrightarrow false) \wedge \\ \forall Z.(r(f(Z)) \leftrightarrow f(Z) = f(a) \vee f(Z) = a)$$

It consists of the abduced atoms, the negative goals without selected literal from which all non-abducible literals have been dropped, and equivalences which are obtained by completing all abduced atoms but only for the selected abducible atoms in negative goals. Evidently, these equivalences can be simplified by reducing the equality atoms. We obtain:

$$r(f(a)) \wedge r(a) \wedge \neg p(a) \wedge \forall Z.(r(f(Z)) \leftrightarrow Z = a)$$

Below, we define a number of simple and useful concepts and notations to be able to formalise this form of answer.

Given a marked formula F (wrt to the marking α) with positive free variables \overline{X} and negative free variables \overline{Y} , the formula $\forall^+(F)$ denotes $\forall \overline{X}.F$ and the formula $\forall^-(F)$ denotes $\forall \overline{Y}.F$.

Assume $p(\bar{t})$ is a marked atom, C_1, \dots, C_n marked program clauses or atoms $p(\bar{s}_i) \leftarrow B_i$ and \bar{Y}_i the tuple of negative variables of C_i . Define $CompDef(p(\bar{t}), \{C_1, \dots, C_n\})$ as the formula $\forall^-(p(\bar{t}) \leftrightarrow \exists \bar{Y}_1. (\bar{t} = \bar{s}_1 \wedge B_1) \vee \dots \vee \exists \bar{Y}_n. (\bar{t} = \bar{s}_n \wedge B_n))$.

For a given abducible atom $p(\bar{t})$ and SLDNFA-derivation K with abduced p -atoms $p(\bar{s}_1), \dots, p(\bar{s}_m)$, we use $CompDef(p(\bar{t}), K)$ as a shorthand for $CompDef(p(\bar{t}), \{p(\bar{s}_1), \dots, p(\bar{s}_m)\})$.

Given is an SLDNFA-derivation $K = ((\theta_1, \dots, \theta_n), T, \alpha)$ for a goal $\leftarrow Q_0$. Below, $\theta^{i..n}$ denotes $\theta_n o \dots o \theta_i$.

Definition 6.1 *The abduced atom set of K is the set $\Delta(K) = \{A \mid A \text{ is an abducible atom in a positive leaf of } K\}$.*

The abductive residue of a goal $\leftarrow Q$ is the goal $\leftarrow L_1, \dots, L_k$ where L_1, \dots, L_k are the equality literals and abducible literals of $\leftarrow Q$.

The negative abductive residue set $NAR(K)$ of K is the set: $\{\forall^-(\leftarrow Q') \mid \leftarrow Q' \text{ is the abductive residue of a negative goal } \leftarrow Q \text{ in } K \text{ without selected literal}^2\}$.

θ_a^K denotes $\theta^{1..n}|_{var(Q_0)}$ and is called the answer substitution generated by K .

The abductive completion $AbdComp(K)$ of K is the set of the completed definitions $CompDef(p(\bar{t}), K)$ of all selected abducible atoms $p(\bar{t})$ of negative goals of K .

The answer generated by K , denoted $Ans(K)$, is the open formula:

$$\theta_a^K \wedge \Delta(K) \wedge AbdComp(K) \wedge NAR(K)$$

In theorem 8.1, it will be proven that if K is a refutation, then $Ans(K)$ is a correct abductive solution for $\leftarrow Q_0$.

Also ground abductive solutions can be extracted from K . Let σ be any ground substitution based on some extension Σ' of Σ with $dom(\sigma) = var(\Delta(K) \wedge NAR(K) \wedge AbdComp(K))$. The extension of Σ with the symbols appearing in a substitution σ is denoted as Σ_σ . Let Δ be any set of ground abducible atoms based on Σ' . Let $\theta_{a,\sigma}^K$ denote $(\sigma o \theta_a^K)|_{var(Q_0)}$. D is the set of non-abducible predicates.

Definition 6.2 $(\Sigma', \Delta, \theta_{a,\sigma}^K)$ is a ground answer derivable from K using σ iff

$$\Delta^D \models_{\Sigma'} \sigma(\Delta(K) \wedge AbdComp(K) \wedge NAR(K))$$

The skolemised answer of K is the triple $(\Sigma_\sigma, \sigma(\Delta(K)), \theta_{a,\sigma}^K)$ where σ assigns different new skolem constants to each variable of $var(\Delta(K) \wedge NAR(K) \wedge AbdComp(K))$.

Consider the SLDNFA-refutation K_6 in figure 6. The ground abductive answers that can be derived from K_6 have the form $(\Sigma', \Delta, \varepsilon)$ with Δ any set of ground abducible atoms containing at least one atom $power_failure(t)$, where t is a ground term different from $c1$. t must be different from $c1$ because of the irreducible equality atom $X = c1$ in the negative goal $\leftarrow X = c1, \neg unloaded(b1)$. The skolemised answer is $(\Sigma \cup \{c_X\}, \{power_failure(c_X)\}, \varepsilon)$. Note that the ground abductive solution $\{power_failure(c1), dry_cell(b1)\}$ is not derivable from this refutation.

In [13], only the skolemised answer was derived from an SLDNFA-refutation. [25] showed for the first time that many ground abductive solutions can be derived from an abductive refutation. This class of ground answers is defined in the following proposition. Given is an SLDNFA-refutation K .

²Note that by definition 5.9 of SLDNFA-refutation, the abductive residue of a negative goal without selected literal in an SLDNFA-refutation contains at least one irreducible equality atom.

Proposition 6.1 *Let σ be a substitution with $\text{dom}(\sigma) = \text{var}(\Delta(K) \wedge \text{AbdComp}(K) \wedge \text{NAR}(K))$, such that each negative goal $\leftarrow Q$ without selected atom in K contains an irreducible equality atom $X^+ = t$ such that $\sigma(X)$ and $\sigma(t)$ are non-unifiable.*

$(\Sigma_\sigma, \sigma(\Delta(K)), \theta_{a,\sigma}^K)$ is a ground answer derivable from K using σ .

The proof is straightforward. Note that the skolemised answer belongs to the above class of ground answers. It is always possible to derive a skolemised answer from an SLDNFA-refutation. Hence the set of ground abductive answers is non-empty. It may be impossible to derive a ground answer based on Σ . An example is the query $\leftarrow p$ w.r.t. the program $P_0^{\{r\}}$ given in section 3.

In theorem 8.1, we prove that a ground answer is a ground abductive solution.

7 Correctness of basic operations

The proof techniques that will be used in most proofs are essentially the ones used in [7]: proofs by rewriting formulas by classical equivalence preserving laws such as commutativity and associativity of \wedge and \vee , distributivity of \wedge and \vee over each other and over \exists and \forall , laws of De Morgan. One special aspect is that the proofs are for 3-valued logic. This poses no problems: the used rewrite rules are equivalence preserving also in 3-valued logic. Below we summarise the other main laws which are often used in the proofs:

| | |
|---|---|
| $\forall X.F[X] \rightarrow F[t]$ | provided that variables of t have no bounded occurrences in F |
| $F \wedge \exists X.G \leftrightarrow \exists X.F \wedge G$ | provided X does not occur free in F |
| $F \vee \forall X.G \leftrightarrow \forall X.F \vee G$ | provided X does not occur free in F |
| $s = t \wedge F[s] \leftrightarrow s = t \wedge F[t]$ | provided that variables of s and t have no bounded occurrences in F |
| $(\exists X.X = t \wedge F[X]) \leftrightarrow F[t]$ | provided that variables of t have no bounded occurrences in F |
| $s = t \rightarrow F[s] \leftrightarrow s = t \rightarrow F[t]$ | provided that variables of s and t have no bounded occurrences in F |
| $(\forall X.X = t \rightarrow F[X]) \leftrightarrow F[t]$ | provided that variables of t have no bounded occurrences in F |

All of these rules can easily be proven. Note that the latter four rules mimic the application of substitutions. Two of them allow to eliminate variables occurring in the domain of a substitution. A common property of these tautologies is that they make assumptions of the kind that some variables have no bound occurrences in certain formulas. When these rules are applied in the proofs, this condition will be satisfied always because of the *standardisation apart* policy of SLDNFA. The effect of standardisation apart is that the same variable is never used in different contexts and occurs either free or is bound to one single quantifier. It is mainly via the above tautologies that standardisation apart has its role in the proof of the correctness of SLDNFA.

7.1 Correctness of Unification

Throughout this and the following two subsections, we assume the presence of a marking α . The following propositions are slight extensions of well-known results and are proven in appendix A.

Proposition 7.1 *There is a terminating algorithm which for any given finite equality set E , reports failure iff E has no positive solved form and returns a positive solved form E_s of E otherwise.*

Proposition 7.2 *Let E be an equality set.*

(a) $\exists(E)$ is satisfiable wrt $\text{FEQ}(\Sigma)$ iff there exists a (positive) solved form E_s of E . Equivalently, $\models_{\Sigma} \forall(\neg E)$ iff E_s has no (positive) solved form.

(b) Let E, E_s be equality sets based on Σ . E_s is a (positive) solved form of E iff E_s is in (positive) solved form and $\models_{\Sigma} \forall(E_s \leftrightarrow E)$.

(c) For any equality set E , either $\text{FEQ}(\Sigma) \models \exists(E)$ or $\text{FEQ}(\Sigma) \models \forall(\leftarrow E)$.

7.2 Correctness of positive and negative resolution

In this section we prove a number of correctness results for positive and negative resolution. Recall that for a goal $\leftarrow Q \equiv \leftarrow L_1, \dots, L_n$, the expression Q denotes the open conjunction $L_1 \wedge \dots \wedge L_n$ and $\leftarrow Q$ denotes the open disjunction $\neg L_1 \vee \dots \vee \neg L_n$.

First we define the notion of trivial resolution.

Definition 7.1 *Let $\leftarrow p(\bar{t}), Q$ be a goal, $p(\bar{s}) \leftarrow B$ a program clause. The trivial resolvent of $\leftarrow p(\bar{t}), Q$ and $p(\bar{s}) \leftarrow B$ on $p(\bar{t})$ is the goal $\leftarrow \bar{t} = \bar{s}, B, Q$.*

Proposition 7.3 *Let $\leftarrow Q \equiv \leftarrow p(\bar{t}), Q'$ be a marked goal and $C \equiv p(\bar{s}) \leftarrow B$ a marked program clause. Let $\leftarrow Q_t$ be their trivial resolvent $\leftarrow \bar{t} = \bar{s}, B, Q$. (a) If $\bar{s} = \bar{t}$ has no solved form, then:*

$$\models_{\Sigma} \forall(\leftarrow Q_t)$$

(b) If $\leftarrow Q_p$ be obtained by positive resolution of $\leftarrow Q$ and C on $p(\bar{t})$ using θ , then:

$$\models_{\Sigma} \forall[Q_t \leftrightarrow \theta \wedge Q_p]$$

(c) Let $\leftarrow Q_n$ be a negative resolvent of $\leftarrow Q$ and C .

$$\models_{\Sigma} \forall^+[\forall^-(Q_t) \leftrightarrow \forall^-(\leftarrow Q_n)]$$

Proof The proof is an easy consequence of the equivalence of an equality set and its solved form. (a) follows immediately from proposition 7.2(a). For (b), let $E_s = \theta$ be the solved form of $\bar{t} = \bar{s}$ used to obtain $\leftarrow Q_p$. By theorem 7.2(b), it holds that:

$$\begin{aligned} \models_{\Sigma} \bar{t} = \bar{s} \wedge B \wedge Q' &\leftrightarrow E_s \wedge B \wedge Q' \\ &\leftrightarrow E_s \wedge E_s(B \wedge Q') \\ &\leftrightarrow \theta \wedge Q_p \end{aligned}$$

This yields (b). To obtain (c), consider again the equivalence:

$$\models_{\Sigma} \bar{t} = \bar{s} \wedge B \wedge Q' \leftrightarrow E_s \wedge B \wedge Q'$$

Negate the formulas in the equivalences. Recall from section 4 that $E_s = E_{s+} \cup E_{s-}$. By universally quantifying the negative variables, and eliminating the negative variables in $\text{dom}(E_{s-})$, we obtain:

$$\begin{aligned} \models_{\Sigma} \forall^{-}(\leftarrow \bar{t} = \bar{s}, B, Q) &\leftrightarrow \forall^{-}(\leftarrow E_{s+}, E_{s-}, B, Q) \\ &\leftrightarrow \forall^{-}(\leftarrow E_{s+}, E_{s-}(B), E_{s-}(Q)) \\ &\leftrightarrow \forall^{-}(\leftarrow Q_n) \end{aligned}$$

□

Proposition 7.4 *Let $\leftarrow Q \equiv \leftarrow p(\bar{t}), Q'$ be a positively marked goal. (a) Let $\leftarrow Q_p$ be derived from $\leftarrow Q$ and a positively marked standardised apart variant C of a program clause of P^A by positive resolution on $p(\bar{t})$ using θ . Then:*

$$\models_{\Sigma} \forall^{+}(\theta(Q) \leftarrow Q_p) \leftarrow \text{CompDef}(p, P^A)$$

(b) *Assume that $\leftarrow Q_1, \dots, \leftarrow Q_m$ are all goals that can be obtained by positive resolution on $p(\bar{t})$ of $\leftarrow Q$ and positively marked and standardised apart variants C_1, \dots, C_m of clauses of the definition of p , using substitutions $\theta^1, \dots, \theta^m$. It holds that:*

$$\models_{\Sigma} \forall^{+}(Q \leftrightarrow \exists|_{\text{var}(Q)}(\theta^1 \wedge Q_1) \vee \dots \vee \exists|_{\text{var}(Q)}(\theta^m \wedge Q_m)) \leftarrow \text{CompDef}(p, P^A)$$

Proof (a) follows straightforwardly from (b): drop all disjuncts of the right side of the equivalence except one, move existential quantifiers as universal quantifiers to the front and then eliminate the substitution θ .

(b) Let C_{m+1}, \dots, C_n be positively marked and standardised apart variants of the other clauses of the definition of p with a head not unifiable with A . Construct a variant $\forall \bar{X}.p(\bar{X}) \leftrightarrow \Psi[\bar{X}]$ of $\text{CompDef}(p, P^A)$ using the sequence C_1, \dots, C_n . It does not share variables with $\leftarrow Q$. Substitute $\Psi[\bar{t}]$ for $p(\bar{t})$ in Q . This is equivalence preserving because variables quantified inside $\Psi[\bar{X}]$ do not occur in Q , due to the standardisation apart. Distribute conjunction of Q over disjunction and existential quantifiers of $\Psi[\bar{t}]$. Now we obtain the formula $\forall^{+}(Q \leftrightarrow \exists|_{\text{var}(Q)}(Q_t^1) \vee \dots \vee \exists|_{\text{var}(Q)}(Q_t^n))$ with Q_t^i trivial resolvents. (b) follow then straightforwardly using proposition 7.3(a+b). □

Proposition 7.5 (a) *Let $\leftarrow Q'$ be the negative resolvent of a marked goal $\leftarrow p(\bar{t}), Q$ and a marked program clause C . The following formula is a tautology:*

$$\models_{\Sigma} \forall^{+}[\forall^{-}(\leftarrow p(\bar{t}), Q) \wedge \forall^{-}(C) \rightarrow \forall^{-}(Q')]$$

(b) *Let $\leftarrow Q \equiv \leftarrow p(\bar{t}), Q'$ be a marked goal and C_1, \dots, C_n a set of marked program clauses with p in the head. Let $\leftarrow Q_1, \dots, \leftarrow Q_n$ be the trivial resolvents of $\leftarrow Q$ and C_1, \dots, C_n on $p(\bar{t})$. The following is a tautology:*

$$\models \forall^{+}[\forall^{-}(\leftarrow Q) \leftrightarrow \forall^{-}(\leftarrow Q_1) \wedge \dots \wedge \forall^{-}(\leftarrow Q_n)] \leftarrow \text{CompDef}(p(\bar{t}), \{C_1, \dots, C_n\})$$

Proof (a) The formula $\forall^-(C)$ is logically equivalent with $\forall^-(\neg p(\bar{s}) \rightarrow \neg B)$. Given this statement, (a) can be proven as follows:

$$\begin{aligned} \models_{\Sigma} \forall^-(\leftarrow p(\bar{t}), Q) \wedge \forall^-(C) &\rightarrow \forall^-(\leftarrow \bar{t} = \bar{s}, p(\bar{t}), Q) && \text{by subsumption} \\ &\rightarrow \forall^-(\leftarrow \bar{t} = \bar{s}, p(\bar{s}), Q) && \text{substituting } \bar{s} \text{ for } \bar{t} \\ &\rightarrow \forall^-(\leftarrow \bar{t} = \bar{s}, B, Q) && \text{using } \forall^-(\neg p(\bar{s}) \rightarrow \neg B) \end{aligned}$$

By proposition 7.3(c), the proposition follows.

(b) Let $CompDef(p(\bar{t}), \{C_1, \dots, C_n\})$ be of the form $\forall \bar{Z}. p(\bar{t}) \leftrightarrow \Psi_1[\bar{t}] \vee \dots \vee \Psi_n[\bar{t}]$. Substitute $\neg p(\bar{t})$ by $\neg \Psi_1[\bar{t}] \wedge \dots \wedge \Psi_n[\bar{t}]$. Distribute disjunction and universal quantifiers of $\forall^-(\leftarrow Q)$ over the conjunction. One obtains conjuncts of the form

$$\forall^-(\neg Q' \vee \forall^-(\neg \bar{t} = \bar{s}_i \vee \neg B_i))$$

The desired formula is now obtained by moving the inner universal quantifiers to the outside. This is equivalence preserving because the inner quantifiers quantify over variables which do not appear in $\neg Q'$. \square

8 Soundness of SLDNFA

Let P^A be an abductive logic program, $\leftarrow Q_0$ a query. Let K be an SLDNFA-refutation for $\leftarrow Q_0$.

Theorem 8.1 (soundness) (a) *It holds that*

$$P^A \models_{\Sigma} \forall(Q_0 \leftarrow Ans(K))$$

(b) *Moreover, $\exists(Ans(K))$ is satisfiable wrt P^A .*

(c) *Let $(\Sigma', \Delta, \theta_{a,\sigma}^K)$ be a ground answer derived from K using substitution σ . It holds that*

$$P + \Delta \models_{\Sigma'} \forall(\theta_{a,\sigma}^K(Q_0))$$

(d) *Moreover, $P + \Delta$ has a Σ' -model.*

Often, an abductive solution is required to be minimal wrt some preference relationship. We cannot prove such a minimality result. However, we will be able to prove a weaker result that if SLDNFA (or one of its variants) terminates, then all minimal solutions wrt certain preferences relations have been found.

Lemma 8.1 (a) *Let Q_r be the root goal of K . It holds that $Q_r = \theta_a^K(Q_0)$.*

(b) $\models \forall(Q_0 \leftarrow Ans(K)) \leftrightarrow \forall(Q_r \leftarrow \Delta(K) \wedge AbdComp(K) \wedge NAR(K))$

Proof (a) can be straightforwardly proven by an induction on the length of K .

(b) Starting from $\forall(Q_0 \leftarrow Ans(K))$, eliminate all variables in $dom(\theta_a^K)$. The result is $\forall(Q_r \leftarrow \Delta(K) \wedge AbdComp(K) \wedge NAR(K))$. \square

This lemma shows that to prove theorem 8.1(a), it suffices to prove $P^A \models_{\Sigma} \forall(Q_r \leftarrow \Delta(K) \wedge AbdComp(K) \wedge NAR(K))$. To prove this we will show that, given P^A and $\Delta(K) \wedge AbdComp(K) \wedge NAR(K)$, each node N (including the leaves) is entailed by the conjunction of its descendants. By a simple induction on the depth of a node, it is then possible to derive Q_r from $\Delta(K) \wedge AbdComp(K) \wedge NAR(K)$.

Definition 8.1 *The meaning $\mathcal{M}(N)$ of a node N of K is an open FOL formula, defined as follows:*

- if N is a positive goal $\leftarrow Q$, then $\mathcal{M}(N)$ is the open conjunction Q
- if N is a negative goal $\leftarrow Q$, then $\mathcal{M}(N)$ is $\forall^-(\leftarrow Q)$.

Lemma 8.2 *Let K be an SLDNFA-derivation. For every positive goal N in which a literal is selected and for every negative goal N in which a negative literal is selected, if N has descendants N_1, \dots, N_m ($m \geq 0$), it holds that $P^A \models_{\Sigma} \forall^+(\mathcal{M}(N) \leftarrow \mathcal{M}(N_1) \wedge \dots \wedge \mathcal{M}(N_m))$.*

Proof The proof is by induction on the length n of the SLDNFA-derivation. For $n = 0$, nothing is to be proven. Assume that the lemma holds for SLDNFA-derivations of length $n - 1$. Let K be an SLDNFA-derivation of length $n - 1$, and K' an SLDNFA-extension of K using a selection in K' . We prove the lemma for K' .

K' is obtained from K by applying a substitution θ of positive variables on all nodes and arcs of K and adding one or more descendants to the selected node N . So for all non-leaf nodes N except the selected one, with descendants N_1, \dots, N_m , it has to be proven that $P^A \models_{\Sigma} \forall^+(\theta(\mathcal{M}(N) \leftarrow \mathcal{M}(N_1) \wedge \dots \wedge \mathcal{M}(N_m)))$. By induction, we know that $P^A \models_{\Sigma} \forall^+(\mathcal{M}(N) \leftarrow \mathcal{M}(N_1) \wedge \dots \wedge \mathcal{M}(N_m))$. Application of a substitution θ to universally quantified variables preserves a logical consequence provided that the newly introduced variables do not occur bounded inside the formula. The variables that are possibly bounded in $\mathcal{M}(N), \mathcal{M}(N_i)$ are negative variables, while θ introduces only positive variables (proposition 5.2).

So it suffices to consider only the last selected node in K . Assume that new descendants N_1, \dots, N_m are added to N .

Assume that a first selection (N, L) is used in K to obtain K' , with $N \equiv \leftarrow L, Q$.

If N is a positive goal and L is a negative literal, then the formula to be proven is $\forall^+(L \wedge Q \leftarrow L \wedge Q)$. This is a tautology. Assume that N is a positive goal and L is a non-abducible p -atom. Then N has one descendant, a positive resolvent $\leftarrow Q'$. Proposition 7.4(a) shows that $\models_{\Sigma} \forall^+(\theta(L \wedge Q) \leftarrow Q') \leftarrow \text{CompDef}(p, P^A)$.

Assume that N is a negative goal and $L = \neg A$ a negative literal. Because the selection is safe, A does not contain negative variables. There are two possible extensions of K . It must be proven that $\forall^+(\forall^-(\leftarrow \neg A, Q) \leftarrow A)$ and $\forall^+(\forall^-(\leftarrow \neg A, Q) \leftarrow \forall^-(\leftarrow Q))$. Both formulas are tautologies since A contains no negative variables.

□

Note that the proof of this lemma is based on the safe selection condition.

Lemma 8.3 *Let K be an SLDNFA-derivation and N a completed negative goal $\leftarrow Q \equiv \leftarrow p(\bar{t}), Q'$ with selected atom $p(\bar{t})$ and descendants N_1, \dots, N_m ($m \geq 0$). Assume that the applied resolvent of the arc from N to N_i is C_i .*

If p is non-abducible then it holds that:

$$\models_{\Sigma} \forall^+[(\mathcal{M}(N) \leftrightarrow \mathcal{M}(N_1) \wedge \dots \wedge \mathcal{M}(N_m)) \leftarrow \text{CompDef}(p, P^A)]$$

If p is abducible then it holds that:

$$\models_{\Sigma} \forall^+[(\mathcal{M}(N) \leftrightarrow \mathcal{M}(N_1) \wedge \dots \wedge \mathcal{M}(N_m)) \leftarrow \text{CompDef}(p(\bar{t}), K)]$$

Proof (a) Let $\leftarrow Q_i^t$ be the trivial resolvent of $\leftarrow Q$ and C_i if p is abducible, or otherwise of $\leftarrow Q$ and a negatively marked variant of C_i , sharing no variables with Q . We prove first that:

$$\models_{\Sigma} \forall^+[\forall^-(\leftarrow Q_i^t) \leftrightarrow \mathcal{M}(N_i)]$$

The proof is by induction on the length of K . For $n = 0$, nothing is to be proven. Assume that the lemma holds for SLDNFA-derivations of length $n - 1$. Let K be an SLDNFA-derivation of length $n - 1$, and K' an SLDNFA-extension of K using a selection in K .

By the induction hypothesis, it holds that for any trivial resolvent Q_i^t of $\leftarrow Q$ and (a variant of) C_i that $\models_{\Sigma} \forall^+[\forall^-(\leftarrow Q_i^t) \leftrightarrow \mathcal{M}(N_i)]$. We must prove that this equivalence is maintained under the application of the substitution θ_n computed at the n 'th step. This is analogous as in lemma 8.2.

Second, assume that a reselection (N, C_i) is made in K and K is extended by appending N_i to N . Then (a) follows directly from proposition 7.3(c).

(b) Let C_1, \dots, C_g be the abduced p -atoms in K if p is abducible, or negatively marked and standardised apart variants of the program clauses of $\text{Def}(p, P^A)$ if p is non-abducible. Obviously, $\text{CompDef}(p(\bar{t}), \{C_1, \dots, C_g\})$ is $\text{CompDef}(p(\bar{t}), K)$ if p is abducible or an instance of $\text{CompDef}(p, P^A)$ (modulo renaming) if p is non-abducible.

Let $\leftarrow Q_j$ be the trivial resolvent of $\leftarrow Q$ and C_j . By proposition 7.5(b), it follows that:

$$\models_{\Sigma} \forall^+[(\forall^-(\leftarrow Q) \leftrightarrow \forall^-(\leftarrow Q_1) \wedge \dots \wedge \forall^-(\leftarrow Q_g)) \leftarrow \text{CompDef}(p(\bar{t}), \{C_1, \dots, C_g\})]$$

Because N is completed, for any C_j which does not correspond to an applied resolvent of N , $\leftarrow Q_j$ contains an unsolvable equality set. By proposition 7.3(a), $\forall^-(\leftarrow Q_j)$ can be dropped from the conjunction. There is a one-to-one correspondence between the remaining C_j 's and the applied resolvents of N . Given this correspondence, the proposition follows directly from (a). \square

Proposition 8.1 *Let K be an SLDNFA-refutation. For every N with descendants N_1, \dots, N_m ($m \geq 0$), it holds that*

$$P^A \models \forall^+[(\mathcal{M}(N) \leftarrow \mathcal{M}(N_1) \wedge \dots \wedge \mathcal{M}(N_m)) \leftarrow \Delta(K) \wedge \text{AbdComp}(K) \wedge \text{NAR}(K)]$$

Proof Let N be any node in K with descendants N_1, \dots, N_m ($m \geq 0$). The proof is by a case analysis.

- If N is a positive non-leaf goal or a negative goal with a negative selected literal, then the proposition follows directly from lemma 8.2. If N is a completed negative goal, then the lemma follows from lemma 8.3.
- If N is a positive leaf, then N is a goal $\leftarrow Q$ with only abduced atoms. Obviously $\models_{\Sigma} \forall^+[Q \leftarrow \Delta(K)]$.
- If $N \equiv \leftarrow Q$ is a negative goal without selected literal, then $NAR(K)$ contains the abductive residue $\leftarrow Q'$ of $\leftarrow Q$. Clearly, $\models_{\Sigma} \forall^+[\forall^-(\leftarrow Q) \leftarrow \forall^-(\leftarrow Q')]$.

□

Proof of theorem 8.1(a).

Let K be an SLDNFA-refutation for the goal Q_0 . Using proposition 8.1, a simple induction on the depth of the nodes in the refutation allows to prove that for each node N :

$$P^A \models_{\Sigma} \forall^+[\mathcal{M}(N) \leftarrow \Delta(K) \wedge AbdComp(K) \wedge NAR(K)]$$

This holds a fortiori for the root of the tree, for which $\mathcal{M}(N)$ is Q_r . Then (a) follows directly from lemma 8.1(b). □

Proof of theorem 8.1(c+d).

(d) The consistency of $P + \Delta$ based on Σ' follows from theorem 2.1.

(c) We show that $P + \Delta \models_{\Sigma'} \forall(\theta_{a,\sigma}^K(Q_0))$.

Take any Σ' -model M of $P + \Delta$. First, we prove that M is a Σ' -model of P^A or equivalently, that $M \models FEQ(\Sigma') \cup Comp(P^A) \cup Abd2(P^A)$. Given that $Comp(P^A) \subseteq Comp(P + \Delta)$ and $FEQ(\Sigma) \subseteq FEQ(\Sigma')$ and the monotonicity of FOL (even under 3-valued semantics), it suffices to show that any abductive predicate p has a 2-valued interpretation. This is the case, because the left of the equivalence $CompDef(p, \Delta)$ contains only equality atoms, which have a 2-valued interpretation in M .

By theorem 8.1(a), it follows that: $M \models \forall^+(Q_r \leftarrow \Delta(K) \wedge AbdComp(K) \wedge NAR(K))$. Consider the substitution σ . Note that $\theta_{a,\sigma}^K(Q_0) = \sigma(Q_r)$; this follows from lemma 8.1(a) and the definition of $\theta_{a,\sigma}^K$. Since $(\Sigma', \Delta, \theta_{a,\sigma}^K)$ is a ground answer, $M \models \sigma(\Delta(K) \wedge AbdComp(K) \wedge NAR(K))$. Hence, $M \models \forall^+(\sigma(Q_r))$.

□

Proof of theorem 8.1(b).

We should prove that there exists a Σ -model M' and variable assignment V such that $M' \models V(\theta_a^K \wedge \Delta(K) \wedge AbdComp(K) \wedge NAR(K))$.

Recall that there is at least one ground answer $(\Sigma_{\sigma}, \Delta_{\sigma}, \theta_{a,\sigma}^K)$ derivable from K , e.g. the skolemised answer. For one ground answer, we use the Σ_{σ} -model M that was constructed in the proof of theorem 8.1(c). It holds that $M \models \sigma(\Delta(K) \wedge AbdComp(K) \wedge NAR(K))$.

Define the variable substitution V with domain $dom(\sigma)$ as follows: for each $X = t \in \sigma$, define $V(X) = \mathcal{F}_M(t)$; i.e. $V(t)$ is the interpretation of t . Obviously, $M \models V(\Delta(K) \wedge AbdComp(K) \wedge NAR(K))$. We should further extend V for all variables of θ_a^K such that $M \models V(\theta_a^K)$.

Note that any variable in $dom(\theta_a^K)$ appears neither in $\Delta(K) \wedge AbdComp(K) \wedge NAR(K)$ nor in $ran(\theta_a^K)$. For any variable X in $ran(\theta_a^K)$ not appearing in $dom(V)$, select an arbitrary domain element x of M and define $V(X) = x$. Next for any atom $X = t \in \theta_a^K$, define $V(X) = \mathcal{F}_M(V(t))$. Obviously, $M \models V(\theta_a^K)$.

Finally, define $M' = M|_\Sigma$. M' is a Σ -model of P^A . M' and V are the desired model and variable substitution.

□

9 Completeness of SLDNFA

Throughout this section, we assume the presence of an abductive logic program P^A based on Σ . To formulate the completeness results, the concept of an SLDNFA-tree is needed. This tree represents the search tree of an SLDNFA-procedure.

Definition 9.1 *A partial SLDNFA-tree for a query $\leftarrow Q_0$ is a tree in which each node N is an SLDNFA-derivation K for $\leftarrow Q_0$. The root is the SLDNFA-derivation of length 0 and if N is a non-leaf with an SLDNFA-derivation K , then there exists a selection in K and the descendants of N are precisely the SLDNFA-extensions of K using the selection.*

An SLDNFA-tree for a query Q_0 is a partial SLDNFA-tree such that each leaf is a failed SLDNFA-derivation or an SLDNFA-refutation.

Branching in an SLDNFA-tree occurs when an SLDNFA-derivation has more than one SLDNFA-extension using a given selection. So branching occurs only when a non-abducible atom is selected in a positive goal or when a negative literal is selected in a negative goal.

Below, we define the concept of state formula associated to an SLDNFA-derivation and the concept of an explanation formula associated to a partial SLDNFA-tree.

Definition 9.2 *Let P^A be an abductive logic program, $\leftarrow Q_0$ a query, $K = ((\theta_1, \dots, \theta_n), T, \alpha)$ an SLDNFA-derivation of $\leftarrow Q_0$.*

The state formula $State(K)$ of K is the open formula:

$$\exists|_{var(Q_0)}(\theta^{0..n} \wedge \mathcal{M}(K))$$

where $\mathcal{M}(K)$ denotes the conjunction of the meaning of all nodes N of K .

Given a finite partial SLDNFA-tree W for a goal $\leftarrow Q_0$, its explanation formula $Expl(W)$ is:

$$State(K_1) \vee \dots \vee State(K_g)$$

where $\{K_1, \dots, K_g\}$ is the set of all leaves of W which are not failed.

$State(K)$ and $Expl(W)$ contain a lot of logically redundant subformulas; this redundancy simplifies the proof of the completeness theorem below. As shown in section 11.3, a logically equivalent and simpler formulae is obtained by dropping $\mathcal{M}(N)$ from $State(K)$, for each positive non-leaf goal N , for each negative goal N in which a negative literal is selected, and for each completed negative goal N in which a non-abducible atom is selected. An explanation formula $Expl(W)$ can be simplified by dropping all redundant parts of all its state formulas.

Theorem 9.1 (completeness) *Assume that $\leftarrow Q_0$ has a finite SLDNFA-tree W .*

- (a) $P^A \models_{\Sigma} \forall(Q_0 \leftrightarrow Expl(W))$
- (b) *Assume that Ψ is a formula based on Σ such that $\text{var}(\Psi) \subseteq \text{var}(Q_0)$. If $P^A \models_{\Sigma} \forall(Q_0 \leftarrow \Psi)$ then $P^A \models_{\Sigma} \forall(Expl(W) \leftarrow \Psi)$.*
- (c) *If all branches of W are finitely failed, then $P^A \models_{\Sigma} \forall(\leftarrow Q_0)$*
- (d) *If P^A is satisfiable with $\exists(Q_0)$ then W contains a successful branch.*
- (e) *Let $(\Sigma', \Delta, \theta)$ be a ground abductive solution for $\leftarrow Q_0$. There exists an SLDNFA-refutation K_i in a leaf of W and a substitution σ such that $\sigma(\Delta(K_i)) \subseteq \Delta$. Moreover $\theta_a^{K_i}(Q_0)$ is more general than $\theta(Q_0)$.*

The main completeness result is item (c) which states that SLDNFA is complete for failure: if there exists a failed SLDNFA-tree for an initial query, then the query has no abductive solutions.

SLDNFA does not generate all ground abductive solutions or all ground solutions satisfying some minimality criterium such as minimal cardinality, or minimal with respect to set inclusion. Remember that the ground abductive solution $\Delta = \{\text{power_failure}(c1), \text{dry_cell}(b1)\}$ cannot be derived from the SLDNFA-refutation in figure 6. This solution is minimal with respect to set inclusion. Other examples will be given in section 10 where two variants of SLDNFA are defined for which stronger completeness can be proven.

Instead, item (e) states a weaker result that each ground abductive solution Δ contains a subset of abduced atoms which are ground instantiations of the atoms in $\Delta(K_i)$ for some K_i in W . Note that this does not imply that $\Delta(K_i)$ contains less elements than Δ : indeed it is possible that σ maps two or more facts of $\Delta(K_i)$ to one fact of Δ .

The proof of the equivalence of the initial query and the explanation formula is based on the following lemma.

Lemma 9.1 *For any SLDNFA-derivation K , let there be a selection in K and let $\{K_1, \dots, K_g\}$ be the set of SLDNFA-extensions of K using the selection. The following equivalence holds:*

$$P^A \models_{\Sigma} \forall^+(\text{State}(K) \leftrightarrow \text{State}(K_1) \vee \dots \vee \text{State}(K_g))$$

Proof The proof is by a case analysis on the type of selection. Assume that K is an SLDNFA-derivation of length $n - 1$. Below, we assume that the goal $N \equiv \leftarrow L, Q'$ is selected in K .

Assume that a first selection (N, L) is made with N a positive goal and L a non-abducible p -atom. Assume that $\leftarrow Q_1, \dots, \leftarrow Q_m$ are obtained by positive resolution of this goal with (variants of) clauses of the definition of p , using substitutions $\theta_n^1, \dots, \theta_n^m$. By proposition 7.4(b), one may add to $State(K)$ the conjunct $\exists|_{var(Q)}(\theta_n^1 \wedge Q_1) \vee \dots \vee \exists|_{var(Q)}(\theta_n^m \wedge Q_m)$. Then distribute the conjunction and the existential quantifiers of $State(K)$ over the disjunctions in this formula and move the existential quantifiers of each disjunct to the front, joining the existential quantifiers of $State(K)$. One obtains m disjuncts of the form $\exists|_{var(Q_0)}(\theta^{0..m-1} \wedge \mathcal{M}(K) \wedge \theta_n^i \wedge Q_i)$. By applying the substitution θ_n^i on the other conjuncts, one obtains $State(K_i)$.

Assume that a first selection $(N, \neg A)$ is made in a positive goal N . $State(K)$ contains $\mathcal{M}(N) = \neg A \wedge Q$. $State(K_1)$ is obtained by adding two conjuncts $\neg A$ and Q to $State(K)$. Obviously, equivalence is preserved.

Assume that a first selection $(N, \neg A)$ is made in a negative goal $N \equiv \leftarrow \neg A, Q$. Then K has an extension K_1 which contains the positive goal $\leftarrow A$ and an extension K_2 which contains the negative goal $\leftarrow Q$. $State(K)$ contains $\mathcal{M}(N) = \forall^-(\leftarrow \neg A, Q)$. The following tautology holds:

$$\forall^+[\forall^-(\leftarrow \neg A, Q) \leftrightarrow A \vee \forall^-(\leftarrow Q)]$$

Add $A \vee \forall^-(\leftarrow Q)$ as a conjunct to $State(K)$; distribute the conjunction and the existential quantifiers of $State(K)$ over the disjunction inside this formula. One obtains $State(K_1) \vee State(K_2)$.

Assume that a first selection (N, A) is made in a negative goal with A an atom. $State(K_1) = State(K)$ and nothing is to be proven.

Assume that a reselection (N, C) is made in K . K_1 is obtained by adding $\forall^-(\leftarrow Q')$ with $\leftarrow Q'$ obtained by negative resolution of $\leftarrow A, Q$ and a program clause or abduced atom C on A . Proposition 7.5(a) shows that $\forall^-(\leftarrow Q')$ is entailed by $\forall^-(\leftarrow A, Q)$ and C , if C is an abducible atom, or $\forall(C)$ if C is a program clause. Either C is a conjunct of $State(K)$ or $\forall(C)$ is entailed by P^A . This implies that $P^A \models_{\Sigma} \forall^+(State(K) \leftrightarrow State(K_1))$.

□

Lemma 9.2 *For any finitely failed derivation K , it holds that:*

$$P^A \models_{\Sigma} \forall^+(\neg State(K))$$

Proof A failed SLDNFA-derivation K contains the negative empty goal or a positive goal N containing a non-abducible atom L which is not unifiable with the head of any program clause. In the first case the inconsistency of $State(K)$ is trivial. In the second case: since the set of SLDNFA-extensions using the selection (N, L) is empty, it follows from lemma 9.1 that $P^A \models_{\Sigma} \forall^+(State(K) \leftrightarrow false)$. □

Proof (of theorem 9.1)

(a) follows from lemma 9.1 and lemma 9.2 by a trivial induction on the number of nodes of the SLDNFA-tree.

(b) follows immediately from (a).

(c) When W is finitely failed, $Expl(W) = false$. By (a), $P^A \models_{\Sigma} \forall(\leftarrow Q_0)$.

(d) follows immediately from (c).

(e) We assume that new variables introduced by θ do not occur in the SLDNFA-tree. This conditions can always made to hold, by renaming some variables. By assumption $P + \Delta \models_{\Sigma'} \forall(\theta(Q_0))$.

Intuitively, the proof goes as follows. Take a Σ' -model M of $P + \Delta$. M is a model of P^A . By (a), $\forall(Q_0 \leftrightarrow Expl(W))$ is satisfied in M . Therefore, for some instance of the variables of $\leftarrow Q_0$ which satisfies Q_0 , there exists a $State(K_i)$ which is also satisfied. $State(K_i)$ is of the form $\exists \bar{Y}.(\theta^{0..n} \wedge \mathcal{M}(K_i))$, so the variable assignment can be extended to the variables \bar{Y} , such that $\theta^{0..n} \wedge \mathcal{M}(K_i)$ holds. In particular, any abduced atom $A \in \Delta(K_i)$ is satisfied, so A must correspond to some abduced atom of Δ . So we get a relation between terms in $\Delta(K_i)$ and in Δ . From this relation, the desired substitution can be obtained. Below, a precise formulation of this reasoning is given.

We construct the model M as follows. Take the Herbrand pre-interpretation of the language $\Sigma' \cup \{c_1, \dots, c_n\}$, with n the number of variables in $\theta(Q_0)$ and c_1, \dots, c_n new constants. By theorem 2.1, this pre-interpretation can be extended to a Σ' -model M of $P + \Delta$. Because for any abducible predicate p , the right hand of $CompDef(p, \Delta)$ contains only the equality predicate, $M \models Abd2(P^A)$. Hence M is a model of P^A and $M \models \forall(Q_0 \leftrightarrow Expl(W))$.

In M , the formula $\forall(\theta(Q_0))$ holds. Let V be the variable assignment of $var(\theta(Q_0))$ which assigns c_i to the variables X_i of $\theta(Q_0)$. It holds that $M \models V(\theta(Q_0))$. This is equivalent to $M \models V(\exists \bar{Z}.\theta \wedge Q_0)$, where $\bar{Z} = dom(\theta)$. Thus, there exists an extension V' of V such that $M \models V'(\theta \wedge Q_0)$.

Since $M \models V'(Q_0)$, it must hold that $M \models V'(State(K_1) \vee \dots \vee State(K_n))$. Therefore, there exists an i such that $M \models V'(State(K_i))$. $State(K_i)$ is a formula of the form $\exists \bar{Y}.\theta^{0..n} \wedge \mathcal{M}(K_i)$, with \bar{Y} all positive variables of K_i not appearing in Q_0 . Again V' can be extended to V'' such that $M \models V''(\theta^{0..n} \wedge \mathcal{M}(K_i))$. Since $\theta_a^{K_i}$ is a restriction of $\theta^{0..n}$, $M \models V''(\theta_a^{K_i})$. From now on, we write $\theta_a^{K_i}$ simply as θ_i .

Define the following substitution $\sigma = \{X = t \mid V''(X) = t \wedge t \notin \{c_1, \dots, c_n\}\} \cup \{X = X_i \mid X \notin var(\theta(Q_0)) \wedge V''(X) = c_i\}$. By construction of σ , it holds for any pair s, t of terms that $V''(t) \equiv V''(s)$ iff $\sigma(t) \equiv \sigma(s)$. We prove that σ is the desired substitution, i.e. first that $\sigma(\Delta(K_i)) \subseteq \Delta$ and second that $\sigma(\theta_i(Q_0)) = \theta(Q_0)$.

First, assume that $\Delta(K_i)$ contains an abduced atom $A \equiv p(\bar{t})$. It holds that $M \models V''(A)$. Assume that $CompDef(p, \Delta) = \forall \bar{X}.p(\bar{X}) \leftrightarrow \bar{X} = \bar{s}_1 \vee \dots \vee \bar{X} = \bar{s}_n$. Since M is a model of this definition, there must exist a j such that $M \models V''(\bar{t}) = V''(\bar{s}_j)$. Since equality is interpreted by identity in M , $V''(\bar{t}) \equiv V''(\bar{s}_j)$, hence $\sigma(\bar{t}) \equiv \sigma(\bar{s}_j)$. Since \bar{s}_j is ground, $\sigma(p(\bar{t})) \equiv \sigma(p(\bar{s}_j)) \equiv p(\bar{s}_j) \in \Delta$.

Second, since $M \models V''(\theta)$ and $M \models V''(\theta_i)$, we have for each term t^{Q_0} occurring in Q_0 , that $M \models V''(t^{Q_0} = \theta(t^{Q_0}))$ and $M \models V''(t^{Q_0} = \theta_i(t^{Q_0}))$. Since equality is identity, $V''(\theta(Q_0)) \equiv V''(\theta_i(Q_0))$. Hence, $\sigma(\theta(Q_0)) \equiv \sigma(\theta_i(Q_0))$. Since $dom(\sigma) \cap var(\theta(Q_0)) = \{\}$, $\sigma(\theta(Q_0)) = \theta(Q_0)$. Hence, $\sigma(\theta_i(Q_0)) \equiv \theta(Q_0)$.

This concludes the proof of item (e).

□

10 Extensions of the abductive procedure.

10.1 SLDNFA^o

In many applications of abduction, e.g. planning, abductive solutions with minimal cardinality are preferred. Consider the following simplified planning program. An action E initialises a condition p if some initial condition $r(E)$ holds when the action takes place. The same type of action initialises q if a second initial condition $s(E)$ holds. Moreover, if both $r(E)$ and $s(E)$ hold, then a third initial condition $o(E)$ must hold. The problem is to find a situation in which both p and q hold. The abducible predicates are $action, r, s$. Consider $P_6^{\{action, r, s\}}$ with P_6 the set:

$$\begin{aligned} p &\leftarrow action(E), r(E) \\ q &\leftarrow action(E), s(E) \\ o(X) \\ violated &\leftarrow action(E), r(E), s(E), \neg o(E) \end{aligned}$$

The query is $\leftarrow p, q, \neg violated$. The ground abductive solution with minimal number of actions is: $\{action(sk), r(sk), s(sk)\}$.

Figure 9 shows an SLDNFA-refutation for the query. This refutation is the leaf of an SLDNFA-tree consisting of one branch. The generated skolemised answer is $\{action(sk_1), r(sk_1), action(sk_2), s(sk_2)\}$. The answer generated by this refutation contains the conjunct $\neg X = Y$. Therefore, the ground abductive solution with minimal cardinality cannot be derived from this derivation.

In this section, we define the extension SLDNFA^o of SLDNFA which unifies abduced atoms with each other, and therefore computes solutions with minimal cardinality. The price is that the extended procedure crosses a larger computation tree. Our formulation of the extended algorithm allows a compromise between the improved completeness and the larger computation tree. It provides the opportunity to specify exactly for what abducible predicates the improved completeness should be obtained. The other abducible predicates are dealt with as in SLDNFA. The special abducible predicates will be called *strongly abducible*.

Below we assume the existence of an abductive logic program $P^{A, SA}$ with two different types of undefined predicates: abducible predicates A and strongly abducible predicates SA and $SA \cap A = \{\}$. SLDNFA^o-resolution is an extension of SLDNFA-resolution in which also strongly abduced atoms in positive goals can be selected.

Definition 10.1 (Selection) *Given is a pre-derivation K . A (first) SLDNFA-(re)selection in K is a (first) SLDNFA^o-(re)selection in K . A tuple (N, A) , where N is a positive goal without selected literal and A is a strongly abducible literal, is a first SLDNFA^o-selection in K .*

Below, we drop again the prefix "SLDNFA^o" from SLDNFA^o-selection.

The next concept to be defined is the notion of an SLDNFA^o-extension of a pre-derivation K .

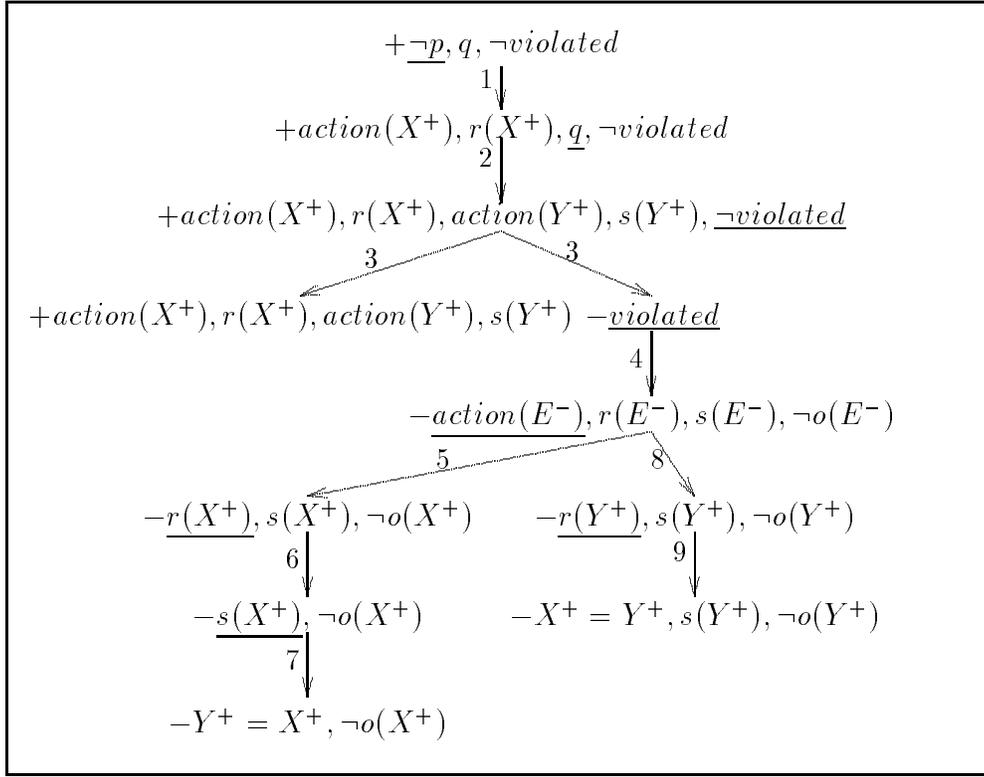


Figure 9: SLDNFA-refutation of $\leftarrow p, q, \neg false$

Definition 10.2 Given is a pre-derivation $K = ((\theta_1, \dots, \theta_n), T, \alpha)$.

Let (N, A) be a first selection in K , such that $N \equiv \leftarrow Q$ is a positive goal and A a strongly abducible atom. Let $\leftarrow Q \equiv \leftarrow A, Q'$ and $A \equiv p(\bar{t})$.

An SLDNFA^o-extension of K using first selection (N, A) is a pre-derivation $K' = ((\theta_1, \dots, \theta_n, \theta), T', \alpha)$, such that T' is obtained from T by adding zero, one or more descendants to N , marking N with selected literal A and applying θ on all nodes and labels of T . θ and the set of descendants S satisfy one of the following conditions:

- Either there exists an atom $p(\bar{s})$ in a positive goal of K such that $\bar{s} \neq \bar{t}$. θ is a solved form of $\bar{t} = \bar{s}$ and $S = \{N'\}$, with N' the positive goal $\leftarrow \theta(Q')$.
- Or $\theta = \varepsilon$ and for any abduced atom $p(\bar{s})$ in a positive goal of K such that $\bar{t} \neq \bar{s}$ and K contains no negative goal $\leftarrow \bar{s} = \bar{t}$ nor $\leftarrow \bar{t} = \bar{s}$, S contains a negative goal $\leftarrow \bar{t} = \bar{s}$.

An SLDNFA^o-extension of K using any other type of selection is an SLDNFA-extension of K using that selection.

SLDNFA^o differs from SLDNFA in its treatment of strongly abducible atoms, by allowing that either resolution with existing abduced facts is performed, or that an abduced fact is asserted to be different from the other abduced facts.

The definition of SLDNFA^o-derivation is analogous as for SLDNFA-derivation.

Definition 10.3 An $SLDNFA^\circ$ -refutation K for a goal $\leftarrow Q_0$ is an $SLDNFA^\circ$ -derivation which satisfies the same conditions as an $SLDNFA$ -refutation. In addition, we require that for any pair $p(\bar{t}), p(\bar{s})$ of different abduced atoms such that p is strongly abducible, K contains a negative goal $\leftarrow \bar{t} = \bar{s}$ or $\leftarrow \bar{s} = \bar{t}$.

The definitions of (partial) $SLDNFA^\circ$ -tree, state and explanation formula remain unaltered. Below, we prove a soundness and completeness theorem for $SLDNFA^\circ$. With respect to the semantics, there is no difference between abducible and strongly abducible predicates. We define $Comp(P^{A,SA}) = Comp(P^{AUSA})$.

Theorem 10.1 $SLDNFA^\circ$ satisfies the soundness results of theorem 8.1. $SLDNFA^\circ$ satisfies the completeness results of theorem 9.1. In addition to item (e), σ maps distinct atoms of $\Delta(K_i)$ to distinct atoms of Δ .

Hence, for each ground abductive solution $(\Sigma', \Delta, \theta)$, each finite $SLDNFA^\circ$ -tree W contains an $SLDNFA^\circ$ -refutation K_i , such that the skolemised answer of K_i contains less strongly abducible atoms than Δ .

Proof Let $\leftarrow Q \equiv \leftarrow A, Q'$ be a positive goal in which a strongly abducible atom $A = p(\bar{t})$ is selected. Assume that a set of abduced atoms $\{A_1, \dots, A_g\}$ occurs in K , such that A_1, \dots, A_g unifies with A . By positively resolving $\leftarrow Q$ and $A_i = p(\bar{s}_i)$ ($1 \leq i \leq g$), one computes positive resolvents $\leftarrow Q_i$ and solved forms θ_i of $\bar{t} = \bar{s}_i$. For the completeness, the following equivalence is important:

$$\models_{\Sigma} \forall^+ (Q \leftrightarrow (\theta_1 \wedge Q_1 \wedge \theta_1(A_1)) \vee \dots \vee (\theta_g \wedge Q_g \wedge \theta_g(A_g)) \vee (Q' \wedge A \wedge \neg \bar{t} = \bar{s}_1 \wedge \dots \wedge \neg \bar{t} = \bar{s}_g))$$

This equivalence is easy to obtain. Consider the following tautology :

$$\bar{t} = \bar{s}_1 \vee \dots \vee \bar{t} = \bar{s}_g \vee (\neg \bar{t} = \bar{s}_1 \wedge \dots \wedge \neg \bar{t} = \bar{s}_g)$$

By adding this tautology to Q as a conjunct, then moving its disjunction outside the conjunction, substituting the equality sets by their solved form and applying the solved form on Q , we obtain the stated equivalence.

From this equivalence the following implications can easily be derived.

$$\begin{aligned} &\models_{\Sigma} \forall^+ (\theta_i(Q) \leftarrow Q_i \wedge \theta_i(A_i)) \quad (1 \leq i \leq g) \\ &\models_{\Sigma} \forall^+ (Q \leftarrow A \wedge Q' \wedge \neg \bar{t} = \bar{s}_1 \wedge \dots \wedge \neg \bar{t} = \bar{s}_g) \end{aligned}$$

The latter two tautologies extend lemma 8.2 for the case that in a positive goal a strongly abducible atom is selected. From the extended lemma, the soundness of $SLDNFA^\circ$ follows in the same way as theorem 8.1.

The above equivalence allows to extend the lemma 9.1 for the case that a positive goal and a strongly abducible atom is selected in K . From this, the correctness of the explanation formula follows (as in theorem 9.1(a)). Using this result, the statements (b), (c), (d) and (e) of theorem 9.1 follow directly for $SLDNFA^\circ$.

Finally, we need to prove that for any pair of different strongly abduced atoms $p(\bar{s}), p(\bar{t})$ in $\Delta(K_i)$, $\sigma(p(\bar{s}))$ and $\sigma(p(\bar{t}))$ are different.

Observe that K_i contains the negative goal $\leftarrow \bar{s} = \bar{t}$ or $\leftarrow \bar{t} = \bar{s}$. Take M and V'' as in the proof of theorem 9.1(e). Recall that $M \models V''(\forall^-(s = t))$.

Verify that $M \models V''(\sigma)$. Hence, $M \models V''(\forall^-(\sigma(s) = \sigma(t)))$. As a consequence, $\sigma(s)$ and $\sigma(t)$ cannot be identical. Hence, $\Delta(K_i)$ and the skolemised answer contain less strongly abduced atoms than Δ .

□

10.2 SLDNFA₊

SLDNFA^o finds a ground abductive answer with minimal cardinality and therefore is a good candidate to be used for planning. However, SLDNFA nor SLDNFA^o can find all minimal ground abductive solutions with respect to set inclusion. An example is found in the faulty lamp problem (section 5). Consider the SLDNFA- and SLDNFA^o-refutation K_6 in figure 6. As argued in section 6, the ground answers derivable from this refutation have the form $(\Sigma', \Delta, \varepsilon)$ with Δ any set of ground abducible atoms containing at least one atom $powerfailure(t)$ where t is a ground term different from $c1$. $\{powerfailure(c1), dry_cell(b1)\}$ is a minimal ground abductive solution (w.r.t. set inclusion) not derivable from this refutation.

SLDNFA can easily be extended in order to find minimal solutions w.r.t. set inclusion. SLDNFA₊-resolution is an extension of the SLDNFA-resolution, which provides a different treatment for irreducible atoms in negative goals. The extended SLDNFA₊ has the more interesting completeness property that for any abductive solution Δ , there exists a generated solution $\Delta(K)$ and a substitution σ which maps $\Delta(K)$ into Δ , but in addition $\sigma(\Delta(K))$ can be proven to be a ground abductive solution for the query.

The notions of (first) SLDNFA₊-(re)selection are defined identical to (first) SLDNFA-(re)selection.

Definition 10.4 *Given is a pre-derivation $K = ((\theta_1, \dots, \theta_n), T, \alpha)$.*

Let $(N, X^+ = t)$ be a first selection in K , such that $N \equiv \leftarrow X^+ = t$, Q' is a negative goal and $X^+ = t$ an irreducible atom.

An SLDNFA₊-extension of K , using first selection $(N, X^+ = t)$ is a pre-derivation $K' = ((\theta_1, \dots, \theta_n, \theta), T', \alpha')$, such that T' is obtained from T by adding one descendant N' to N , labelling N with selected literal $X^+ = t$ and applying θ on all nodes and labels of T . α' , θ and N' satisfy one of the following conditions:

- *Either $\theta = \varepsilon$ and N' is a negative node containing $\leftarrow X^+ = t$.*
- *Or there exists a substitution $\delta = \{Y^- = Z_Y \mid Y^- \text{ is a negative variable in } t \text{ and } Z_Y \text{ is a fresh variable not appearing in } K\}$. α' is obtained from α by marking all variables Z_Y positive. $\theta = \{X^+ = \delta(t)\}$ and N' is a negative node with goal $\theta(\delta(\leftarrow Q'))$.*

An SLDNFA₊-extension of K using any other type of selection is an SLDNFA-extension of K using that selection.

The definition of SLDNFA₊-derivation remains unaltered.

Definition 10.5 An $SLDNFA_+$ -refutation K for a goal $\leftarrow Q_0$ is an $SLDNFA_+$ -derivation which satisfies the same conditions as an $SLDNFA$ -refutation. In addition, we require that all negative leaves in K without selected literal contain atomic irreducible equality goals.

Figure 10 presents an $SLDNFA_+$ -refutation for the goal $\leftarrow \text{faulty_lamp}$ which generates the solution with the dry battery.

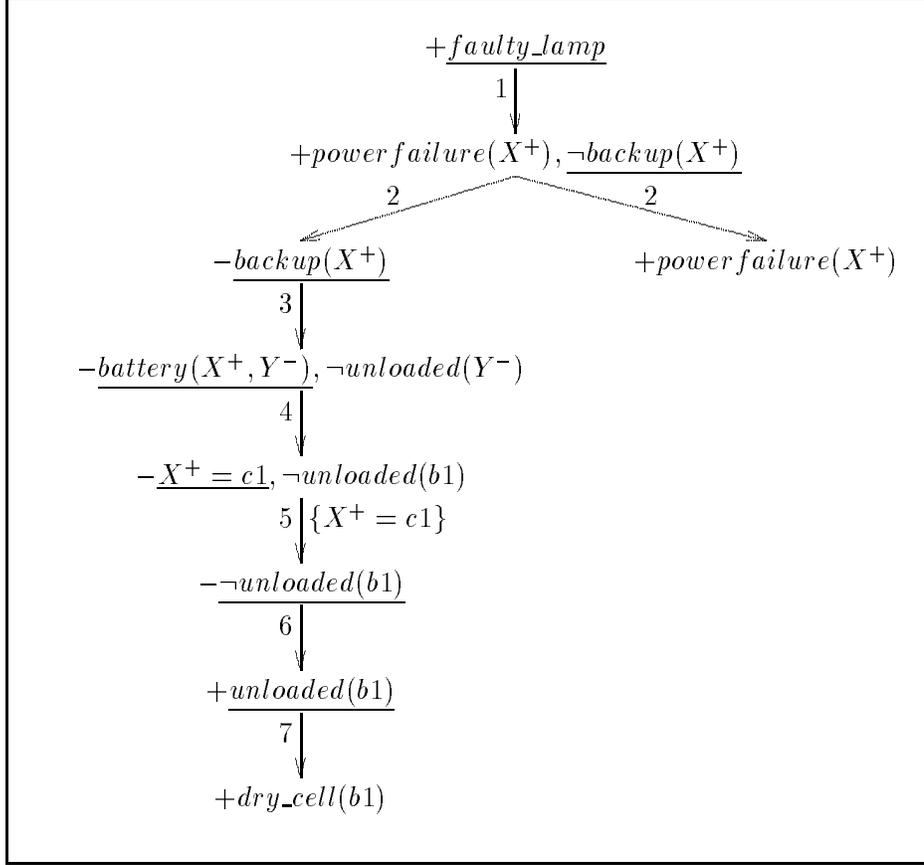


Figure 10: $SLDNFA$ -refutation of $\leftarrow p$

The definitions of (partial) $SLDNFA_+$ -tree, state and explanation formula remain unaltered. An $SLDNFA_+$ -tree is larger than the corresponding $SLDNFA$ -tree. Often, $SLDNFA_+$ will loop while $SLDNFA$ does not. As a trivial example, consider the program P_7^{\exists} , with P_7 the set:

$$\begin{aligned} p &\leftarrow p \\ q(a) &\leftarrow p \end{aligned}$$

The query $\leftarrow \neg q(X)$ has a finite $SLDNFA$ -tree but has no finite $SLDNFA_+$ -tree.

Theorem 10.2 $SLDNFA_+$ satisfies the soundness result as formulated in theorem 8.1 for $SLDNFA$ and the completeness results as formulated in theorem 9.1 for $SLDNFA$.

In addition, for any ground abductive solution $(\Sigma', \Delta, \theta)$, each finite SLDNFA₊-tree W contains a refutation K_i from which a ground answer $(\Sigma'', \Delta', \theta')$ is derivable such that $\Delta' \subseteq \Delta$.

Moreover, if W is obtained using prudent selections only, then $\Sigma'' \subseteq \Sigma'$ and $\theta'(Q_0)$ is more general than $\theta(Q_0)$.

The example below shows that if W is not obtained using prudent selections, then it cannot be guaranteed that $\Sigma'' \subseteq \Sigma'$ nor that $\theta'(Q_0)$ is more general than $\theta(Q_0)$. Consider the following logic program (without abducible predicates) based on an alphabet Σ with predicates p, d and constant a :

$$\begin{aligned} q(X) &\leftarrow d(X) \\ q(X) &\leftarrow \neg d(X) \\ d(a) \end{aligned}$$

Verify that $(\Sigma, \{\}, \varepsilon)$ is a ground abductive solution for the query $\leftarrow q(X)$. This query has a simple unique SLDNFA-tree W which is also the unique SLDNFA^o- and SLDNFA₊-tree. It contains two refutations, the first of which returns $X = a$, the second returns the disequality constraint $\leftarrow X = a$. In all ground answers $(\Sigma', \{\}, \theta)$ that can be derived from each of these refutations, $\theta(X)$ is ground, and if $\theta(X)$ is not a , then Σ' is a strict superset of Σ .

Proof Let $\leftarrow Q$ be a marked goal of the form $\leftarrow X^+ = t, Q'$ with $X^+ = t$ an irreducible atom. Define $\delta = \{Y^- = Z_Y^+ \mid Y^-$ is a negative variable in t and Z_Y^+ is a new positive variable not appearing in $Q\}$ and let \overline{Z} be $\text{ran}(\delta)$.

Let $\leftarrow Q''$ be $\theta(\delta(\leftarrow Q'))$.

We start with proving the following equivalence:

$$\models_{\Sigma} \forall^+ [\forall^-(\leftarrow Q) \leftrightarrow \forall^-(\leftarrow X^+ = t) \vee \exists \overline{Z}. \theta \wedge \forall^-(\leftarrow Q'')]$$

Clearly, the formula $\forall^+(\forall^-(\neg X^+ = t) \vee \exists \overline{Z}. \theta)$ is a tautology. Using this tautology one can derive that:

$$\models \forall^+ [\forall^-(\leftarrow Q) \leftrightarrow (\forall^-(\leftarrow Q) \wedge \forall^-(\neg X^+ = t)) \vee (\exists \overline{Z}. \theta \wedge \forall^-(\leftarrow Q'))]$$

In the first disjunct, $\forall^-(\leftarrow Q)$ is subsumed by $\forall^-(\neg X^+ = t)$ and can be dropped.

In the second disjunct, substitute X^+ by $\delta(t)$ in $\leftarrow Q$. One obtains $\exists \overline{Z}. \theta \wedge \forall^-(\leftarrow \delta(t) = t, Q')$. δ is obviously a solved form of $\delta(t) = t$. Hence, by theorem 7.2(b), one can substitute this disjunct by $\exists \overline{Z}. \theta \wedge \forall^-(\leftarrow \delta, Q')$. Finally, by eliminating the negative variables of $\text{dom}(\delta)$ in $\forall^-(\leftarrow \delta, Q')$, we obtain $\exists \overline{Z}. \theta \wedge \forall^-(\leftarrow Q'')$.

From the above equivalence the following two implications can be derived in a straightforward way:

$$\begin{aligned} \models_{\Sigma} \forall^+ [\forall^-(\leftarrow Q) \leftarrow \forall^-(\leftarrow X^+ = t)] \\ \models_{\Sigma} \forall^+ [\theta(\forall^-(\leftarrow Q)) \leftarrow \forall^-(\leftarrow Q'')] \end{aligned}$$

These two tautologies extend lemma 8.2 for the case that in a negative node an irreducible atom is selected. From the extended lemma, the soundness of SLDNFA_+ follows in the same way as theorem 8.1.

The above equivalence allows to extend lemma 9.1 for the case that an irreducible atom is selected in a negative goal. From this the correctness of the completeness theorem 9.1 follows.

Consider any ground abductive solution $(\Sigma', \Delta, \theta)$. Take a Herbrand model of $P + \Delta$ of the language $\Sigma' \cup \{c_1, \dots, c_n\}$, a refutation K_i in the SLDNFA_+ -tree W and V'' as in the proof of theorem 9.1(e). Note that V'' is a ground substitution of the variables of $\Delta(K_i) \wedge \text{AbdComp}(\Delta(K_i)) \wedge \text{NAR}(K_i)$. By its construction, $V''(\Delta(K_i)) \subseteq \Delta$.

We verify that V'' and $(\Sigma_{V''}, V''(\Delta(K_i)), \theta_{a, V''}^{K_i})$ is a ground answer derivable from K_i using V'' . V'' is by construction a ground variable substitution of the variables of $\Delta(K_i) \wedge \text{AbdComp}(\Delta(K_i)) \wedge \text{NAR}(K_i)$. It suffices to show that for each negative goal without selected atoms $\leftarrow X^+ = t$ in K_i , $V''(X^+), V''(t)$ are not unifiable. It holds that $M \models \forall^-(\leftarrow V''(X^+) = V''(t))$, hence $V''(X^+), V''(t)$ cannot have a unifier. By lemma 6.1, $(\Sigma_{V''}, V''(\Delta(K_i)), \theta_{a, V''}^{K_i})$ is a ground answer derivable from K_i .

Finally, assume that W is obtained using only prudent selections. By proposition 5.2, each positive variable Z^+ appearing in $\text{NAR}(K_i)$ occurs in $\Delta(K_i)$. Since V'' maps atoms of $\Delta(K_i)$ to abduced atoms of Δ based on Σ' , $V''(Z)$ is based on Σ' and by the construction of σ , $\sigma(Z) \equiv V''(Z)$ is a ground term based on Σ' . As a consequence, also $(\Sigma_\sigma, \sigma(\Delta(K_i)), \theta_{a, \sigma}^{K_i})$ satisfies the conditions of lemma 6.1. By construction of σ , $\Sigma_\sigma \subseteq \Sigma'$ and $\theta_{a, \sigma}^{K_i}(Q_0) \equiv \theta(Q_0)$.

□

Observe that the modifications to SLDNFA in SLDNFA° and SLDNFA_+ stand orthogonal to each other. That is, they can be combined into a new procedure SLDNFA_+° . This procedure is sound, and as a completeness result it can be stated that it generates both skolemised answers with minimal cardinality and that all minimal ground abductive solutions w.r.t. set inclusion can be derived from an SLDNFA_+° -tree. We obtain a (still primitive) framework of abductive procedures, in which a number of parameters can be set in order to fit the abductive procedure to the problem under consideration.

11 Discussion

11.1 Implementation of SLDNFA

SLDNFA is an effective procedure and a prototype has been implemented. The prototype has been extended with a constraint solver for the theory of linear order. As argued in [18], the prototype can be used as a general temporal reasoner and as an abductive planner for abductive event calculus (where the linear order is the order $<$ on time points). [18] applies the system on some planning problems and on some well-known temporal reasoning benchmark problems. The procedure uses SLDNFA° , with

the predicate *happens/1* as strongly abducible predicate. As a consequence, the procedure generates plans with a minimal number of events. The prototype is implemented as an enhanced vanilla meta-program on top of Prolog.

Our experience with the use of the prototype for planning has highlighted the need for an intelligent control strategy. Our initial implementation used the straightforward *depth first, left to right* control strategy. In many examples, the system entered an infinite branch of the search tree. We have solved this looping problem by using an iterative deepening regime. We plan to integrate other techniques that would avoid the looping and increase the efficiency, such as loop detection, intelligent control and intelligent backtracking.

An extension of the prototype with a finite domain constraint solver and with higher order cardinality and sum operators has been used successfully to schedule maintenance tasks in power plants. In this realistic problem, the cardinality and sum operators are used to express the higher order constraints that the number of power units in maintenance in any week may not exceed a certain number, respectively that the total capacity of power units not in maintenance in any week must be larger than a certain limit value. The logic of the problem is represented in 7 straightforward axioms which are considerably simpler than the constraint logic program that was developed to solve the same task.

11.2 Correctness of SLDNFA wrt other semantics

As mentioned in section 2, the 3-valued completion semantics is the weakest semantics presented so far for abductive logic programming. In particular, a model according to 2-valued completion semantics [8], generalised stable semantics [32], generalised well-founded semantics [49], justification semantics [14, 10] is a model according to 3-valued completion semantics (theorem 2.2). This implies that if P^A entails F according to 3-valued completion semantics, then P^A entails F also according to these other semantics.

The items (a) and (c) of the soundness theorem 8.1 and the items (a), (b) and (c) of the completeness theorem 9.1 state entailments of P^A under 3-valued completion semantics and hence, these hold under all stronger semantics. Also theorem 9.1 (d) which is based directly on (c) continues to hold. What may be lost under the stronger semantics is the consistency of a generated abductive answer. Consider the abductive logic program $P_8^{\{r\}}$ with P_8 the set:

$$p \leftarrow r, \neg p$$

The abductive answer of the query $\leftarrow r$ is the atomic formula r ; however, r is not satisfiable wrt $P_8^{\{r\}}$ under 2-valued completion semantics and generalised stable semantics. This shows that the items (b) and (d) of theorem 8.1 do not hold automatically for the stronger semantics. Neither does item (e) of theorem 9.1, since its proof is based on the existence of a model. As a counterexample, consider the logic program $P_9^{\{\}} with P_9 the set:$

$$\begin{aligned} p &\leftarrow \neg X = a, \neg p \\ s &(a) \end{aligned}$$

Under 2-valued completion semantics and generalised stable semantics, $P_9^{\{\}}$ entails $\forall X.X = a$. The goal $\leftarrow s(X)$ has a unique computed ground answer $(\Sigma, \{\}, \{X = a\})$.

According to theorem 9.1(e), this answer substitution should be most general; however, the empty substitution is a correct answer substitution under 2-valued completion semantics and generalised stable semantics.

The proof of the consistency of the generated abductive answers (theorem 8.1(b+d)) and item (e) of theorem 9.1 are based on theorem 2.1 which guarantees the existence of a model of the 3-valued completion. As proven in [14], this theorem holds also for the generalised well-founded semantics [49] and the justification semantics [14, 10]. Hence, for these semantics, all items of the soundness theorem 8.1 and the completeness theorem 9.1 continue to hold. As shown above, theorem 8.1(b+d) and theorem 9.1(e) do not hold in general with respect to 2-valued completion semantics [8] and generalised stable semantics [32]. However, for stratified, acyclic and locally stratified abductive logic programs, the theorem 2.1 can be proven wrt 2-valued completion semantics [8] and generalised stable semantics [32][14, 10]. Hence, also wrt to these semantics, all items of the soundness and completeness theorems continue to hold for important classes of programs.

11.3 Minimality of SLDNFA-answers

As correctness conditions for abductive answers, we have imposed that an abductive answer should entail the initial query and should be consistent. A commonly used extra condition is that an abductive answer should be *minimal* with respect to some pre-defined preference relation.

With respect to the minimality of ground abductive solutions, theorems 10.1 and 10.2 state weaker minimality results with respect to two natural preference orders: cardinality order and set inclusion. Not all ground answers generated by SLDNFA° or SLDNFA_+ are minimal, but all minimal solutions can be derived from the SLDNFA° -, resp. SLDNFA_+ -refutations in a finite SLDNFA° -, resp. SLDNFA_+ -tree. It is straightforward to extend SLDNFA° such that only solutions with minimal cardinality are generated. A naive implementation would be to compute an SLDNFA° -tree, and eliminate all non-minimal ground answers afterwards. Evidently, for efficiency the testing of non-minimality should be integrated with the computation of the SLDNFA° -tree; this allows run-time pruning of non-minimal branches. It is more difficult to implement SLDNFA_+ such that only minimal solutions with respect to set inclusion are generated, since in general, there are an infinite number of ground answers derivable from a refutation. We do not investigate this issue any further.

With respect to the minimality of (non-ground) abductive solutions, a natural preference pre-order is based on logical implication. For two open formulas Ψ_1, Ψ_2 based on Σ define:

$$\Psi_1 \leq \Psi_2 \text{ iff } P^A \models_\Sigma \forall (\Psi_1 \leftarrow \Psi_2)$$

It is straightforward to verify that \leq defines a pre-order (transitive, reflexive). One defines that x is minimal wrt to a pre-order \leq iff if $y \leq x$ then $x \leq y$.

Theorem 9.1(b) shows that the explanation formula $\text{Expl}(W)$ is a minimal formula entailing Q_0 wrt \leq . Recall that the explanation formula contains in general non-abducible predicates, therefore, it is not an *abductive answer* according to definition 3.2. The minimality wrt \leq of the disjunction of abductive answers $\text{Ans}(K_1) \vee \dots \vee \text{Ans}(K_g)$ does not hold in general.

One way to obtain that $Expl(W)$ or an equivalent formula contains only equality and abducible predicates and hence, that $Expl(W)$ is a minimal answer wrt \leq , would be to use $SLDNFA_+$ with a selection rule which delays the selection of an abducible atom in a negative goal until there are no non-abducible literals left. Below, we investigate this selection rule more closely.

Definition 11.1 *Given a finite SLDNFA-tree ($SLDNFA^\circ$ -, $SLDNFA_+$ -, $SLDNFA_+^\circ$ -tree) W , define $AbdExpl(W)$ as the formula obtained from $Expl(W)$ by simplifying $State(K_i)$ for each non-failed leaf K_i of W in the following way: drop $\mathcal{M}(N)$ from $State(K_i)$ for each positive goal N in K_i with a selected literal and for each negative goal N with a negative literal or an non-abducible atom as selected literal.*

Proposition 11.1 $P^A \models \forall(Expl(W) \leftrightarrow AbdExpl(W))$

This is a straightforward consequence of lemma 8.2 and lemma 8.3.

Assume that W is a finite $SLDNFA_+$ -tree which is constructed using a selection rule which delays selection of abducible atoms in negative goals until only abducible literals are left. In that case each negative goal N in each non-failed leaf of W in which an abducible atom is selected contains no non-abducible literals. This is proven in the next proposition.

Proposition 11.2 $AbdExpl(W)$ is an abductive answer minimal wrt \leq .

Proof If $AbdExpl(W)$ contains $\mathcal{M}(N)$ and N is a positive goal, then N has no selected literal and hence contains only abducible atoms. If N is a negative goal, then either N has an abducible atom as selected literal, but then by the selection rule, all literals in N are abducible; either N has no selected literal. But a negative goal without selected literal in an $SLDNFA_+$ -refutation contains only an equality atom. Hence, $AbdExpl(W)$ contains only abducible atoms and equality. Due to the equivalence of $AbdExpl(W)$ and $Expl(W)$, it follows that $AbdExpl(W)$ is an abductive answer formula which is minimal wrt \leq . \square

The problem with this way of using $SLDNFA_+$ is that this selection rule will often lead to floundering negation or looping. Consider the program $P_{10}^{\{r\}}$ with P_{10} the set:

$$\begin{array}{l} p \leftarrow r(X), \neg q(X) \\ q(a) \end{array}$$

Then the query $\leftarrow r(a), \neg p$ has a straightforward finite $SLDNFA_+$ -tree (with one branch) but when using the above selection rule, floundering negation occurs on $\neg q(X^-)$.

11.4 Widening the Applicability of SLDNFA

The functionality of an abductive procedure such as SLDNFA, and any abductive procedure which satisfies similar or stronger soundness and completeness theorems, can be extended far beyond abductive reasoning wrt abductive logic programs. It turns out that simple techniques allow to transform a much wider class of reasoning problems than typical abductive reasoning problems, wrt to a much wider class of theories than the typical abductive logic programs into a simple abductive problem

formulated wrt to a simple abductive logic program. These transformations have been proposed in [16].

The logic that can be handled is an integration of first-order logic (FOL) and *general* abductive logic programs (consisting of *general* program clauses which contain FOL formulas in the body [38]). This extends the formalism of abductive frameworks as defined in [32], which allows normal program clauses and FOL axioms in clausal form. Assume that a theory $\mathcal{T} = (P^A, T)$, with P^A a general abductive logic program³ and T a set of FOL axioms is given. It is simple to transform \mathcal{T} to an abductive logic program of the type that can be dealt with by SLDNFA. Construct the general abductive logic program by adding general clauses, $violated \leftarrow \neg F$, for each $F \in T$. Here, *violated* is a new propositional predicate. Then apply the Lloyd-Topor transformation [39] on this general abductive program. The result is a normal abductive logic program P'^A such that $(P'^A, \{\neg violated\})$ is logically equivalent to \mathcal{T} . More precisely, for any formula F based on the original alphabet, we have $\mathcal{T} \models_{\Sigma} F$ iff $(P'^A, \{\neg violated\}) \models_{\Sigma} F$ iff $P'^A \models_{\Sigma} \neg violated \rightarrow F$. This shows that P'^A can be used for problem solving in \mathcal{T} .

In [16] (and informally also in [15]), we have shown that using the above transformation technique, SLDNFA can be used not only for solving abductive problems with respect to \mathcal{T} , but also for solving deductive and satisfiability checking problems.

11.4.1 Deduction

The use of SLDNFA for deduction is based on theorem 9.1(c), which shows that if SLDNFA fails finitely, $\forall(\leftarrow Q_0)$ is entailed. Assume that \mathcal{T} is transformed to P'^A and that one wants to know whether $\mathcal{T} \models_{\Sigma} F$. If SLDNFA fails finitely on the query $\leftarrow \neg F, \neg violated$, then $\mathcal{T} \models_{\Sigma} F$; if it succeeds, then it is not true that $\mathcal{T} \models_{\Sigma} F$; otherwise, when floundering negation or looping occurs, nothing can be decided. It may be necessary to simplify $\leftarrow \neg F, \neg violated$ using the Lloyd-Topor transformation.

Note that an SLDNFA^o-tree and an SLDNFA₊-tree are larger than the SLDNFA-tree. Hence SLDNFA is more efficient for deduction than SLDNFA^o and SLDNFA₊.

11.4.2 Satisfiability Checking

Analogously, the satisfiability of F wrt \mathcal{T} can be checked by posing the query $\leftarrow F, \neg violated$. If SLDNFA or one of its variants succeeds, then F is satisfiable wrt \mathcal{T} due to theorem 8.1(a+b); in case of finite failure, F is not satisfiable; otherwise, nothing can be decided. Of the three variants, SLDNFA^o is the best candidate for proving finite satisfiability because it generates solutions with minimal number of abduced atoms. Executing SLDNFA^o using an iterative deepening strategy will ultimately find a finite abductive solution, if one exists. It is easy to construct examples where both SLDNFA and SLDNFA₊ loop without finding a solution while SLDNFA^o returns a finite solution. Consider $P_{13}^{\{next\}}$ with P_{13} the set:

$$\begin{aligned} & p(a) \\ & p(X) \leftarrow next(Y, X) \\ & f \leftarrow p(X), \neg exists_next(X) \end{aligned}$$

³A general abductive logic program contains general clauses: clauses in which the bodies are FOL formulas. The 3-valued completion semantics is trivially extended for general abductive logic programs.

$$exists_next(X) \leftarrow next(X, Y)$$

The set $\{next(a, a)\}$ is a ground answer for the query $\leftarrow \neg f$ and is computed by SLDNFA^o in case $next$ is a strongly abducible fact. SLDNFA and SLDNFA₊ are unable to produce a solution. Instead, a loop occurs producing larger and larger sets $\{next(a, X_1), next(X_1, X_2), next(X_2, X_3), \dots\}$.

The fact that FOL theories can be mapped to abductive logic programs entails that even for hierarchical abductive logic programs, SLDNFA and abductive procedures satisfying similar or stronger soundness and completeness theorems (in particular, theorem 8.1(a+b) and theorem 9.1(c), cannot be proven to terminate without floundering. Indeed, transforming a FOL theory in the above described way results in a hierarchical abductive logic program (the Lloyd-Topor transformation does not introduce recursion). Hence, a terminating non-floundering abductive procedure for hierarchical programs would be able to decide the consistency of a FOL theory in finite time. It is well-known that, in general, checking the consistency of a FOL theory is co-semi-decidable.

11.4.3 Database Updating

[31] applies abduction for intentional database updating in the context of deductive databases. An intentional update formula is a formula which states a desired property about the database. The intentional update problem consists of updating the base predicates of the database such that the intentional update formula is satisfied. [31] presents an abductive procedure and illustrates how this procedure can be used for the intentional update problem. Given a query $\leftarrow Q$ representing the intentional database update, the procedure computes a number of abductive solutions Δ , each consisting of ground literals of extensional database predicates (base predicates). One of these solutions is selected according to some preference criterion (e.g. minimal difference of old and new database) or by a query-the-user procedure. Using the selected solution, the extensional database D is updated: all atoms A such that $A \in \Delta \setminus D$ are inserted; all atoms A such that $A \in D$ and $\neg A \in \Delta$ are retracted from D .

The abductive procedure used in [31] does not solve the floundering abduction problem. A simple database example in which the floundering abduction problem arises goes as follows. Assume that the management of a firm decides that all employees of the sales department of the firm get a bonus. The intentional update formula can be formulated as follows:

$$\forall X. \neg employee(X) \vee \neg department(X, sales) \vee bonus(X)$$

The predicates *employee*, *department* and *bonus* are base predicates and are to be interpreted as abductive predicates. To compute the updated database, an abductive procedure is called with the following *general* query:

$$\leftarrow \forall X. \neg employee(X) \vee \neg department(X, sales) \vee bonus(X)$$

Using the Lloyd-Topor transformation, the general query can be transformed into the query $\leftarrow \neg p$ and the following definition for p :

$$p \leftarrow employee(X), department(X, sales), \neg bonus(X)$$

Note that under the completion semantics, it holds that:

$$p \leftrightarrow \exists X. \text{employee}(X) \wedge \text{department}(X, \text{sales}) \wedge \neg \text{bonus}(X)$$

It is clear that since base predicates are abductive, the query $\leftarrow \neg p$ leads to floundering abduction or floundering negation.

SLDNFA is not suited directly for this type of intentional update problem. The problem is that SLDNFA tends to minimise the set of abduced atoms Δ , while in the context of intentional database updating, one will usually prefer that the updated database is as close as possible to the old database. For example, due to its policy of minimising abductive solutions, SLDNFA returns for $\leftarrow \neg p$ the ground abductive solution with Δ the empty set. The resulting database indeed entails the intentional update formula, but the update of the database in this way implies that all employees should be fired without a bonus, which can hardly be considered as a correct implementation of the intention of the management (which was to give those people a bonus).

Beside the problem that naive application of SLDNFA does not try to minimise the difference between old and new database, the example also points to a problem with the above intentional update, which is independent of SLDNFA's minimisation strategy. It shows that the intentional update formula is imprecise, since it does not specify whether the intention is to delete *employee* or *department* facts or to insert *bonus* facts.

It turns out that both problems, the imprecise formulation of the intentional update and the unpleasant tendency of SLDNFA to minimise base/abductive predicates rather than the difference of old and new database, can be solved by the use of a meta-theory, an idea first used in [3]. [3] computes extensional database updates from a given set of intentional update formulas, by applying a model generator on a FOL meta-theory. Instead, we propose to use an abductive procedure on an abductive enhanced vanilla meta-program. Below we sketch this promising technique.

The kernel of our approach is the following abductive vanilla meta-program, which describes the predicate *new* representing the updated intentional database in terms of the predicate *old* representing the old intentional database:

$$\begin{aligned} \text{old}(F) &\leftarrow \text{base}(F), \text{db}(F) \\ \text{old}(F) &\leftarrow \text{clause}(F, B), \text{old}(B) \\ \text{old}(\text{true}) &\leftarrow \\ \text{old}((F, B)) &\leftarrow \text{old}(F), \text{old}(B) \\ \text{old}(\neg F) &\leftarrow \neg \text{old}(F) \\ \\ \text{new}(F) &\leftarrow \text{base}(F), \text{db}(F), \neg \text{delete}(F) \\ \text{new}(F) &\leftarrow \text{base}(F), \text{insert}(F) \\ \text{new}(\text{true}) &\leftarrow \\ \text{new}(F) &\leftarrow \text{clause}(F, B), \text{new}(B) \\ \text{new}((F, B)) &\leftarrow \text{new}(F), \text{new}(B) \\ \text{new}(\neg F) &\leftarrow \neg \text{new}(F) \end{aligned}$$

Here, *base* represents the base predicates, *db* implements the (access to the) extensional database, *clause* represents the definition of the view predicates. The predicates *delete* and *insert* represent respectively the deleted and inserted facts.

In our approach, the intentional database update in the sales department example would be formulated as follows:

$$\forall X. old(employee(X)), old(department(X, sales)) \rightarrow new(bonus(X))$$

The goal is to find a minimal set of inserts and delete operations to the old database such that old and new database satisfy the above formula. SLDNFA^o allows to solve this problem in the following way. We define the predicates *insert* and *delete* as strongly abducible. Applying the Lloyd-Topor transformation on the above formula yields a query $\leftarrow \neg p'$ and the following definition for p' :

$$p' \leftarrow old(employee(X)), old(department(X, sales)), \neg new(bonus(X))$$

The query $\leftarrow \neg p'$ can be given to SLDNFA^o. By theorem 10.1, when it terminates, SLDNFA^o has generated ground answers with minimal cardinality, consisting of *insert* and *delete* facts. As a consequence, those extensional databases which satisfy the intentional update and which are as close as possible to the old extensional database can be derived from the generated ground answers.

Though at present we do not have conducted a formal investigation of this technique, it seems to enjoy the same advantages as the approach of [3]:

- High expressivity of the language for formulating integrity constraints and intentional update formulas. FOL formulas are allowed; intentional updates can refer to old and new state at the same time (as in the sales department example). As a consequence, static and dynamic integrity constraints can be formulated. A static integrity constraint formulates a constraint on one state of the database; a dynamic integrity constraint formulates a constraint between the database and the updated database. An example taken from [3]:

$$\forall X, Sal, Sal_1. old(salary(X, Sal)) \wedge new(salary(X, Sal_1)) \rightarrow Sal_1 \geq Sal$$

- Computation of one extensional update satisfying a set of intentional update formulas at the same time (e.g. by taking the conjunction of the intentional update formulas).
- Automated integrity recovery of static and dynamic integrity constraints, in parallel with the computation of the extensional update.

Normally, integrity constraints are checked after an extensional update is generated. Using our approach it is possible to generate extensional updates which automatically satisfy the static and dynamic integrity constraints. By transforming the integrity constraints into a definition for a new predicate *violated* (as above) and adding $\neg new(violated)$ as an intentional update to the set of intentional updates, an abductive procedure will generate extensional updates which satisfy all integrity constraints.

With respect to the application of abduction for integrity recovery, a problem is that the above method is not incremental: i.e. it does not exploit the fact that the old database satisfies the static integrity constraints, but rechecks blindly all integrity constraints. In the past, incremental methods have been developed for integrity checking [37], [4], [36]. It should be investigated to what extent these techniques can be extended in order to obtain incremental integrity recovery methods.

11.4.4 Constraint Solving

The field of applications that can be covered by SLDNFA can be further drastically widened by incorporating constraint solvers in them. The integration of constraint solving techniques in abductive logic programming raises no different problems than those raised by their integration in SLDNF. Recently, abductive procedures have been combined with constraint solvers for partial order [45], for linear order [10], with finite domain constraint solvers and CLP(R) [30] [2].

11.5 Comparison with other procedures

For abductive logic programs without negation, the SLD-procedure can be extended easily to an abductive procedure [9], [23]. Given an initial query $\leftarrow Q$, these abductive procedures build an SLD-derivation, but resolve only on non-abducible predicates; the procedures return an answer substitution θ , together with a goal $\leftarrow Q'$, consisting of residual abducible atoms. It follows easily from the soundness of SLD-resolution that $P^A \models \forall(\theta(Q) \leftarrow Q')$. The procedure in [21] is an extension of this type of procedure, but uses skolemisation of non-ground abducible atoms and is able to deal with integrity constraints.

The first attempt to develop an abductive procedure for abductive logic programs with negation was presented in [52]. This -unformalised- abductive procedure is proposed in the context of planning and explanation in abductive event calculus. The procedure solves the floundering abduction problem by skolemising abducible atoms in positive goals. In many respects, SLDNFA and even more, SLDNFA₊^o, can be considered as a formalisation for this procedure, especially in the version with skolemisation (as in [13]). Building on the ideas in [52], [46, 45, 44] proposes an abductive procedure and an extension for planning in the event calculus. The solver can be proven correct only for a restricted class of abductive logic programs.

[31] proposed a formal abductive procedure based on the view of negation by failure as a special form of abduction, a view first presented in [22]. The procedure -which we mentioned already in section 11.4- generates sets of ground abducible literals. A correctness proof, for stratified abductive logic programs with respect to generalised stable semantics [32], is given. The procedure does not provide a solution for floundering abduction or floundering negation and is not sound in general with respect to non-stratified abductive logic programs. For example, the query $\leftarrow r$ wrt to $P_s^{\{r\}}$ (section 11.2) will succeed with $\Delta = \{r\}$. However, $P_s + \Delta$ has no generalised stable model. This soundness problem was solved in [51], which incorporates additional consistency checking in the procedure. The resulting procedure was proven sound and complete in the sense of theorem 9.1(c+d), but wrt generalised stable semantics.

In [8], an abductive procedure is presented which, for a given hierarchical abductive logic program P^A and query $\leftarrow Q$, derives an *explanation formula* E , equivalent with Q under the (2-valued) completion of P^A . This is done by repeatedly substituting atoms of non-abducible predicates by the equivalent part in their if-and-only-if definition, until no non-abducible atoms are left over. This technique provides a solution for floundering abduction and floundering negation. Observe that an explanation formula generated by naive rewriting may in general be very complex and inconsistent. Checking its satisfiability could be done using tableau-theorem proving [53] or Satchmo [41], but is

co-semi-decidable (see above). Moreover, in case of recursion, repeated naive rewriting of non-abducible atoms by their definition necessarily goes into a loop. In order to cope with these problems, [8] argues that a *normalisation technique* should be build in the procedure, but does not describe how this could be done.

Recall from section 11.3 that SLDNFA_+ with the selection rule which delays selection of abducible atoms in negative goals until no non-abducible literals occur in it, also generates explanation formulas consisting only of equality and abducible predicates. We argue that this instance of SLDNFA can be seen as an implementation of the procedure of [8] augmented *with a normalisation technique*. Two remarks can be made. First, recall that SLDNFA_+ does not provide a solution for floundering negation. In this sense, it is only a partial implementation of the procedure of [8]. Second, the approach of generating equivalent abductive explanation formulas will often fail to discover solutions, even when the floundering negation is solved completely. Consider the following propositional example $P_{11}^{\{r\}}$ with P_{11} the set:

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

It is easy to prove that no abductive explanation formula Ψ exists such that $p \leftrightarrow \Psi$ under 2- or 3-valued completion semantics. SLDNFA on the other hand investigates the computation tree branch per branch and, using an iterative deepening regime or loop detection, will find the abductive answer r .

Another procedure, related to abductive reasoning, is presented in [28]. This paper proposes a belief revision procedure to insert (or delete) a formula F in a given logic program P ; more precisely, it computes an update of P such that F can be proven (or no longer can be proven) from it. To insert F , the procedure tries to construct an SLDNF -refutation for the goal $\leftarrow F$ by adding facts to P to succeed positive goals and by deleting program clauses of P to fail negative goals. To delete F , the procedure tries to construct a failed SLDNF -tree for $\leftarrow F$ following a similar strategy.

A problem with this procedure is that, when solutions are generated, program clauses may be deleted which are needed elsewhere to satisfy positive goals, or that atoms are inserted which make negative goals to succeed. This problem is partially solved in [28] by checking afterwards whether the formula to be deleted or to be inserted has a finitely failed SLDNF -tree, respectively an SLDNF -refutation with respect to the updated program. This assures correctness, but many solutions cannot be found. For example, consider the logic program P_{12} :

$$p \leftarrow \neg q(a), q(X)$$

Assume that P_{12} is to be updated such that p can be derived. An example of such an update would be to insert the atom $q(b)$. The procedure in [28] will generate in a first step the update $\text{assert}(q(X))$. In the final step, it will discover that by adding $q(X)$ to the program, p does not have an SLDNF -refutation and this solution will be rejected. The solution $q(b)$ is not found. Note that SLDNFA will generate the skolemised answer $\{q(c_X)\}$ for the abductive logic program $P_{12}^{\{q\}}$.

In applications in the context of belief revision or database updating, it may be necessary to retract defeasible program clauses or integrity constraints. SLDNFA or any similar abductive procedure do not retract program clauses. However, as illustrated

in [31], there is a straightforward way to transform a set of program clauses marked as *defeasible* by the user into an equivalent abductive logic program such that the deletion of the defeasible clauses can be emulated by abducing abducible atoms. For example, a defeasible clause $C \equiv A \leftarrow B$ would be transformed into $A \leftarrow B, \neg del_C$ with del_C an abducible predicate. Abducing the atom del_C corresponds to deleting C . By adding arguments to del_C , it is possible to delete instances of this program clauses.

Recently, [54] proposes an interesting form of abductive resolution. It is an abductive extension of the SLDFA-resolution [19], a generic form of resolution which provides a framework for constructive negation. In principle, the abductive extension of SLDFA-resolution of [54] provides a generic framework for abductive procedures which solve the floundering abduction and negation problem. However, the problems to implement the abductive SLDFA-resolution are much larger than the problems to implement a non-abductive SLDFA-resolution. There are two unsolved problems:

- The non-abductive SLDFA relies on a normalisation or decision procedure for equality formulas (wrt FEQ). Such procedures are well-known and have been implemented. On the other hand, the abductive SLDFA defined in [54] relies on a generic *normalisation procedure* for checking the consistency of general abductive formulas; this procedure is called at each resolution step. The problem of determining the consistency of such a problem is co-semi-decidable. Just as for the procedure in [8], one could use tableau-theorem proving [53] or Satchmo [41]. Worse than in the case of [8], the consistency checking procedure should be called after each computation step. The practical integration of (an incremental version of) such a procedure in SLDFA is an unsolved problem.
- The generation of answers from a negative derivation is not defined in a constructive way. There may be an infinite number of answers.

To the best of our knowledge, there are currently no non-floundering implementations of the abductive SLDFA.

In [31], it was argued that floundering abduction and floundering negation are related phenomena. Our work with SLDNFA shows that the problem of floundering negation is strictly harder than floundering abduction. Solutions for constructive negation for non-abducible programs like the one proposed in [5] cannot be simply incorporated in an abductive procedure.

Also mentioned earlier, Fung [25] and Fung and Kowalski [26] propose the abductive *iff* procedure in the context of a generalised abductive framework for database updating and belief revision. This approach extends the approach of [31] for using abduction for database updating (see discussion in section 11.4). As argued in [25], in the special case that the database is empty, the *iff* procedure can be seen as a hybrid of the rewrite procedure of [8] and SLDNFA (in its formulation of [13]). Just as the procedure in [8], *iff* is formalised as a rewrite procedure, using the equivalences of the Clark completion, but incorporates normalisation techniques similar to the ones in SLDNFA. Independent of us, and as it appears, prior to us, Fung developed normalisation techniques treating universal and existential variables, rather than variables and skolem constants. *iff* provides a solution for floundering abduction but not for floundering negation.

As we argued in section 10.4, also SLDNFA can be interpreted as a rewrite procedure. Structurally, *iff* is quite similar to the old SLDNFA₊ of [13]; due to the

elimination of skolemisation in the current formalisation of SLDNFA, the similarities have become even stronger. In many respects though, the work presented here and in [25, 26] is complementary. Interestingly, due to the structural similarities between *iff* and SLDNFA, we believe the complementary results that are proven for SLDNFA can be transferred to *iff* and vice versa. While we have formalised SLDNFA in a logic programming style, [25, 26] has chosen for a formalisation as a rewrite procedure using the completion. This formalisation leads to more perspicuous proofs, due to the more obvious relationship between the procedure and the semantics. The formalisation of *iff* makes it easier to integrate glass-box constraint solvers in the procedure. As a matter of fact, [25, 26] formalises unification as a glass-box constraint solver. [25, 26] provides a soundness and two completeness results. Both completeness results assert that, under certain circumstances, *iff* generates all abductive solutions with least cardinality. One theorem proves this in case a finite computation tree exists (this is comparable with the result in theorems 10.1 and 10.2). A second, sophisticated theorem proves this in case of finite or infinite computation trees constructed using a fair selection rule. Much of Fung’s work is concerned with the use of *iff* for database updating, an issue which we only briefly considered in section 11.4.

On the other hand, in the study here, we show to what extent the techniques in SLDNFA allow to solve the floundering negation problem. We have considered also in more detail what kind of information can be extracted from an abductive refutation. An answer in [25] consists essentially of a ground database update, i.e. a set of positive and negative ground literals. The abductive answer presented in section 6 provides a much more general abductive explanation. Because we split up SLDNFA₊ in several pieces (SLDNFA, SLDNFA^o, SLDNFA₊), we are able to investigate in detail the effect of certain additional inference rules. This allows to provide more detailed minimality results than in [25] which proves only that solutions with minimal cardinality are generated. In section 11.4, we show how to apply abductive procedures, like SLDNFA or *iff* not only for abduction but also for deduction and consistency checking. Finally, as argued in section 10.4, we believe that the meta-approach to using abduction for database updating has important advantages compared to the approach followed in [31] and [25].

12 Acknowledgements

We thank Maurice Bruynooghe and anonymous referees for many helpful suggestions. We thank Eric Fung for pointing us to an error in a previous version of SLDNFA.

References

- [1] K. R. Apt and K. Doets. A new definition of SLDNF-resolution. *Journal of Logic Programming*, 18:177–190, 1994.
- [2] G. Bruneel and P. Clarebout. Een implementatie van SLDNFA+CLP(R) en zijn toepassing voor het redeneren over continue verandering. Masterthesis, supervisors: Marc Denecker and Danny De Schreye, Dept. of Computing, K.U.Leuven, in Dutch, 1994.

- [3] F. Bry. Intensional Updates: Abduction via Deduction. In *Proc. of the International Conference on Logic Programming*, pages 561–575, 1990.
- [4] F. Bry, R. Manthey, and B. Martens. Integrity verification in knowledge bases. In A. Voronkov, editor, *Logic Programming. Proceedings of the First and Second Russian Conference on Logic Programming*, Lecture Notes in Computer Science 592, pages 114–139. Springer-Verlag, 1991.
- [5] D. Chan. Constructive Negation Based on the Completed Database. In R. Kowalski and K.A. Bowen, editors, *Proc. of the 5th International Conference on Logic Programming*. MIT Press, 1988.
- [6] E. Charniak and D. McDermott. *Introduction to Artificial Intelligence*. Addison-Wesley, 1985.
- [7] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Databases*, pages 293–322. Plenum Press, 1978.
- [8] L. Console, D. Theseider Dupre, and P. Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.
- [9] P.T. Cox and T. Pietrzykowski. Causes for events: their computation and application. In *Proc. of the 8th International Conference on Automated Deduction*, 1986.
- [10] M. Denecker. *Knowledge Representation and Reasoning in Incomplete Logic Programming*. PhD thesis, Department of Computer Science, K.U.Leuven, 1993.
- [11] M. Denecker. A Terminological Interpretation of (Abductive) Logic Programming. In V.W. Marek, A. Nerode, and M. Truszczynski, editors, *International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 15–29. Springer, Lecture notes in Artificial Intelligence 928, 1995.
- [12] M. Denecker and D. De Schreye. A family of abductive procedures for normal abductive programs, their soundness and completeness. Technical Report 136, Department of Computer Science, K.U.Leuven, 1992.
- [13] M. Denecker and D. De Schreye. SLDNFA; an abductive procedure for normal abductive programs. In K.R. Apt, editor, *Proc. of the International Joint Conference and Symposium on Logic Programming*, pages 686–700, 1992.
- [14] M. Denecker and D. De Schreye. Justification semantics: a unifying framework for the semantics of logic programs. In *Proc. of the Logic Programming and Nonmonotonic Reasoning Workshop*, pages 365–379, 1993.
- [15] M. Denecker and D. De Schreye. Representing incomplete knowledge in abductive logic programming. In *Proc. of the International Symposium on Logic Programming*, pages 147–163, 1993.
- [16] M. Denecker and D. De Schreye. Reasoning in an integration of Classical Logic and Logic Programming. Draft, K.U.Leuven, 1995.

- [17] M. Denecker and D. De Schreye. Representing Incomplete Knowledge in Abductive Logic Programming. *Journal of Logic and Computation*, 5(5):553–578, September 1995.
- [18] M. Denecker, L. Missiaen, and M. Bruynooghe. Temporal reasoning with abductive event calculus. In *Proc. of the European Conference on Artificial Intelligence*, 1992.
- [19] W. Drabent. What is failure? an approach to constructive negation. Technical Report LITH-IDA-R-91-23, Linkoping University, 1993.
- [20] E. Eder. Properties of Substitutions and Unifications. *Journal of Symbolic Computation*, 1(1):31–46–312, 1985.
- [21] K. Eshghi. Abductive planning with Event Calculus. In R.A. Kowalski and K.A. Bowen, editors, *Proc. of the International Conference on Logic Programming*, 1988.
- [22] K. Eshghi and R.A. Kowalski. Abduction compared with negation as failure. In *Proc. of the International Conference on Logic Programming*. MIT-press, 1989.
- [23] J.J. Finger and M.R. Genesereth. Residue: a deductive approach to design synthesis. Technical Report STAN-CS-85-1035, Department of Computere Science, Stanford University, 1985.
- [24] M. Fitting. A Kripke-Kleene Semantics for Logic Programs. *Journal of Logic Programming*, 2(4):295–312, 1985.
- [25] T.H. Fung. An Abductive Proof Procedure Based on the Clark Completion. Technical report, Departement of Computing, Imperial College, in preparation of a PhD, 1995.
- [26] T.H. Fung and R. Kowalski. The iff proof procedure for abductive logic programming.
- [27] M. Gelfond and V. Lifschitz. Describing Action and Change by Logic Programs. In *Proc. of the 9th Int. Joint Conf. and Symp. on Logic Programming*, 1992.
- [28] A. Guessoum and J.W. Lloyd. Updating knowledge bases ii. Technical Report TR-90-13, Department of Computer Science, University of Bristol, 1990.
- [29] K. Inoue, Y. Ohta, and R. Hasegawa. Bottom-up Abduction by Model Generation. Technical Report TR-816, Institute for New Generation Computer Technology, Japan, 1993.
- [30] A. C. Kakas, R.A. Kowalski, and F. Toni. Abductive Logic Programming. *Journal of Logic and Computation*, 2(6):719–770, 1993.
- [31] A.C. Kakas and P. Mancarella. Database updates through abduction. In *Proc. of the 16th Very large Database Conference*, pages 650–661, 1990.

- [32] A.C. Kakas and P. Mancarella. Generalised stable models: a semantics for abduction. In *Proc. of the European Conference on Artificial Intelligence*, 1990.
- [33] S.C. Kleene. *Introduction to Metamathematics*, volume 1 of *Bibliotheca Mathematica*. Van Nostrand & Wolters-Noordhoff/North-Holland, Princeton, NJ & Groningen/Amsterdam, 1952.
- [34] K. Kunen. Negation in Logic Programming. *Journal of Logic Programming*, 4(3):289–308, 1987.
- [35] J.-L. Lassez, M.J. Maher, and K. Marriott. Unification revisited. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 587–625. Morgan Kaufmann, 1988.
- [36] M. Leuschel and B. Martens. Partial deduction of the ground representation and its application to integrity checking. Technical Report CW 210, Department Computerwetenschappen, K.U.Leuven, Belgium, April 1995. To appear in ILPS'95.
- [37] J. W. Lloyd, E. A. Sonenberg, and R. W. Topor. Integrity checking in stratified databases. *Journal of Logic Programming*, 4(4):331–343, 1987.
- [38] J.W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, 1987.
- [39] J.W. Lloyd and R.W. Topor. Making prolog more expressive. *Journal of Logic Programming*, 1(3):225–240, 1984.
- [40] M.J. Maher. Complete axiomatizations of the algebras of finite, rational and infinite trees. In *Proceedings of the Third Annual Symposium on Logic in Computer Science*, pages 348–357. The MIT Press, 1988.
- [41] R. Manthey and F. Bry. Satchmo: a theorem prover implemented in prolog. In *Proc. of the Conference on Automated Deduction*. Springer-Verlag, 1988.
- [42] A. Martelli and U. Montanari. An efficient unification algorithm. *Transactions on Programming Languages and Systems*, 4(2):258–282, 1982.
- [43] M. Martelli and C. Tricombi. A New SLDNF-Tree. *Information Processing Letters*, 43(2):57–62, 1992.
- [44] Lode R. Missiaen, Marc Denecker, and Maurice Bruynooghe. CHICA, an abductive planning system based on event calculus. *Journal of Logic and Computation*, 5(5):579–602, September 1995.
- [45] L.R. Missiaen. *Localized abductive planning with the event calculus*. PhD thesis, Department of Computer Science, K.U.Leuven, 1991.
- [46] L.R. Missiaen. Localized abductive planning for robot assembly. In *Proceedings 1991 IEEE Conference on Robotics and Automation*, pages 605–610. IEEE Robotics and Automation Society, 1991.

- [47] R.C. Moore. The role of logic in knowledge representation and commonsense reasoning. In *Proc. of AAAI-82*, pages 428–433, 1982.
- [48] C.S. Peirce. *Philosophical Writings of Peirce*. Dover Publications, New York, 1955.
- [49] L.M. Pereira, J.N. Aparicio, and J.J. Alferes. Hypothetical Reasoning with Well Founded Semantics. In B. Mayoh, editor, *Proc. of the 3th Scandinavian Conference on AI*. IOS Press, 1991.
- [50] D. Poole. A Logical Framework for Default Reasoning. *Artificial Intelligence*, 36:27–47, 1988.
- [51] K. Satoh and N. Iwayama. A Query Evaluation method for Abductive Logic Programming. In K.R. Apt, editor, *Proc. of the International Joint Conference and Symposium on Logic Programming*, 1992.
- [52] M. Shanahan. Prediction is deduction but explanation is abduction. In *Proc. of the IJCAI89*, page 1055, 1989.
- [53] Raymond Smullyan. *First-Order Logic*. Springer, New York, second edition, 1968.
- [54] F. Teusink. Using SLDFA-resolution with abduction. In *Proc. of the ILPS'93 workshop on Logic Programming with Incomplete Information*, pages 35–47, 1993.

A Soundness of Unification

In this section, we prove some slight extensions of well-known theorems about the relationship of unification and *FEQ*. We prove proposition 7.1 which states that there exists a terminating algorithm which transforms equality sets in positive solved form if possible and reports failure otherwise. This proposition is a trivial extension of theorems about diverse unification algorithms (e.g. [42]) but is new because the notion of positive solved form is new. Throughout this section, we assume the existence of a marking α of variables.

We prove also proposition 7.2 which states that (a) an equality set is consistent iff it has a (positive) solved form and (b) for any equality set E_s in (positive) solved form, E_s is a (positive) solved form of E iff $\models_{\Sigma} \forall (E \leftrightarrow E_s)$. (a) is well-known and is a trivial consequence of proposition 7.1. (b) gives a characterisation of the notion of mgu in terms of *FEQ*. We have not found a proof of (b) in the literature. To the best of our knowledge, the relationship between unification and the theory *FEQ* has not earlier been investigated in the same depth as we do here.

The following proposition reformulates proposition 7.1.

Proposition A.1 *There is a terminating algorithm which for any given finite marked equality set E , reports failure iff E has no positive solved form and returns a positive solved form E_s of E otherwise.*

Moreover, if the algorithm fails, then $\models_{\Sigma} \neg \exists (E)$. If it returns a positive solved form E_s then $\models_{\Sigma} \forall (E \leftrightarrow E_s)$.

Proof The equality reduction algorithm is defined as follows. Given a marked equality set E as input, it acts by the following two steps.

- Algorithm A.1 (Equality Reduction)**
- *step 1) Apply one of the existing unification algorithms on E (e.g. [42]). If the algorithm fails, report failure. Otherwise assume that the output is a solved form E'_s . Go to step 2.*
 - *step 2) Construct a substitution σ by selecting for each Y^- with non empty set $\{X^+ | X^+ = Y^- \in E'_s\}$, one element X^+ from this set and adding $Y^- = X^+$ to σ . Return $E_s = \sigma o E'_s$.*

It follows directly from the correctness of the unification algorithm that the equality reduction algorithm terminates and that the algorithm reports failure iff E has no solved form and $\models_{\Sigma} \neg \exists(E)$. Moreover, if step (1) yields a solved form E'_s , then E'_s is an idempotent mgu and $\models_{\Sigma} \forall(E \leftrightarrow E'_s)$.

E_s is $\sigma o E'_s$. E_s is obtained from E'_s by switching the atom $X^+ = Y^- \in E'_s$ to $Y^- = X^+ \in E_s$ and substituting X^+ for Y^- at the right of all other atoms of E'_s , for all $Y^- = X^+ \in \sigma$.

It is straightforward to prove that E_s is in solved form. That it is in positive solved form is easy to see too. Assume $X^+ = Y^- \in E_s$. Since σ has no negative variables in its range, $X^+ = Y^- \in E'_s$. By construction of σ , there exists an atom $Y^- = Z^+ \in \sigma$. But then, $E_s = \sigma o E'_s$ contains $X^+ = Z^+$ and not $X^+ = Y^-$.

It is obvious that E_s can be obtained from E'_s by application of the symmetry law and repeated application of the substitution law of the standard equality. E'_s can be obtained from E_s in an analogous way. Hence, $\models_{\Sigma} \forall(E'_s \leftrightarrow E_s)$.

It remains to show that E_s is an mgu of E . Obviously $E_s = \sigma o E'_s$ is a unifier of E . To prove that E_s is an mgu of E , it suffices to show that for some substitution σ' , $E'_s = \sigma' o E_s$. Define σ' as the inverse of σ : $\sigma' = \{X^+ = Y^- | Y^- = X^+ \in \sigma\}$. It is easily verified that $E'_s = \sigma o E_s$.

□

The following lemma is trivial.

Lemma A.1 *For any equality set E_s in solved form, $\exists(E_s)$ is satisfiable wrt $FEQ(\Sigma)$. Moreover, $\models_{\Sigma} \exists(E_s)$.*

Proof $FEQ(\Sigma)$ is consistent. Take any Σ -model M of $FEQ(\Sigma)$ and any variable assignment V of the variables of $ran(E_s)$. For any $X = t \in E_s$, extend V such that $V(X) = \mathcal{F}_M(V(t))$. Obviously, $M \models V(E_s)$. □

Lemma A.2 *Let Σ be an alphabet, θ an idempotent substitution based on Σ . Let \bar{s}, \bar{t} be tuples of terms based on Σ . θ is a unifier of \bar{s}, \bar{t} iff $\models_{\Sigma} \forall(\theta \rightarrow \bar{s} = \bar{t})$.*

Proof The proof is for a pair of terms s, t but can be trivially extended to the case of tuples of terms \bar{s}, \bar{t} .

We use the following observation. Using repeated application of the tautology $X = t \rightarrow F[X] \leftrightarrow X = t \rightarrow F[t]$, one finds that:

$$\models \forall[(\theta \rightarrow s = t) \leftrightarrow (\theta \rightarrow \theta(s) = \theta(t))]$$

Since the variables at both sides of the equivalence are the same, this implies directly:

$$\models \forall(\theta \rightarrow s = t) \leftrightarrow \forall(\theta \rightarrow \theta(s) = \theta(t))$$

So, it suffices to check that $\models_{\Sigma} \forall(\theta \rightarrow \theta(s) = \theta(t))$ iff $\theta(s) \equiv \theta(t)$.

When $\theta(s) \equiv \theta(t)$, it holds that $\models_{\Sigma} \forall(\theta \rightarrow \theta(s) = \theta(t))$.

Vice versa, assume that $\models_{\Sigma} \forall(\theta \rightarrow \theta(s) = \theta(t))$ and $\theta(s) \not\equiv \theta(t)$. We construct a model M of $FEQ(\Sigma)$ and a variable assignment V such that $M \models V(\theta) \wedge V(\theta(t)) \neq V(\theta(s))$, thus obtaining a contradiction.

Consider a Herbrand interpretation M of Σ' , an extension of Σ with two new constants c_1, c_2 . $M|_{\Sigma}$ is a non-Herbrand Σ -model of $FEQ(\Sigma)$. Because $\theta(s) \not\equiv \theta(t)$, there exists a disagreement pair s_1, t_1 of subterms of $\theta(s), \theta(t)$ such that either s_1, t_1 have different principal functors, or are different variables or one is a variable and the other is a non-variable term. Define a variable assignment V as follows:

- if s_1 is a variable, then $V = \{s_1 = c_1\}$
- else, if t_1 is a variable then $V = \{t_1 = c_1\}$
- else, $V = \{\}$

Now extend V to $\text{ran}(\theta) \cup \text{var}(\theta(s)) \cup \text{var}(\theta(t))$ by assigning other variables the value c_2 . From the construction of V , it follows that $V(s_1) \neq V(t_1)$; hence $V(\theta(s))$ and $V(\theta(t))$ are two non-identical ground terms. Since θ is idempotent, V does not assign values to variables in $\text{dom}(\theta)$. Finally, we extend V to $\text{dom}(\theta)$ by defining $V(X) \equiv V(t)$ for any $X = t \in \theta$.

We find that $M|_{\Sigma} \models V(\theta) \wedge V(\theta(s)) \neq V(\theta(t))$. □

Lemma A.3 *Let θ, σ be idempotent substitutions.
 σ is more general than θ iff θ is a unifier of σ .*

Proof Lemma 3.3 in [20] proves that if σ is idempotent, then σ is more general than θ iff $\theta = \theta\sigma$. So it suffices to show that $\theta = \theta\sigma$ iff θ is a unifier of σ .

Assume that $\theta = \theta\sigma$. For any $X = t \in \sigma$, it holds that $\theta(X) \equiv \theta(\sigma(X)) \equiv \theta(t)$. Hence, θ is a unifier of σ .

Assume that θ is a unifier of σ . For $X = t \in \sigma$, we should show that $X = \theta(t) \in \theta$ or that $X \equiv \theta(t)$. This follows straightforwardly from the fact that θ is a unifier of X , and hence, that $\theta(X) \equiv \theta(t)$.

For $X \in \text{dom}(\theta) \setminus \text{dom}(\sigma)$, it holds that $X = \theta(X) \in \theta\sigma$ and $X = \theta(X) \in \theta$. □

Using the above lemmas, the next proposition can be proven. It reformulates proposition 7.2.

Proposition A.2 *Let E be an equality set.*

(a) $FEQ(\Sigma) + \exists(E)$ is consistent iff there exists a (positive) solved form E_s of E .

(b) Let E, E_s be equality sets based on Σ . E_s is a (positive) solved form of E iff E_s is in (positive) solved form and $\models_{\Sigma} \forall(E_s \leftrightarrow E)$.

(c) For any equality set E , either $FEQ(\Sigma) \models \exists(E)$ or $FEQ(\Sigma) \models \forall(\leftarrow E)$.

Proof (a) is a straightforward implication of proposition A.1 and lemma A.1.

(b) Because a positive solved form is a solved form, it suffices to proof this theorem for solved forms.

Assume that E_s is in solved form and $\models_{\Sigma} \forall(E \leftrightarrow E_s)$. By lemma A.2, E_s is a unifier of E . Let σ be any idempotent unifier of E . By lemma A.2, $\models_{\Sigma} \forall(\sigma \rightarrow E)$. Hence, $\models_{\Sigma} \forall(\sigma \rightarrow E_s)$. By application of lemma A.2 and lemma A.3, E_s is more general than σ . Hence, E_s is a solved form of E .

Vice versa, assume that E_s is a solved form of E . By lemma A.2, $\models_{\Sigma} \forall(E_s \rightarrow E)$. Because E_s is in solved form, $FEQ(\Sigma) + \exists(E_s)$ is consistent (lemma A.1). Therefore $FEQ(\Sigma) + \exists(E)$ is consistent. By proposition A.1, E has a solved form σ . Because E_s is an mgu, E_s is more general than σ . By lemma A.3, σ is a unifier of E_s and hence, using lemma A.2, $\models_{\Sigma} \forall(\sigma \rightarrow E_s)$. Summarising we have:

$$\models_{\Sigma} \forall(E_s \rightarrow E)$$

$$\models_{\Sigma} \forall(\sigma \leftrightarrow E)$$

$$\models_{\Sigma} \forall(\sigma \rightarrow E_s)$$

It follows that $\models_{\Sigma} \forall(E_s \leftrightarrow E)$.

(c) is a trivial consequence of proposition A.1 and lemma A.1. □

In a personal communication, Krzysztof Apt showed that the proof of (b) is trivial in case Σ contains an infinite number of functors. With an infinite Σ , $FEQ(\Sigma)$ is a complete theory [40]. In that case, the formula $\forall(E_s \rightarrow E)$ is entailed iff one model of $FEQ(\Sigma)$ can be found which satisfies it. The Herbrand pre-interpretation is such a model; this is proven e.g. in [35].