A class of programs for which SLDNF resolution and NAF rule are complete

Stefano Baratella
Dipartimento di Matematica, Università degli Studi di Trento, Povo, I-38050 Trento, Italy

Communicated by M. Nivat
Received October 1991
Revised April 1992

Abstract
Baratella, S., A class of programs for which SLDNF resolution and NAF rule are complete, Theoretical Computer Science 120 (1993) 83–99.

In this paper, we prove completeness of SLDNF resolution and NAF rule for the class of allowed and locally stratified programs satisfying the further condition of well-behavedness. Well-behavedness imposes a computational restriction on programs whose aim is to ensure that the 3- and 2-valued consequences of Clark's completion of a program belonging to the above-mentioned class do coincide. Then one can apply Kunen's 3-valued completeness theorem in order to get a 2-valued completeness result.

0. Introduction

In [13], Kunen stresses his belief on the necessity for a logic program of having a declarative semantics. Among the attempts of modelling in a declarative way, the procedural behaviour of a logic program, the one presented in [14], has so far been the most successful. Such an attempt is based on the use of Lukasiewicz's 3-valued logic: it is natural to use a third truth value u (undefined) in order to describe a computation that fails to halt. The reader can also refer [10] for a similar approach.

In [14], Kunen proves that completeness of SLDNF resolution and NAF rule holds (in a 3-valued setting) for any allowed program and any allowed query clause. As a consequence of this result, Kunen proves that the SLDNF resolution and NAF rule...
completeness holds (in a 2-valued setting) for the class of allowed programs and query clauses that satisfy the condition of strictness. Loosely speaking, the core of the proof consists in showing that for the above-mentioned class of programs and query clauses, the 3- and 2-valued logical consequences of Clark’s program completion of a program are the same. Thus, the 3-valued completeness result can be applied in order to obtain a completeness result in a 2-valued setting.

In this paper, we present a sufficient condition under which Kunen’s completeness result can be used to prove a 2-valued completeness theorem. We will deal with a class of logic programs and query clauses for which the 3- and 2-valued logical consequences of Clark’s program completion coincide when restricted to universal quantification and negated existential quantification of the body of those clauses. We will work with allowed locally stratified programs and query clauses for which the condition of well-behavedness holds (see Section 2 for the definitions).

A 2-valued completeness result for a class of programs and query clauses including the former one was conjectured in [5]. The hypothesis of allowedness is introduced here in order to avoid the phenomenon of floundering, but it has some other relevant consequences: for instance, it drastically restricts the kind of positive answers computed by a program. Local stratification ensures 2-valued consistency of the program completion, while well-behavedness permits one to transform every 3-valued model of the program completion into a 2-valued model of it which, intuitively, completes the partial information given by the 3-valued model and also respects further conditions.

Loosely speaking, well-behavedness requires atoms which violate strictness to be SLDNF-decidable, i.e., those atoms must have either an SLDNF refutation or an SLDNF finite failure tree. So well-behavedness introduces computational restrictions in addition to the syntactical ones provided by allowedness and local stratification. For this reason, well-behavedness is not easily detectable.

On the other hand, following [5], we believe that it is important to narrow the gap between the class of programs and query clauses for which SLDNF resolution and NAF rule are known to be complete and the class for which the completeness actually holds.

The present author has proved in [3] that SLDNF resolution and NAF rule are 2-valued-complete for the class of programs and query clauses that are allowed, semi-strict and well behaved, and it is plausible that completeness still holds if in the above hypothesis we replace semi-strictness by local call-consistency.

The paper is mostly self-contained. However, the reader is referred to [8] for a complete survey on the classes of locally consistent, locally stratified and strict programs (and many others). There the authors also provide several examples and investigate in detail the mutual relationships among the various classes.

The structure of the paper is the following: in Section 1 we briefly review the semantical aspects of Łukasiewicz’s 3-valued logic, referring the reader to [1] for a syntactical presentation of the system.

Section 2 contains a critical discussion of some aspects of logic programming that hardly have a logical flavour, together with the main definitions. In Section 3, our
Completeness of SLDNF resolution and NAF rule is proved. Section 4 is devoted to a brief conclusion. The appendix (Section 5) contains some of the proofs.

1. Syntax and semantics

For the logic-theoretic and model-theoretic background, see [4] and [7], respectively. We will refer to them for most of the notation and terminology. However, in this section we briefly review the main definitions, in an attempt to make this paper as much self-contained as possible. We also introduce some additional notation and definitions.

Throughout this paper $\Omega$ will denote the class of all ordinal numbers, $\omega$ the first limit ordinal and $\aleph_1$ the first uncountable ordinal.

We will work with ordinary 2-valued logic as well as Lukasiewicz's 3-valued logic. Such a logic has three truth values $t$ (true), $f$ (false) and $u$ (undefined). $t$ and $f$ correspond to the classical truth values. Only in this section, we assume the truth values to be totally ordered as follows:

$$f < u < t.\$$

An operation $\neg$ is defined on these truth values. It behaves like classical negation on $\{t, f\}$, while $\neg u = u$.

We can now introduce the connectives $\land$ (and), $\lor$ (or), $\rightarrow$ (implication) and give their truth tables in the following way. Let $v, w \in \{t, f, u\}:

$$v \land w = \min \{v, w\},$$
$$v \lor w = \max \{v, w\},$$
$$v \rightarrow w = \begin{cases} \neg v \lor w & \text{if } v \geq w \\ t & \text{otherwise.} \end{cases}$$

Note, in particular, that $u \rightarrow u$ has truth value $t$. We will assume our language $\mathcal{L}$ defined in advance. $\mathcal{V}, \mathcal{P}, \mathcal{F}$ are the sets of $\mathcal{L}$-variables, $\mathcal{L}$-predicate symbols and $\mathcal{L}$-terms, respectively.

In the sequel we shall often commit the abuse of saying predicate (variable, constant, function) in place of predicate (variable, constant, function) symbol.

**Definition 1.1.** An $\mathcal{L}$-preinterpretation $A$ consists of

1. a nonempty set $A$, called universe or domain;
2. an element $c^A \in A$ for every $\mathcal{L}$-constant symbol $c$; and
3. an n-ary function $f^A : A^n \rightarrow A$ for every n-ary function symbol $f$, $n > 0$.

An $\mathcal{L}$-structure ($\mathcal{L}$-interpretation) based on $A$ has, in addition,

4. a function $p^A : A^n \rightarrow \{t, f, u\}$ for every n-ary predicate symbol $p.$
So an ordinary 2-valued structure can be viewed as a 3-valued structure where the value $\bot$ is never taken. As usual, the domain of a structure is denoted by the light-upper case letter of the name of the structure. In the sequel, we will stick to this convention without further notice. Given a 3-valued $\mathcal{L}$-structure $A$ and an assignment of values to variables

$$\sigma: \mathcal{U} \rightarrow A,$$

the interpretation of a term $t$ in $A$ under the assignment $\sigma$ (notation $t^{A,\sigma}$) is defined exactly as in the 2-valued case. The truth value of an atomic formula $p(t_1, \ldots, t_n)$ (notation $p(t_1, \ldots, t_n)^{A,\sigma}$) is defined as follows: $p(t_1, \ldots, t_n)^{A,\sigma} = p^{A}(t_1^{A,\sigma}, \ldots, t_n^{A,\sigma})$. If we are dealing with a language $\mathcal{L}$ with equality, we must also define $(t_1 = t_2)^{A,\sigma}$, where $t_1$ and $t_2$ are $\mathcal{L}$-terms, this is done in the obvious way.

Note that equality is treated as 2-valued. The propositional connectives are treated according to their truth tables.

For the quantified formulas:

$$(\forall x \varphi)^{A,\sigma} = \min \{ \varphi^{A,\tau}: \tau: \mathcal{U} \rightarrow A \text{ and } \tau(y) = \sigma(y) \text{ for all } y \neq x \},$$

$$(\exists x \varphi)^{A,\sigma} = \max \{ \varphi^{A,\tau}: \tau: \mathcal{U} \rightarrow A \text{ and } \tau(y) = \sigma(y) \text{ for all } y \neq x \}. $$

We will also use the notation $A \models_{3} \varphi$ for $\varphi^{A,\sigma}$. As usual, $\models_{i}$ will denote the logical consequence relation in $i$-valued logic, $i = 2$ or $3$.

The reader is referred to [1] for a proof-theoretical analysis of propositional Lukasiewicz's 3-valued logic, including an Hilbert type representation as well as a cut-free Gentzen type formulation of the system.

2. Some remarks on programs and computations

Since we are mostly interested in the logical aspects, we do not want to enter in detail on the computational issues and/or problems. A good source for some of these aspects is [16]. See also [11] for all the (indeed many) nonlogical features of the theory of logic programming.

A literal is an atomic formula or a negated atomic formula. A program clause is of the form $\varphi(\lambda_1, \ldots, \lambda_n)$, where $\varphi$ is an atomic formula (head) and $\lambda_1, \ldots, \lambda_n$ are literals (body) and $n \geq 0$. If $n = 0$, we shall write just $\varphi$.

A program is a finite set of program clauses.

A query clause (goal) has the form $\varphi(\lambda_1, \ldots, \lambda_n)$, where $n \geq 0$. If $n = 0$, we write it as $T$ (true).

We shall use $\varphi$ and $\neg$ in place of $\leftarrow$ and $\neg$ in the program clauses and queries to stress the fact that $\varphi$ and $\neg$ are not logical negation and implication. In particular, $\neg$ is only a procedural approximation of logical negation, known as negation as (finite) failure (NAF). Loosely, NAF works as follows: if all the computations of a given program $P$ on a closed atom $x$ fail in a finite amount of time (or, in other
words, if, using linear resolution as the unique computational rule, all the attempts of proving that \( \alpha \) is a logical consequence of \( P \) fail in a finite amount of time, then the system is allowed to infer \( \neg \alpha \) from \( P \).

Roughly speaking, we might say that we are interested in finding suitable conditions on the programs under which \( \neg \) behaves like \( \neg \), for only in that case we have a chance of describing in a declarative way the procedural behaviour of a logic program.

In some cases a good candidate for the declarative semantics of a program \( P \) is its Clark's completion, denoted by \( comp(P) \) (see [16] for the definition). The choice of \( comp(P) \) is motivated by the fact that \( P \) computes more than it says. To be less vague, under NAF, a program can compute also (closed) negative literals of the form \( \neg \alpha \) (in the current terminology one says that \( P \) succeeds on \( comp(P) \) iff \( P \) finitely fails on \( \alpha \)).

On the other hand, if we view \( P \) as a first-order theory in classical logic made of universally quantified formulas in clausal form, no negated atomic formula follows logically from it. So we enrich \( P \), intuitively by replacing, at the declarative level, the if's in the clauses with iff's (!) and we focus our interest on the logical consequences of \( comp(P) \).

As suspected, this operation is not painless: under the translation sending \( \neg \) and \( \neg \) into \( \land \) and \( \land \), respectively, \( P \) is a consistent theory in classical 2-valued logic, whereas \( comp(P) \) is not, always. The formal definition of Clark's completion can be found in [16]. We recall here that \( comp(P) \) comes equipped with a theory of equality that imposes the free interpretation of equality. Such a theory is the declarative counterpart of the mechanism of uniﬁcation that is part of resolution.

It is well known that if \( P \) is any program, then the domain of any model of \( comp(P) \) contains an isomorphic copy of its Herbrand base (HB) and the restriction of \( \neg \) to (the copy of) the Herbrand base is the identity relation (see [16, Chapter 3]). The next result shows that it is not restrictive to consider only models of \( comp(P) \) on whose whole universes \( = \) is the identity relation.

**Theorem 2.1.** Let \( P \) be a program and let \( M \models comp(P) \). Then there exists a structure \( N \) such that

1. \( = \) is the identity relation on \( N \),
2. there exists a surjective function \( h: M \rightarrow N \) such that, for every formula \( \phi \) and for every assignment of values \( \sigma: V \rightarrow M \) if we let \( \bar{\sigma} = h \circ \sigma \), then

\[
\phi^M,\sigma = \phi^N,\bar{\sigma}.
\]

In particular, \( N \models comp(P) \).

**Proof.** Let \( M \models comp(P) \). We define

\[
a \approx b \iff M \models a = b.
\]
Since $M$ is a model of the equality theory of $\text{comp}(P)$, $\approx$ is an equivalence relation on $M$. Let $[a]$ be the $\approx$-equivalence class of $a \in M$. We define $N$ as follows:

1. $N = \{[a] : a \in M\}$;
2. $c^N = [c^M]$ for every constant symbol $c$;
3. $f^N([a_1], \ldots, [a_n]) = [f^M(a_1, \ldots, a_n)]$ for every function symbol $f$ and all $a_1, \ldots, a_n \in M$;
4. $p^N([a_1], \ldots, [a_n]) = p^M(a_1, \ldots, a_n)$ for every predicate symbol $p$ and all $a_1, \ldots, a_n \in M$.

$f^N$ and $p^N$ are well defined because $M$ satisfies the equality theory of $\text{comp}(P)$. We define

$$h : M \rightarrow N,$$

$$a \mapsto [a].$$

Note that $h$ is injective on the copy of $\text{HB}$ contained in $M$ and that, by definition of $\approx$, $\approx$ behaves like identity on $N$.

A routine induction argument on the complexity of a term shows that, for every term $t$ and every $\sigma : \forall \rightarrow M$, if we define $\bar{\sigma}$ as in the statement of the theorem, then

$$t^N,\sigma = [t^M,\sigma].$$

Also, by induction on the complexity of a formula, one shows that, for every $\sigma$ and $\bar{\sigma}$ as above,

$$\phi^M,\sigma = \phi^N,\bar{\sigma}. $$

In particular, $\psi^M = \psi^N$, for every sentence $\psi$, from which it follows that $N \models \text{comp}(P) \square$

We mentioned that in some cases, $\text{comp}(P)$ is a good candidate for the declarative semantics of $P$. This statement has to be understood as follows: for some classes of programs (for instance, the programs with no negative literals in the bodies of their clauses), a result of soundness and completeness of SLDNF resolution and NAF holds with respect to the program completion. Namely, the answers computed by a logic program on a given query clause (whatever an answer is) are those and only those that logically follows from the program completion (or, according to Gödel’s completeness theorem, those that can be proved from the program completion in first-order classical logic).

Unfortunately, this is not true, in general, in presence of clauses with negative literals in their bodies. Consider, for instance, a program $P$ whose completion is inconsistent. $\text{comp}(P)$ proves every formula, while $P$ does not both succeed and fail on the same query clause. Indeed, there are also some more subtle problems. They are discussed in detail in [5].

We introduce now our first restriction on clauses and programs.
Completeness of SLDNF resolution and NAF rule

Definition 2.2. A (program or query) clause is allowed if every variable occurring in it occurs in some positive literal in the body. A set \( S \) of (program or query) clauses is allowed if every clause in it is allowed.

In this context, allowedness is introduced in order to avoid the computational phenomenon of floundering, caused by the inadequacy of NAF in the treatment of negative literals with free variables, but it has indeed other relevant consequences, as shown in [16, 5].

We give now some other definitions. Then, we introduce the class of programs and query clauses which we consider in Section 3.

Definition 2.3. Let \( P \) be a program and \( A \) an \( \mathcal{L} \)-structure. We let

\[
B_A = \{ p(a_1, \ldots, a_n) : P \text{ has arity } n, n \in \omega \text{ and } (a_1, \ldots, a_n) \in A^n \}.
\]

If \( A \) is a Herbrand structure (i.e. one whose universe is the set of all closed \( \mathcal{L} \)-terms), we shall use \( HB \) for \( B_A \).

Note that \( B_A \) also depends on \( P \). So, a better notation would have been \( B_{A, P} \). There is no ambiguity in omitting the \( P \) if we are dealing with a single program \( P \).

The following two definitions are taken from [13].

Definition 2.4. Let \( P \) be a program and let \( \alpha \) and \( \beta \) be atoms. \( \alpha \triangleright \beta \) means that there exists an instance (under a substitution) of a clause in \( P \) with head \( \alpha \) and an occurrence of \( \beta \) in the body. \( \triangleright \) is the reflexive and transitive closure of \( \triangleright \). \( \triangleright \) is the strict partial order defined by: \( \alpha \triangleright \beta \) if \( \alpha \triangleright \beta \) and not \( \beta \triangleright \alpha \).

We also need the notion of signed dependency \( \triangleright_i \), where \( i \) takes value \( +1 \) or \( -1 \).

We let \( \alpha \triangleright_{+1} \beta \) (or \( \alpha \triangleright_{-1} \beta \)) iff there exists an instance of a clause in \( P \) with head \( \alpha \) and a (negated) occurrence of \( \beta \) in the body.

\( \triangleright_i \) is extended as follows: \( \triangleright_i \) is the least relation on the set of atoms such that

\[
\alpha \triangleright_{+1} \alpha \text{ and } (\alpha \triangleright_{+1} \beta \text{ and } \beta \triangleright_{+1} \gamma) \Rightarrow \alpha \triangleright_{i+1} \gamma, \quad \text{with } i, j \in \{-1, +1\}.
\]

Remark 2.5. The notion of signed dependence relative to a program \( P \) can be defined also between elements of \( B_A \), where \( A \) is any \( \mathcal{L} \)-preinterpretation (one simply thinks of assignments of values to variables in the domain of the preinterpretation, in place of syntactical substitutions).

Definition 2.6. Let \( \psi \) be the body of a clause and let \( \beta \) be any atom. We let \( \psi \triangleright_1 \beta \) iff there exists some atom \( \alpha \) for which at least one of the following holds:

1. there exists a positive occurrence of \( \alpha \) in \( \psi \) and \( \alpha \triangleright_1 \beta \);
2. there exists a negative occurrence of \( \alpha \) in \( \psi \) and \( \alpha \triangleright_{-1} \beta \).

Note that Remark 2.5 also applies to the last definition.
Definition 2.7. A local level mapping of a program $P$ is a mapping from $HB$ to $N$. We refer to the value of a closed atom $B$ under this mapping as the level of $B$.

Definition 2.8. A program $P$ is locally stratified if it has a local level mapping $lev$ such that, for all atom $B, C \in HB$, the following hold:
1. if $B \geq C$ then $lev(B) \geq lev(C)$;
2. if $B \geq ^{-1}C$ then $lev(B) > lev(C)$.

$P$ is locally call-consistent if it has a local level mapping satisfying (1) and the following:
3. if $B \geq ^{-1}C$ and $B \geq ^{+}C$ then $lev(B) > lev(C)$.

In [2], the present author has proved that the following proposition holds.

Proposition 2.9. Let $P$ be a locally call-consistent program and let $A$ be an $i$-valued structure $(i = 2$ or $3)$ that is a model of the equality theory of $comp(P)$. Then there exists a map $lev$ such that conditions (1) and (3) of Definition 2.8 hold for all $B, C \in B_A$.

The proof of the above proposition is based on the fact that the equality theory of $comp(P)$ imposes the free interpretation of equality. Indeed, the same proof gives also the following (see [2]) proposition.

Proposition 2.10. Let $P$ be a locally stratified program and let $A$ be an $i$-valued structure $(i = 2$ or $3)$ that is a model of the equality theory of $comp(P)$. Then there exists a map $lev$ such that conditions (1) and (2) of Definition 2.8 hold for all $B, C \in B_A$.

Definition 2.11. An atom $x$ is an anti-instance of another atom $x'$ if $x'$ is an instance of $x$.

We say that $x$ is the least common anti-instance of a (possibly infinite) set $S$ of atoms having all the same predicate symbol (notation $lca(S)$) if
1. $x$ is an anti-instance of each element of $S$;
2. any other atom satisfying (1) is an anti-instance of $x$.

It can be shown (see [15]) that the $lca$ of a set, if existing, is unique up to variable renaming.

Definition 2.12. Let $x$ be an atom. We denote by $pred(x)$ the predicate symbol occurring in $x$. Let $S$ be a set of atoms. We define $L(S)$ as follows. Let $p \in \mathcal{P}$. Let $S_p = \{ x \in S: \text{pred}(x) = p \}$ and $anti_p = lca(S_p)$. Then

$L(S) = \{ \text{anti}_p: p \text{ is a predicate symbol occurring in } S \}$.

Definition 2.13. Let $P$ be a program, and $\varphi$ a query clause. Let $PN$ be the set of all $C \in HB$ such that there exists some closed instance $\varphi'$ of $\varphi$ on closed $\mathcal{L}$-terms for which $\varphi' \geq ^{-1}C$ and $\varphi' \geq ^{+}C$. 
We say that $P \cup \{ \varphi \}$ is well-behaved if for each $\alpha \in L(PN)$, $P \cup \{ \varphi \}$ has either an SLDNF-finite failure tree or an SLDNF refutation with empty substitution as computed answer.

Roughly speaking, $P \cup \{ \varphi \}$ is well behaved if, for every closed instance $\varphi'$ of $\varphi$, every closed atom which violates the condition of strictness introduced in [14] relative to $P \cup \{ \varphi' \}$ is SLDNF-decidable. Indeed, the condition of well-behavedness requires more for the presence of the lca operator.

In the next section we will work with allowed locally stratified programs $P$ and clauses $\varphi$ such that $P \cup \{ \varphi \}$ is well behaved and allowed.

As an example, consider the following program:

$$P = \{ p(x) \varphi q(c), r(x); p(x) \varphi \sim q(c), r(x); q(d) \varphi p(c); r(c) \},$$

where $c$ and $d$ are two constant symbols. Let $\varphi p(x)$ be the query clause. It is easy to check that $P$ is locally stratified and that $P \cup \{ \varphi p(x) \}$ is well behaved and allowed.

Since $\text{comp}(P) \models_2 p(c)$, the completeness result, proved in Section 3, implies the existence of an SLDNF refutation for $P \cup \{ \varphi p(x) \}$ with computed answer $x/c$. In this case the existence of such an SLDNF refutation can be easily verified, but cannot be derived from the 2-valued completeness result proved in [14] since the program $P$ is not even call-consistent, and hence, a fortiori, it is not strict. The reader is referred to [14] for the definition of call-consistency and for its relationship with strictness.

3. Completeness

Throughout this section we assume the following partial order on truth values: $u < f$, $u < t$. The reason for defining such a partial order lies in Propositions 3.2 and 3.3. Let $A$ and $B$ be two 3-valued $L$-structures based on the same preinterpretation. We let $A \leq B$ iff, for every predicate symbol $p$, $p$ $n$-ary, and for all $a \in A^a$, $p^A(a) \leq p^B(a)$.

We recall now some definitions and results from [14]. Let $Str_3$ be the class of all 3-valued $L$-structures. Let $P$ be a logical program and $\text{comp}(P)$ its Clark's completion.

**Definition 3.1.** $TP : Str_3 \to Str_3$ is the map assigning to every 3-valued $L$-structure $A$, a 3-valued $L$-structure $B$ with same domain, same interpretation of constant, function symbols and equality symbol as $A$, and such that for every $n$-ary predicate symbol $p$ and for every $a \in A^a$,

1. $p^B(a) = t$ iff there is some clause $\varphi$ in $P$ of the form $p(s) \varphi \psi$ and there exists an assignment of values $\theta : \gamma \to A$ such that $s^A, \theta = a$ and $A \models_3 \psi \theta$;
2. $p^B(a) = f$ iff for every clause $\varphi$ in $P$ of the form $p(s) \varphi \psi$ and for every assignment of values $\theta : \gamma \to A$, if $s^A, \theta = a$ then $A \models_3 \neg \psi \theta$;
3. $p^B(a) = u$ otherwise.
Kunen [13, 14] proves the following propositions.

**Proposition 3.2.** $TP$ is monotonic with respect to the partial order $\leq$ defined on $Str_3$.

**Proposition 3.3.** Let $A$ be a 3-valued structure. Then $A \models_3 \text{comp}(P)$ iff $TP(A) = A$.

One easily recognizes Proposition 3.3 as the 3-valued version of the well-known 2-valued result that characterizes models of $\text{comp}(P)$ as fixed points of the immediate consequence operator. As an immediate corollary of Propositions 3.2 and 3.3, we have that the completion of every logic program $P$ is consistent with respect to Łukasiewicz's 3-valued logic. This is not so surprising if we realize that the "worst" (and easiest) example of program whose completion is 2-valued inconsistent, namely $\{p \lor \neg p\}$, has a 3-valued consistent completion.

The most important result in [14] is the following proposition.

**Proposition 3.4 (3-valued completeness of SLDNF resolution and NAF rule).** Suppose $P$ is allowed. Let $\psi$ be the body of an allowed query clause and $\sigma: \forall \rightarrow \exists$ a substitution acting only on the variables occurring in $\psi$ ($\sigma$ can be only the empty substitution if $\psi$ is closed).

If $\text{comp}(P) \models_3 \forall(\psi\sigma)$ then there exists an SLDNF refutation for $P \cup \{\sigma\psi\}$ with computed answer $\sigma$. If $\text{comp}(P) \models_3 \neg\exists\psi$ then there exists an SLDNF finite failure tree for $P \cup \{\sigma\psi\}$.

Let now $\mathcal{C}$ be a class of allowed logic programs such that for all $P$ in $\mathcal{C}$, the 2- and 3-valued logical consequences of $\text{comp}(P)$ are the same when restricted to formulas of the form $\forall\psi$ and $\neg\exists\psi$, where $\psi$ is a finite conjunction of literals. Then Proposition 3.4 yields a 2-valued completeness result for SLDNF resolution and NAF rule with respect to allowed query clauses for the class $\mathcal{C}$. A class $\mathcal{C}$ for which the above-mentioned condition holds is that of strict (allowed) programs, as proved in [14].

What is relevant is that one turns the problem whether SLDNF resolution and NAF rule are complete for a given class of programs into a pure logical problem. Even if the class of programs that we are dealing is defined in terms of computational matters, as in our case, Kunen's 3-valued completeness result can be applied if the class satisfies the purely logical condition mentioned above.

In this section, our purpose is to prove that SLDNF resolution and NAF rule are 2-valued-complete for the class of allowed locally stratified programs and query clauses, under the additional assumption of well-behavedness. We will do this by using Kunen's indirect technique.

From here on, $P$ will be a locally stratified program and $A$ a 3-valued $\mathcal{L}'$-structure. Let $\text{lev}$ be a map as in Proposition 2.3.

**Definition 3.5.** Let $\gamma$ be a countable ordinal. We let

$$B_\gamma = \{ B \in B_A : \text{lev}(B) = \gamma \} \quad \text{and} \quad B_\eta = \bigcup \{ B_\eta : \eta < \gamma \}.$$
Let $X$ be any set. From here on $\mathcal{P}(X)$ will denote the power set of $X$.

**Definition 3.6.** Let $\gamma$ be a countable ordinal. We define

$$T_\gamma: \mathcal{P}(B_\gamma) \rightarrow \mathcal{P}(B_\gamma)$$

in the following (usual) way: for every $K \in \mathcal{P}(B_\gamma)$, $B \in T_\gamma(K)$ iff $B \in B_\gamma$ and there exists a closed instance on $A$ of a clause in $P$ with head $B$, say $B \not\supset B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$, such that $B_i \in K$ and $C_j \notin K$, for all $1 \leq i \leq m$ and $1 \leq j \leq n$.

The next remark is rather general. It says that, under certain assumptions, a subset $K$ of the set $B_M$ relative to a given $\mathcal{L}$-structure $M$ (see Definition 2.3) yields, in a natural way, a model of $\text{comp}(P)$.

**Remark 3.7.** Let $M$ be an $\mathcal{L}$-structure that is a model of the equality theory of $\text{comp}(P)$ and such that $=$ is the identity relation of $M$. Let $K \subseteq B_M$ be such that $K \cap B_\gamma$ is a fixed point of $T_\gamma$ for every countable ordinal $\gamma$ (where now $B_\gamma$ is that relative to the domain $M$ of $M$). Note that the condition of local stratifiability suffices to ensure the existence of fixed points for the map $T_\gamma$.

Let us define $N$ having the same domain, same interpretation of equality, constant and function symbols as $M$. By the characterization of models of $\text{comp}(P)$ as fixed points of the immediate consequence map, we have, by interpreting all $\mathcal{L}$ predicate symbols in the following way: for every $\omega$, for every nary predicate symbol $p$, and for every $m_1, \ldots, m_n \in M$,

$$p^N(m_1, \ldots, m_n) = t \iff p(m_1, \ldots, m_n) \in K.$$  

$N$ becomes a 2-valued model of $\text{comp}(P)$. The condition that $=$ is the identity relation of $M$ makes sure that all the axioms of the equality theory of $\text{comp}(P)$ are true in $N$.

**Definition 3.8.** Let $\gamma$ be a countable ordinal. Let $K_\gamma$ be any fixed point for $T_\gamma$. We define

$$T_\gamma(K_\gamma): \mathcal{P}(B_\gamma) \times \mathcal{P}(B_\gamma) \rightarrow \mathcal{P}(B_\gamma) \times \mathcal{P}(B_\gamma)$$

as follows: Let $(I_1, J_1) \in \mathcal{P}(B_\gamma) \times \mathcal{P}(B_\gamma)$ and let $(I_2, J_2)$ be $T_\gamma(K_\gamma)((I_1, J_1))$.

Then

1. $B \in I_2$ iff $B \in I_1$ or $B \in B_\gamma$ and there exists a closed instance of a clause in $P$ with head $B$, say $B \not\supset B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$, such that $B_i \in K_\gamma \cup I_1$ and $C_j \notin B_\gamma \setminus K_\gamma$, for all $1 \leq i \leq m$ and $1 \leq j \leq n$.

2. $B \in J_2$ iff $B \in J_1$ or $B \in B_\gamma$ and for every closed instance $B \not\supset B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$ of a clause in $P$ whose head is $B$ there is $i \in \{1, \ldots, m\}$ such that $B_i \in (B, \setminus K_\gamma) \cup J_1$ or there is $j \in \{1, \ldots, n\}$ such that $C_j \in K_\gamma$.

We define a partial order $\leq$ on $\mathcal{P}(B_\gamma) \times P(B_\gamma)$ as follows:

$$(I_1, J_1) \leq (I_2, J_2) \iff I_1 \subseteq I_2 \text{ and } J_1 \subseteq J_2.$$
Note that \((\mathcal{P}(B_\alpha) \times \mathcal{P}(B_\beta), \ll)\) is a complete lattice, with respect to componentwise set-theoretic inclusion and intersection, and that \(T_p(K_x)\) is monotonic with respect to \(\ll\). We shall denote by \(T_p(K_x) \uparrow x\) the usual \(x\)-power of \(T_p(K_x), x \in On\).

Our main result is the following theorem.

**Theorem 3.9.** Let \(P\) be a locally stratified program and let \(\varphi\) be a query clause such that \(P \cup \{ \varphi \}\) is well behaved and allowed. Then 3 and 2-valued logical consequences of \(\text{comp}(P)\) are the same when restricted to formulas of the form \(\forall \varphi\) and \(\neg \exists \varphi\), namely,

1. \(\text{comp}(P) \models_3 \forall \varphi \iff \text{comp}(P) \models_2 \forall \varphi\), and
2. \(\text{comp}(P) \models_3 \neg \exists \varphi \iff \text{comp}(P) \models_2 \neg \exists \varphi\).

Since it is true that every 3-valued logic consequence is also a 2-valued one (3-valued structures where the value \(u\) is never taken, behave exactly as ordinary 2-valued structures), we need to prove only the right-to-left arrows of (1) and (2). From here on we will deal explicitly with (1), for (2) can be proved simply by “dualizing” the proof of (1). The proof of (1) will be by contraposition. So what we will show is the following theorem.

**Theorem 3.10.** Let \(P, \varphi\) be as in the statement of Theorem 3.9. Let \(A\) be a 3-valued model of \(\text{comp}(P)\) such that \(=\) is the identity relation on \(A\) (note that this condition is not restrictive by Theorem 2.1). Assume that \((\forall \varphi)^A = f\) or \(u\). Then there is a 2-valued model \(B\) of \(\text{comp}(P)\) such that \((\forall \varphi)^B = f\).

Before proving the theorem, we need some preliminary results. Let \(\varphi\) be a conjunction of literals and let \(\varphi'\) be a closed instance on \(A\) of \(\varphi\) such that \((\varphi')^A = u\). We define

\[
U_f = \{ B \in B_A : B^A = u \text{ and } \varphi' \geq_1 B \}
\]

and

\[
U_t = \{ B \in B_A : B^A = u \text{ and } \varphi' \geq_0 B \}.
\]

\(U_f \cap U_t = \emptyset\), by Lemma A.1 (see the appendix).

For every countable ordinal \(\delta\) we define

\[
U_{f, \delta} = \{ B \in U_f : \text{lev}(B) = \delta \} \quad \text{and} \quad U_{t, \delta} = \{ B \in U_t : \text{lev}(B) = \delta \}.
\]

Similarly, let

\[
F_\delta = \{ B \in B_A : B^A = f \text{ and } \text{lev}(B) = \delta \}
\]

and

\[
T_\delta = \{ B \in B_A : B^A = t \text{ and } \text{lev}(B) = \delta \}
\]

and, finally,

\[
F_\delta^* = U_{f, \delta} \cup F_\delta \quad \text{and} \quad T_\delta^* = U_{t, \delta} \cup T_\delta.
\]
We can now prove the following result showing that an induction on the countable ordinals allows us to transform a 3-valued model of $\text{comp}(P)$ (so, essentially a fixed point for the 3-valued immediate consequence map) into a 2-valued model of $\text{comp}(P)$ (i.e., into a fixed point for the 2-valued immediate consequence map) in such a way that the final model agrees with the initial one on the atoms that had already a truth value $t$ or $f$ and that, moreover, assigns to certain atoms that had truth value $u$, a prescribed truth value in the set $\{f, t\}$.

**Theorem 3.11.** Under the assumptions of the previous theorem, for every countable ordinal $\delta$ there exists a fixed point $K_\delta$ for the map $T^\delta_P$ such that

(a) $\cup\{T^\gamma_P : \gamma < \delta\} \subseteq K_\delta$,

(b) $(\cup\{F^\gamma_P : \gamma < \delta\}) \cap K_\delta = \emptyset$,

(c) for all $\gamma < \delta$, $K_\gamma \cap B_\delta = K_\gamma$.

**Proof.** Induction on $\delta$. Let $K_0 = \emptyset$.

If $\delta$ is a limit, we let $K_\delta = \cup\{K_\gamma : \gamma < \delta\}$. By definition of $T^\delta_P$, the set $K_\delta$ is a fixed point for such a map (here we make essential use of the fact that the sets $K_\gamma$, for $\gamma < \delta$, are fixed points for the corresponding maps and that every clause is a finite syntactical object).

The conditions (a) and (c) are easily verified. If it were that (b) does not hold, then there would be some $\gamma < \delta$ such that $F^\gamma_P \cap K_\delta \neq \emptyset$. Then, by (c), $F^\gamma_P \cap K_{\gamma+1} \neq \emptyset$, so contradicting the inductive hypothesis.

Let $\delta$ be a successor ordinal, say $\delta = \gamma + 1$. Let

$$(T_\gamma, 0, F_\gamma, 0) = (T^\gamma_P, F^\gamma_P)$$

and

$$(T_\gamma, \sigma, F_\gamma, \sigma) = T_P(K_\gamma) \sigma \eta((T_\gamma, 0, F_\gamma, 0)).$$

Since $T_P(K_\gamma)$ is monotonic on a complete lattice, and since

$$(T_\gamma, 0, F_\gamma, 0) \subseteq T_P(K_\gamma)((T_\gamma, 0, F_\gamma, 0)),$$

there exists an ordinal $\xi$ such that $(T_\gamma, \sigma, F_\gamma, \sigma)$ is a fixed point for $T_P(K_\gamma)$.

We let $K_\delta = K_\gamma \cup (B_\gamma \setminus F_\gamma, \sigma)$. Since $T_\gamma, \sigma \cap F_\gamma, \sigma = \emptyset$ (see Lemma A.4), $K_\delta$ satisfies (a)--(c).

For, first of all, $K_\delta$ is a fixed point for the map $T_P(K_\delta)$ because for every element $B$ in $B_\gamma \setminus F_\gamma, \delta$ there is a clause that, duly instantiated, has its head equal to $B$. Also, the conditions (a) and (b) are satisfied merely by definition of $K_\delta$. Condition (c) holds because nothing has been changed on the construction performed at the previous stages.

Now, by inductive hypothesis, $K_\gamma$ is a fixed point for $T^\gamma_P$, so in order to show that $K_\delta$ is a fixed point for $T^\delta_P$, it suffices to show that

$$B_\in (B_\gamma \setminus F_\gamma, \sigma) \Leftrightarrow B_\in T^\delta_P(K_\delta) \text{ for all } B_\in B_\gamma.$$

Let $B_\in (B_\gamma \setminus F_\gamma, \sigma)$. Then, since $B \notin F_\gamma, \sigma$, there is at least one closed instance of a clause in $P$ whose head is $B$, say $B \circ B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_m$, such that $B_\in K_\gamma \cup (B_\gamma \setminus F_\gamma, \sigma)$.
and $C_j \not\in K_j$, for all $1 \leq i \leq m$ and all $1 \leq j \leq n$ (recall that $P$ is locally stratified). Hence $B \in T^p(K)$.

For the converse, let $B \in B_\gamma \cap T^p(K)$. Then there is at least one closed instance of a clause in $P$ whose head is $B$, say $B \leftarrow B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$, such that $B_i \in K$ and $C_j \not\in K_j$, for all $1 \leq i \leq m$ and all $1 \leq j \leq n$. Since no $B_i$ of level $\gamma$ in $K$ belongs to $F_{\gamma, x}$, it follows that $B \in \left( B_\gamma \setminus F_{\gamma, x} \right)$. □

We are now ready to prove Theorem 3.10.

Proof of Theorem 3.10. The only nontrivial case is when $(\forall \varphi)^A = u$. We will say at the end of this proof how to deal with the case when $(\forall \varphi)^A = f$. Hence, we assume $(\forall \varphi)^A = u$.

Let $x = (x_1, \ldots, x_n)$ be the list of variables occurring in $\varphi$. Then there is $a = (a_1, \ldots, a_m) \in A^*$ such that $\varphi' = \varphi[x/a]$ is $u$ in $A$. We construct $U_f$ and $U_t$ relative to $\varphi'$ and we apply Theorem 3.11, so getting a sequence of fixed points $(K_\delta)_{\delta \in \kappa}$, each satisfying the properties (a)-(c) in the statement of Theorem 3.11.

As described in Remark 3.7, we can therefore get a 2-valued model $B$ of $\text{comp}(P)$ from the sequence of fixed points. Note that, as a consequence of conditions (a) and (b) in the statement of Theorem 3.11, $A \leq B$ (so to say, $t$'s and $f$'s of $A$ are preserved in $B$).

We claim that $(\forall \varphi)^B = f$. It suffices to show that $(\varphi')^B = f$. This follows from the definition of $U_f$ and $U_t$ and from the construction of the sequence of fixed points. For, think for a moment $\varphi'$ to be made up of a single atom $B$ instantiated on $A$ and assume $B^A = u$. In the construction of the sequence of fixed points (on which $B$ depends), we have forced to take value $t$ all the instantiated atoms that are undefined in $A$ and on which $B$ depends negatively (putting them in $U_t$).

Also, we have forced to take value $f$ all the instantiated atoms that are undefined in $A$ and on which $B$ depends positively (putting them in $U_f$).

Therefore, since $B$ is a fixed point that preserves the $t$'s and $f$'s of $A$, it follows that there is no closed instance of clause whose head is $B$ and whose body is true in $B$. Hence, $B^B = f$.

Similarly, if $\varphi'$ is $\sim B$, the definition of $U_f$ and $U_t$ and the construction of the sequence of fixed points ensure that there is at least one closed instance of clause whose head is $B$ and whose body is true in $B$. Hence, since $B$ is a fixed point, $B^B = t$ and so $(\neg B)^B = f$.

If $\varphi'$ is a conjunction of literals, one can argue on the conjuncts.

Eventually, if $(\forall \varphi)^A = f$, since there is a closed instance $\varphi'$ of $\varphi$ on $A$ such that $(\varphi')^A = f$, one simply lets $U_f = U_t = \emptyset$ and then proceeds in the construction of the sequence of fixed points as in Theorem 3.11. Since $B$ is a fixed point that preserves the $t$'s and $f$'s of $A$, it turns out that $(\varphi')^B = f$, which implies $(\forall \varphi)^B = f$. □

By the remark between the statements of Theorems 3.9 and 3.10, the proof of Theorem 3.9 is now straightforward.

We conclude this section with our 2-valued completeness result.
Theorem 3.12. Let $P$ be a locally stratified program and $\varphi$ a query clause such that $P \cup \{ \varphi \}$ is well behaved and allowed. Let $\sigma : \mathcal{F} \rightarrow \mathcal{F}$ be a substitution acting only on the variables occurring in $\varphi$ (if $\varphi$ is closed).

1. If $\text{comp}(P) \models_2 \forall \varphi \sigma$ then there exists an SLDNF refutation for $P \cup \{ \varphi \}$ with computed answer $\sigma$.

2. If $\text{comp}(P) \models_2 \neg \exists \varphi$ then there exists an SLDNF-finite failure tree for $P \cup \{ \varphi \}$.

Proof. Straightforward from Theorem 3.9 and Proposition 3.4. $\square$

4. Conclusion

A stronger version of the completeness result we have proved was conjectured in [5], Section 5.41. In [5], the author states that his conjecture, if proved, would significantly narrow the gap between the class of programs and goals for which SLDNF resolution is known to be complete, and the class of programs and goals for which the completeness actually holds. We share his opinion.

We believe that completeness can be proved also for locally call-consistent allowed programs with respect to well behaved and allowed query clauses. In order to apply the same technique of proof used in this paper, one would need a constructive proof of consistency for the completion of a locally call-consistent program. So far there is no such proof (Sato's proof in [17] makes use of Zorn's lemma, while Cavedon's constructive proof in [S] is unfortunately not correct).

We also believe that important completeness results could be obtained by weakening the condition of allowedness on programs. That involves a more adequate treatment of negative literals by (a variant of) SLDNF resolution and is therefore beyond the scope of this paper.

Appendix

In this section we provide the proofs for some of the statements that have been used in Section 3.

Lemma A.1. $U_f \cap U_i = \emptyset$.

Proof. Assume not and let $B \in U_f \cap U_i$. Since $U_f$ and $U_i$ are relative to a 3-valued structure $A$ that is a model of the equality theory of $\text{comp}(P)$, Corollary 2.1 in [3] ensures that it is possible to find a substitution $\sigma : \mathcal{F} \rightarrow \mathcal{F}$ such that $\varphi \sigma \geq_{+} x$ and $\varphi \sigma \geq_{-} x$, for some atom $x$ for which there exists an assignment of values $\tau : \mathcal{F} \rightarrow A$ such that $\varphi \sigma \tau = \varphi'$ and $\tau(x) = B$. So $x$ is an instance of $lca\{ D \in HB : D \in PN \text{ and } \text{pred}(D) = \text{pred}(B) \}$.
Recalling that $\mathbf{P} \cup \{ \varphi \}$ is assumed to be well behaved and invoking soundness of SLDNF resolution and NAF rule with respect to 3-valued logic, it follows that every 3-valued model of $\text{comp}(\mathbf{P})$ is either a model of $\forall \sigma$ or $\neg \exists \gamma$; a contradiction.

Lemma A.2. Under the assumptions and with the notation of Section 3, let $B \in B_\beta$ and $\beta \in \text{On}$. Then $B \in F_{\gamma, \beta}$ iff one of the following holds:

1. $B^A = t$.
2. $B \in U_{\gamma, \beta}$.
3. $\beta > 0$ and there is $\eta < \beta$ such that for every closed instance of clause of $\mathbf{P}$ on $A$ whose head is $B$, say $B \varphi B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$, at least one of the following hold:
   a. there is some $1 \leq j \leq n$ such that $C_j \in K_\gamma$;
   b. there is some $1 \leq i \leq m$ such that $\text{lev}(B_i) < \gamma$ and $B_i \notin F_{\gamma, \eta}$ or $\text{lev}(B_i) < \gamma$ and $B_i \notin K_\gamma$.

Proof. The right to left arrow is straightforward.

Let us prove the other implication by induction on $\beta$. The case $\beta = 0$ is trivial.

If $\beta$ is a limit, then $B \in F_{\gamma, \beta}$ implies $B \in F_{\gamma, \chi}$ for some $\chi < \beta$, so we can apply the inductive hypothesis to $F_{\gamma, \chi}$.

If $\beta$ is a successor, say $\beta = \chi + 1$ and if (1) and (2) do not hold, then, by definition of $F_{\gamma, \chi + 1}$, (3) holds for some $\eta < \beta$.

Lemma A.3. Under the assumptions and with the notation of Section 3, let $B \in B_\beta$ and $\beta \in \text{On}$. Then $B \in T_{\gamma, \beta}$ iff one of the following holds:

1. $B^A = t$;
2. $B \in U_{\gamma, \beta}$;
3. $\beta > 0$ and there is $\eta < \beta$ such that for some closed instance of clause of $\mathbf{P}$ on $A$ whose head is $B$, say $B \varphi B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$, the following hold:
   a. for every $1 \leq i \leq m$, $B_i \in T_{\gamma, \eta} \cup K_\gamma$;
   b. for every $1 \leq j \leq n$, $C_j \notin K_\gamma$.

Proof. Similar to the proof of the previous lemma.

Lemma A.4. Let $\gamma \in \mathbb{N}_1$, $\eta \in \text{On}$ and let $F_{\gamma, \eta}$, $T_{\gamma, \eta}$ be as in Section 3. Then $F_{\gamma, \eta} \cap T_{\gamma, \eta} = \emptyset$.

Proof. Induction on $\eta$. If $\eta = 0$ the claim follows from the assumption that $A \models_3 \text{comp}(\mathbf{P})$ and from Lemma A.1. The case $\eta$ limit is straightforward.

Let $\eta = \xi + 1$. In this case $(F_{\gamma, \eta} \setminus F_{\gamma, \xi}) \cap (T_{\gamma, \eta} \setminus T_{\gamma, \xi}) = \emptyset$, by definition of $K_\gamma$.

So, by inductive hypothesis, it suffices to show that

$$(F_{\gamma, \eta} \setminus F_{\gamma, \xi}) \cap T_{\gamma, \eta} = F_{\gamma, \xi} \cap (T_{\gamma, \eta} \setminus T_{\gamma, \xi}) = \emptyset.$$

Let us show that $F_{\gamma, \xi} \cap (T_{\gamma, \eta} \setminus T_{\gamma, \xi}) = \emptyset$. 

We apply Lemma A.2. Let $B \in T_{\gamma,\eta} \setminus T_{\gamma,\zeta}$. Then there exists a closed instance on $A$ of a clause in $P$, say $B \supseteq B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$, such that, for all $1 \leq i \leq m$, $B_i \in T_{\gamma,\zeta} \cup K$, and, for all $1 \leq j \leq n$, $C_j \notin K$. Thus, by inductive hypothesis, if $\beta > 0$, condition (3) in Lemma A.2 does not hold for any $\mu < \xi$.

It cannot be $B^A = \top$. For, if it were so, by Proposition 3.4 and by the fact that $t$'s and $f$'s in $A$ are preserved in $K$, (see Theorem 3.11), we would get to a contradiction. So condition (1) in Lemma A.2 does not hold.

If it were $B \in U_{f,\zeta}$, let $B \supseteq B_1, \ldots, B_m, \sim C_1, \ldots, \sim C_n$ be the closed instance on $A$ of a clause in $P$ that caused $B \in T_{\gamma,\eta} \setminus T_{\gamma,\zeta}$. Then there must be some positive atoms $B_i$ of level $\gamma$ in the body of such an instance that are undefined in $A$ and that are in $T_{\gamma,\zeta}$.

By definition of $U_{f,\zeta}$, we would have that those $B_i$ are in $F_{\gamma,0} \subseteq F_{\gamma,\zeta}$; a contradiction because, by inductive hypothesis, $T_{\gamma,\zeta} \cap F_{\gamma,\zeta} = \emptyset$. So, also condition (2) of Lemma A.2 does not hold and therefore $B \notin F_{\gamma,\zeta}$.

That proves $F_{\gamma,\zeta} \cap (T_{\gamma,\eta} \setminus T_{\gamma,\zeta}) = \emptyset$.

The proof that $T_{\gamma,\zeta} \cap (F_{\gamma,\eta} \setminus F_{\gamma,\zeta}) = \emptyset$ can be carried out in the same way, by taking $B \in F_{\gamma,\eta} \setminus F_{\gamma,\zeta}$ and by proving that $B$ cannot satisfy any of the three conditions of Lemma A.3 relative to $T_{\gamma,\zeta}$. □

References