# Tight Logic Programs

Esra Erdem and Vladimir Lifschitz
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
Email: {esra,vl}@cs.utexas.edu

**Abstract**

This note is about the relationship between two theories of negation as failure—one based on program completion, the other based on stable models, or answer sets. François Fages showed that if a logic program satisfies a certain syntactic condition, which is now called "tightness," then its stable models can be characterized as the models of its completion. We extend the definition of tightness, and Fages' theorem, to programs with nested expressions in the bodies of rules, and study tight logic programs containing the definition of the transitive closure of a predicate.

## 1 Introduction

This note is about the relationship between two theories of negation as failure—one based on program completion [Clark, 1978], the other based on stable models, or answer sets [Gelfond and Lifschitz, 1988]. François Fages [1994] showed that if a logic program satisfies a certain syntactic condition, which is now called "tightness,"[1] then its stable models can be characterized as the models of its completion. Lifschitz [1996] observed that Fages' theorem can be extended to programs with infinitely many rules and to programs with classical negation [Gelfond and Lifschitz, 1991] if the concept of completion in the statement of the theorem is replaced by its semantic counterpart—the concept of a supported model [Apt *et al.*, 1988]. Fages' theorem was further generalized in [Babovich *et al.*, 2000].

In this paper we show how to extend Fages' theorem to programs with nested expressions in the bodies of rules. A generalization of the completion

[1] The term used by Fages is "positive-order-consistent."

semantics to such programs was proposed in [Lloyd and Topor, 1984], and a similar generalization of the answer set semantics is given in [Lifschitz *et al.*, 1999]. Here is an example. Program

$$\begin{array}{l} p \leftarrow not\ not\ p, \\ p \leftarrow p, q \end{array} \tag{1}$$

contains nested occurrences of negation as failure in the body of the first rule.[2] It belongs to the syntactic class for which our theorem guarantees the equivalence of the answer set semantics to the completion semantics. This program has two answer sets $\emptyset$, $\{p\}$; they are identical to the models of the completion

$$\begin{array}{l} p \equiv \neg\neg p \vee (p \wedge q), \\ q \equiv \bot \end{array} \tag{2}$$

of this program.

A preliminary report on the tightness of programs with nested expressions is published in [Erdem and Lifschitz, 2001].

The second question studied here is the tightness of logic programs containing the definition of the transitive closure of a predicate:

$$\begin{array}{l} tc(x,y) \leftarrow p(x,y), \\ tc(x,y) \leftarrow p(x,v), tc(v,y). \end{array} \tag{3}$$

Such rules are found in many useful programs. Unfortunately, the definition of tightness may be difficult to verify directly for a program containing these rules. We give here a sufficient condition that can make this easier. A preliminary report on this part of the work was presented at the 2001 AAAI Spring Symposium on Answer Set Programming.

The concept of a tight program and Fages' theorem may be interesting for three reasons.

First, the completion semantics and the stable model semantics are among the most widely used definitions of the meaning of negation as failure, and it is useful to know under what condition they are equivalent to each other.

Second, the class of tight programs is important from the perspective of answer set programming. Whenever the two semantics are equivalent, answer sets for the program can be computed by solving the satisfiability problem for its completion. This can be done using a propositional solver, such

[2] The double negation in the first rule of (1) is redundant from the point of view of the completion semantics, but it does affect the program's answer sets. On the other hand, the second rule is redundant from the point of view of the answer set semantics. But, generally, dropping a rule like this can change a program's completion in an essential way.

as SATO[3] or CHAFF[4], and the use of answer set solvers, such as SMODELS[5] and DLV[6], becomes unnecessary [Babovich *et al.*, 2000]. Our generalization of Fages' theorem allows us to apply this method to SMODELS programs containing the { } construct [Niemelä *et al.*, 1999], because such programs can be viewed as a special case of programs with nested expressions [Ferraris and Lifschitz, 2001]. For instance, the first of rules (1) can be written in the syntax of SMODELS as $\{p\}$.

Third, program completion is closely related to the semantics of causal theories [Giunchiglia *et al.*, 2001] and to the input language of the Causal Calculator[7]. Fages' theorem and its generalizations help us understand the relationship between causal theories and the Causal Calculator on the one hand, and logic programs and answer set programming on the other.

We begin by reviewing answer sets, closure, supportedness and completion for programs with nested expressions (Sections 2–3). After extending Fages' definition of tightness to such programs (Section 4), we state our generalization of Fages' theorem (Section 5). As an example, we show how this generalized form applies to a formalization of the $n$-queens problem in the language of SMODELS (Section 6). Then we study the tightness of programs containing the definition of the transitive closure of a relation (Section 7). A logic programming description of the blocks world is used as an example in Section 8. Proofs are collected in Section 9.

## 2 Programs and Answer Sets

This section is a review of the answer set semantics for nondisjunctive programs with nested expressions.

The words *atom* and *literal* are understood here as in propositional logic; we call the sign $\neg$ in a negative literal $\neg A$ *classical negation*, to distinguish it from the symbol for negation as failure (*not*). *Elementary formulas* are literals and the 0-place connectives $\bot$ and $\top$. *Formulas* are built from elementary formulas using the unary connective *not* and the binary connectives , (conjunction) and ; (disjunction). A *(nondisjunctive) rule* is an expression of the form

$$Head \leftarrow Body \tag{4}$$

[3] `http://www.cs.uiowa.edu/~hzhang/sato.html` .
[4] `http://www.ee.princeton.edu/~chaff/` .
[5] `http://www.tcs.hut.fi/Software/smodels/` .
[6] `http://www.dbai.tuwien.ac.at/proj/dlv/` .
[7] `http://www.cs.utexas.edu/users/tag/cc/ccalc.html` .

where *Head* is a literal or $\bot$, and *Body* is a formula.[8] If $Body = \top$, we will drop both the body and the arrow separating it from the head; rules with the body $\top$ are called *facts*. If $Head = \bot$, we will drop the head; rules with the head $\bot$ are called *constraints*.

A *(nondisjunctive logic) program* is a set of rules.

We define when a consistent set $X$ of literals *satisfies* a formula $F$ (symbolically, $X \models F$) recursively, as follows:

- for elementary $F$, $X \models F$ if $F \in X$ or $F = \top$,
- $X \models not\ F$ if $X \not\models F$,
- $X \models (F, G)$ if $X \models F$ and $X \models G$,
- $X \models (F; G)$ if $X \models F$ or $X \models G$.

A consistent set $X$ of literals is *closed under* a program $\Pi$ if, for every rule (4) in $\Pi$, $Head \in X$ whenever $X \models Body$.

Let $\Pi$ be a program without negation as failure. We say that $X$ is an *answer set* for $\Pi$ if $X$ is minimal among the consistent sets of literals closed under $\Pi$. It is easy to see that there can be at most one such set. For instance, the answer set for the program

$$\begin{array}{l} p \\ p \leftarrow p, q \end{array} \tag{5}$$

is $\{p\}$.

The *reduct* $\Pi^X$ of a program $\Pi$ relative to a set $X$ of literals is obtained from $\Pi$ by replacing every maximal occurrence of a formula of the form $not\ F$ in $\Pi$ (that is, every occurrence of $not\ F$ that is not in the scope of another *not*) with $\bot$ if $X \models F$, and with $\top$ otherwise.[9] A consistent set $X$ of literals is an *answer set* for $\Pi$ if it is the answer set for the reduct $\Pi^X$. For instance, $\{p\}$ is an answer set for program (1) since it is the answer set for the reduct (5) of (1) relative to $\{p\}$.

We say that a formula (or a program) is *normal* if it does not contain classical negation. We will sometimes identify a normal formula $F$ with the propositional formula obtained from $F$ by replacing every comma with $\wedge$, every semicolon with $\vee$, and evey occurrence of *not* with $\neg$. It is easy to see that, for any normal formula $F$ and any set $X$ of atoms, $X \models F$ iff $X$ satisfies $F$ in the sense of propositional logic.

[8] In [Lifschitz *et al.*, 1999], the syntax of rules is more general: the head may be an arbitrary formula, in particular a disjunction.

[9] This definition is equivalent to the recursive definition of the reduct given in [Lifschitz *et al.*, 1999].

## 3 Supported Sets and Completion

This section is a review of supported sets and completion for the class of logic programs introduced above.

We say that a set $X$ of literals is *supported by* a program $\Pi$ if for every literal $L \in X$ there exists a rule (4) in $\Pi$ such that $Head = L$ and $X \models Body$.[10] For instance, each of the sets $\{p\}$, $\{\neg q\}$ is supported by the program

$$\begin{array}{l} p \leftarrow not\ \neg q \\ \neg q \leftarrow not\ p \end{array} \tag{6}$$

but their union is not. The set $\{p\}$ is supported by program (1); $\{q\}$ is not.

We are interested in the relationship between the concept of an answer set on the one hand, and the conjunction of the closure and supportedness conditions on the other. According to the following proposition, every answer set is both closed under the program and supported by it.

**Proposition 1** *For any program* $\Pi$ *and any consistent set* $X$ *of literals, if* $X$ *is an answer set for* $\Pi$ *then* $X$ *is closed under and supported by* $\Pi$.

For instance, the answer sets for program (6) are $\{p\}$ and $\{\neg q\}$; each of them is closed under and supported by (6). The answer sets for program (1) are $\emptyset$ and $\{p\}$; each of them is closed under and supported by (1).

The converse, in general, is not true. The easiest counterexample is the program consisting of one rule $p \leftarrow p$. The set $\{p\}$ is closed under this program and supported by it, although it is not an answer set. Tightness, defined in Section 4 below, is a syntactic condition that eliminates programs like this.

Let $\Pi$ be a finite normal program. The "completion" of $\Pi$ is the set of propositional formulas defined as follows. If $A$ is an atom or the symbol $\bot$, by $Comp(\Pi, A)$ we denote the propositional formula

$$A \equiv (Body_1 \vee \cdots \vee Body_k) \tag{7}$$

where the disjunction extends over all rules

$$A \leftarrow Body_i \tag{8}$$

[10] Baral and Gelfond [1994] and Inoue and Sakama [1998] generalize the definition of supportedness from [Apt *et al.*, 1988] in a different direction—they extend it to disjunctive programs.

in $\Pi$ with the head $A$. The *completion* of $\Pi$ is the set of formulas $Comp(\Pi, A)$ for all $A$.[11]

For instance, the bodies of rules (1), written as propositional formulas, are $\neg\neg p$ and $p \wedge q$; for this program $\Pi$, formulas (2) are $Comp(\Pi, p)$ and $Comp(\Pi, q)$. In addition to these two formulas, the completion of this program includes also $Comp(\Pi, \bot)$, which is the tautology $\bot \equiv \bot$.

In application to finite normal programs, the conjunction of closure and supportedness exactly corresponds to the program's completion:

**Proposition 2** *For any finite normal program* $\Pi$*, a set of atoms satisfies the completion of* $\Pi$ *iff it is closed under and supported by* $\Pi$*.*

From Propositions 1 and 2, we conclude:

**Corollary 1** *For any finite normal program* $\Pi$ *and any set* $X$ *of atoms, if* $X$ *satisfies the completion of* $\Pi$ *then* $X$ *is closed under and supported by* $\Pi$*.*

# 4 Tight Programs

To define the concept of a tight program, we need a few auxiliary definitions.

Recall that an occurrence of a formula $F$ in a formula $G$ is *singular* if the symbol before this occurrence is $\neg$; otherwise, the occurrence is *regular* [Lifschitz *et al.*, 1999]. It is clear that the occurrence of $F$ can be singular only if $F$ is an atom. For any formula $G$, by $lit(G)$ we denote the set of all literals having regular occurrences in $G$. For instance, $lit(p; not\ \neg r) = \{p, \neg r\}$.

Any formula $F$ can be written, in exactly one way, as the conjunction $F_1, \ldots, F_n$ $(n \geq 1)$ of its *conjunctive components*—formulas that do not have the form $G, H$. The *positive part* $F^p$ of $F$ is the conjunction of all conjunctive components $F_i$ of $F$ such that at least one occurrence of an atom in $F_i$ is not in the scope of negation as failure. For instance, the positive part of $p, not\ q, r$ (no matter whether the conjunctive terms are grouped to the left or to the right) is $p, r$. The positive part of $(p; not\ q), not\ r$ is $p; not\ q$. The positive part of $p; not\ q; r$ is $p; not\ q; r$. The positive part of $not\ p$ is $\top$.

For any program $\Pi$ and any set $X$ of literals, we say about literals $L, L' \in X$ that $L$ is a *parent* of $L'$ relative to $\Pi$ and $X$ if there is a rule (4) in $\Pi$ such that

[11] This is essentially the definition from [Lloyd and Topor, 1984] restricted to the propositional case. It is restricted to finite programs to avoid the need to use an infinite disjunction in (7).

- $X \models Body$,
- $L \in lit(Body^p)$, and
- $L' = Head$.

For instance, the parents of $p$ relative to the program

$$\begin{array}{l} p \leftarrow not\ q, \\ q \leftarrow not\ p, \\ p \leftarrow p, r \end{array} \tag{9}$$

and the set $\{p, q, r\}$ are $p$ and $r$; on the other hand, $p$ has no parents relative to (9) and the set $\{p, q\}$.

Now we are ready to give the main definition of this paper:

A program $\Pi$ is *tight* on a set $X$ of literals if there is no infinite sequence $L_0, L_1, \ldots$ of elements of $X$ such that for every $i$, $L_{i+1}$ is a parent of $L_i$ relative to $\Pi$ and $X$.

In other words, $\Pi$ is tight on a set $X$ iff the parent relation relative to $\Pi$ and $X$ is well-founded.[12]

For instance, program (1) is tight on $\{p\}$: $p$ does not have parents relative to (1) and $\{p\}$, so the parent relation relative to (1) and $\{p\}$ is well-founded. But that program is not tight on $\{p, q\}$. Indeed, $p$ and $q$ are the parents of $p$ relative to (1) and $\{p, q\}$, so that in the sequence $p, p, \ldots$ every element is followed by its parent.

The proposition below gives an equivalent characterization of tightness:

**Proposition 3** *A program $\Pi$ is tight on a set $X$ of literals iff there exists a function $\lambda$ from $X$ to ordinals satisfying the following condition:*

*(*) for every rule (4) in $\Pi$, if $Head \in X$ and $X \models Body$ then, for all $L \in X \cap lit(Body^p)$, $\lambda(L) < \lambda(Head)$.*

For instance, to show that program (1) is tight on $\{p\}$, we can take $\lambda(p) = 0$. To show that program

$$p \leftarrow (q; not\ r) \tag{10}$$

is tight on $\{p, q, r\}$, take $\lambda(p) = 1$, $\lambda(q) = \lambda(r) = 0$.

If $X$ is finite then the values of $\lambda$ in the statement of Proposition 3 can be assumed to be finite.

[12] A binary relation $R$ is *well-founded* if there is no infinite sequence $x_0, x_1, \ldots$ of elements of its domain such that, for all $i$, $x_{i+1} R x_i$.

Proposition 3 is a special case of the following general fact:

**Lemma from Set Theory** *A binary relation $R$ is well-founded iff there exists a function $\lambda$ from the domain of $R$ to ordinals such that, for all $x$ and $y$, $xRy$ implies $\lambda(x) < \lambda(y)$.*

To compare the definition of tightness above with the definition given earlier in [Babovich *et al.*, 2000], assume that the rules of $\Pi$ have the form

$$Head \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \tag{11}$$

where each $L_i$ is a literal. In this case, condition (*) says: *for every rule (11) in $\Pi$, if*

$$Head \in X, \tag{12}$$

$$L_1, \ldots, L_m \in X \tag{13}$$

*and*

$$L_{m+1}, \ldots, L_n \notin X \tag{14}$$

*then, for all $L \in X \cap \{L_1, \ldots, L_m\}$, $\lambda(L) < \lambda(Head)$.* In view of (13), the intersection $X \cap \{L_1, \ldots, L_m\}$ here can be replaced by $\{L_1, \ldots, L_m\}$. The only difference between this form of condition (*) and the corresponding condition in [Babovich *et al.*, 2000] is the presence of restriction (14).[13] The additional generality given by including (14) can be illustrated by the program

$$p$$
$$q$$
$$p \leftarrow p, not\ q$$

—it is tight on $\{p, q\}$ in the sense of this paper, but not in the sense of [Babovich *et al.*, 2000].

## 5 Fages' Theorem

Our generalization of Fages' theorem is stated as follows:

**Theorem 1** *For any program $\Pi$ and any consistent set $X$ of literals such that $\Pi$ is tight on $X$, $X$ is an answer set for $\Pi$ iff $X$ is closed under and supported by $\Pi$.*

[13] The possibility of including this restriction was suggested to us by Hudson Turner on May 2, 2001.

For instance, program (6) is tight on the sets $\{p\}$ and $\{\neg q\}$, which are closed under and supported by (6). By the theorem above, it follows that $\{p\}$ and $\{\neg q\}$ are the answer sets for (6).

By Proposition 2, we conclude:

**Corollary 2** *For any finite normal program* $\Pi$ *and any set* $X$ *of atoms such that* $\Pi$ *is tight on* $X$*,* $X$ *is an answer set for* $\Pi$ *iff* $X$ *satisfies the completion of* $\Pi$*.*

For instance, program (1) is tight on the models $\emptyset$, $\{p\}$ of its completion (2). In accordance with Corollary 2, these two models are the answer sets for (1).

In the modification of Theorem 1 stated below tightness is replaced by a more restrictive but simpler condition. We say that a literal $L$ is a *positive body element* of a program $\Pi$ if $\Pi$ contains a rule $Head \leftarrow Body$, with $Head \neq \bot$, such that $Body$ contains a regular occurrence of $L$ that is not in the scope of negation as failure. We denote by $pos(\Pi)$ the set of positive body elements of $\Pi$. For instance, if $\Pi$ is (1) then $pos(\Pi) = \{p, q\}$.

**Theorem 2** *For any program* $\Pi$ *and any set* $X$ *of literals disjoint from* $pos(\Pi)$*,* $X$ *is an answer set for* $\Pi$ *iff* $X$ *is closed under and supported by* $\Pi$*.*

By Proposition 2, we conclude:

**Corollary 3** *For any finite normal program* $\Pi$ *and any set* $X$ *of atoms disjoint from* $pos(\Pi)$*,* $X$ *is an answer set for* $\Pi$ *iff* $X$ *satisfies the completion of* $\Pi$*.*

An example of the use of this corollary is discussed in the next section.

## 6 Example: The N-Queens Problem

In the $n$-queens problem, the goal is to find a configuration of $n$ queens on an $n \times n$ chessboard such that no queen can be taken by any other queen. In other words, no two queens may be on the same row, on the same column, or on the same diagonal.

The $n$-queens problem can be described in the language of SMODELS as shown in Figure 1. The first line in this program is shorthand for $n$ facts

```
number(1).
. . .
number(n).
```

```
number(1..n).

1{queen(R,C) : number(R)}1 :- number(C).

:- queen(R,C), queen(R,C1), number(R;C;C1), C < C1.

:- queen(R,C), queen(R1,C1), number(R;C;R1;C1),
   C < C1, abs(R-R1) == abs(C-C1).
```

Figure 1: An SMODELS representation of the $n$-queens problem.

The next line is a "choice rule"; it instructs SMODELS to select atoms of the form `queen(R,C)` for including in an answer set in such a way that, for every column `C`, exactly one atom `queen(R,C)` be selected. The first of the two constraints at the end of Figure 1 prohibits two queens on the same row; `number(R;C;C1)` is shorthand for `number(R)`, `number(C)`, `number(C1)`. The last constraint prohibits two queens on the same diagonal. The answer sets for this program are in a 1–1 correspondence with the possible arrangements of $n$ queens.

Figure 1 can be turned into a program in the sense of Section 2 in two steps. The first step is grounding. The grounded program consists of the rules

$$1\{queen(1, C), \dots, queen(n, C)\}1 \tag{15}$$

for all $C$ in $\{1, \dots, n\}$,

$$\leftarrow queen(R, C), queen(R, C1) \tag{16}$$

for all $R, C, C1$ in $\{1, \dots, n\}$ such that $C < C1$, and

$$\leftarrow queen(R, C), queen(R1, C1) \tag{17}$$

for all $R, R1, C, C1$ in $\{1, \dots, n\}$ such that $C < C1$ and $|R - R1| = |C - C1|$. The second step is rewriting rule (15) in terms of nested expressions, as described in [Ferraris and Lifschitz, 2001, Section 6]:

$$\begin{array}{l} queen(R, C) \leftarrow not\ not\ queen(R, C) \\ \leftarrow queen(R, C), queen(R1, C) \qquad (R < R1) \\ \leftarrow not\ queen(1, C), \dots, not\ queen(n, C). \end{array} \tag{18}$$

The first of these rules allows each of the atoms $queen(R, C)$ to be included or not included in an answer set arbitrarily. The second rule prohibits the

selections that include more than one atom $queen(R, C)$ with the same value of $C$. The third rule prohibits the selections that include no such atoms for some value of $C$.

To sum up, the translation of Figure 1 into the language of logic programs in the sense of this paper is program (16)–(18). (Note the use of nested negations in the first of rules (18).) This is a finite normal program without positive body elements. By Corollary 3, the answer sets for this program are identical to the models of its completion.

This fact can be used to find solutions to the $n$-queens problem by running a satisfiability solver on the program's completion. In our experiments, the completion was generated and clausified by the Causal Calculator. Then SATO found a model of the completion in 0.1 seconds for $n = 20$, and in 0.3 seconds for $n = 30$. This example suggests that using satisfiability solvers to compute answer sets for tight programs may be computationally advantageous. Systematic experimental evaluation of this form of answer set programming is a topic for future research.

## 7 Transitive Closure

In logic programming, the transitive closure $tc$ of a binary predicate $p$ is usually defined by the rules

$$tc(x, y) \leftarrow p(x, y),$$
$$tc(x, y) \leftarrow p(x, v), tc(v, y).$$

If we combine this definition *Def* with any set $\Pi$ of facts defining $p$, and consider the minimal model of the resulting program, the extent of $tc$ in this model will be the transitive closure of the extent of $p$. In this sense, *Def* is a correct characterization of the concept of transitive closure. We know, on the other hand, that the sets of atoms closed under and supported by $\Pi \cup Def$ may be different from the minimal model. In these "spurious" sets of atoms, $tc$ is weaker than the transitive closure of $p$. This is a blemish on the concept of completion as a sematics for logic programs. The absence of "spurious" sets of atoms can be assured by requiring that facts $\Pi$ define relation $p$ to be acyclic.

In this section we study the more general situation when $\Pi$ is a logic program, not necessarily a set of facts. This program may define several predicates besides $p$. Even $tc$ is allowed to occur in $\Pi$, except that all occurrences of this predicate are supposed to be in the bodies of rules, so that all rules defining $tc$ in $\Pi \cup Def$ will belong to *Def*. The rules of $\Pi$

may include negation as failure, and, accordingly, we talk about answer sets instead of the minimal model. Program $\Pi \cup Def$ may have many answer sets. According to Proposition 4 below, the extent of $tc$ in each of these sets is the transitive closure of the extent of $p$ in the same set.

Recall that programs in the sense of Section 2 are propositional objects; there are no variables in them. Expressions containing variables, such as $Def$, can be treated as schematic: we select a non-empty set $C$ of symbols ("object constants") and view an expression with variables as shorthand for the set of all its ground instances obtained by substituting these symbols for variables. It is convenient, however, to be a little more general. In the theorem below, $p$ and $tc$ are assumed to be functions from $C \times C$ to the set of atoms such that all atoms $p(x, y)$ and $tc(x, y)$ are pairwise distinct.

**Proposition 4** *Let $\Pi$ be a program that does not contain atoms of the form $tc(x, y)$ in the heads of rules. If $X$ is an answer set for $\Pi \cup Def$ then*

$$\{\langle x, y\rangle : tc(x, y) \in X\} \tag{19}$$

*is the transitive closure of*

$$\{\langle x, y\rangle : p(x, y) \in X\}. \tag{20}$$

If atoms of the form $tc(x, y)$ do not occur in $\Pi$ at all then the answer sets for $\Pi \cup Def$ are actually in a 1-1 correspondence with the answer sets for $\Pi$.[14] The answer set for $\Pi \cup Def$ corresponding to an answer set $X$ for $\Pi$ is obtained from $X$ by adding a set of atoms of the form $tc(x, y)$. This is easy to prove using the splitting set theorem [Lifschitz and Turner, 1994].

Under what conditions can we assert that the consistent sets of literals closed under and supported by a program containing $Def$ are not "spurious"? As we know from Theorem 1, such a condition is provided by the tightness of the program. The verification of the tightness of programs containing $Def$ is facilitated by the theorem below, which tells us that in some cases the tightness of a program is preserved after adding $Def$ to it.

For any program $\Pi$ and any set $X$ of literals, we say about literals $L, L' \in X$ that $L$ is an *ancestor* of $L'$ relative to $\Pi$ and $X$ if there exists a finite sequence of literals $L_1, \ldots, L_n \in X$ $(n > 1)$ such that $L = L_1$, $L' = L_n$ and for every $i$ $(1 \leq i < n)$, $L_i$ is a parent of $L_{i+1}$ relative to $\Pi$ and $X$. In other words, the ancestor relation is the transitive closure of the parent relation.

[14] This observation is due to Hudson Turner (personal communication, October 3, 2000).

**Theorem 3** *Let $\Pi$ be a program that does not contain atoms of the form $tc(x, y)$ in the heads of rules. For any set $X$ of literals, if*

*(i)* *$\Pi$ is tight on $X$,*

*(ii)* *$\{\langle x, y\rangle : \ p(y, x) \in X\}$ is well-founded, and*

*(iii)* *no atom of the form $tc(x, y)$ is an ancestor of an atom of the form $p(x, y)$ relative to $\Pi$ and $X$,*

*then $\Pi \cup Def$ is tight on $X$.*

By Theorem 1 and Proposition 4, we conclude:

**Corollary 4** *Let $\Pi$ be a program that does not contain atoms of the form $tc(x, y)$ in the heads of rules, and let $X$ be a set of literals satisfying conditions (i)–(iii) from Theorem 3. If, in addition,*

*(iv)* *$X$ is a consistent set closed under and supported by $\Pi \cup Def$*

*then $X$ is an answer set for $\Pi \cup Def$, and*

$$\{\langle x, y\rangle : tc(x, y) \in X\}$$

*is the transitive closure of*

$$\{\langle x, y\rangle : p(x, y) \in X\}.$$

Proposition 2 shows that if $X$ is a set of atoms and $\Pi$ is a finite program without classical negation then condition (iv) can reformulated as follows:

*(iv′)* *$X$ is a model of the completion of $\Pi \cup Def$.*

Condition (ii) in the statement of Theorem 3 is similar to the acyclicity property mentioned in the introduction. In fact, if the underlying set $C$ of constants is finite then (ii) is obviously equivalent to the following condition: there is no finite sequence $x_1, \ldots, x_n \in C$ $(n > 1)$ such that

$$p(x_1, x_2), \ldots, p(x_{n-1}, x_n), p(x_n, x_1) \in X. \tag{21}$$

For an infinite $C$, well-foundedness implies acyclicity, but not the other way around.

Here is a useful syntactic sufficient condition for (ii):

**Proposition 5** *If* $\Pi$ *contains constraint*

$$\leftarrow tc(x, x) \tag{22}$$

*and* $C$ *is finite then, for every set* $X$ *of literals closed under* $\Pi \cup Def$*, set* $\{\langle x, y\rangle : p(y, x) \in X\}$ *is well-founded.*

Without condition (ii), the assertion of the theorem would be incorrect. Program $\Pi$ that consists of one fact $p(1, 1)$, with $C = \{1, 2\}$ and

$$X = \{p(1, 1), tc(1, 1), tc(1, 2)\},$$

provides a counterexample.

Condition (iii) can be verified by checking, for instance, that $p$ does not depend positively on $tc$ in the dependency graph of $\Pi$. This condition is essential as well. Indeed, take $\Pi$ to be

$$p(x, y) \leftarrow tc(x, y).$$

With $C = \{1, 2\}$, set $X = \{p(2, 1), tc(2, 1)\}$ is closed under and supported by $\Pi \cup Def$, but is is not an answer set for $\Pi \cup Def$: the only answer set for this program is empty.

## 8 Example: The Blocks World

As an example of the use of Theorem 3, consider a "history program" for the blocks world—a program whose answer sets represent possible "histories"of the blocks world over a fixed time interval. A history of the blocks world is characterized by the truth values of atoms of two kinds: $on(b, l, t)$ ("block $b$ is on location $l$ at time $t$") and $move(b, l, t)$ ("block $b$ is moved to location $l$ between times $t$ and $t + 1$"). Here

- $b$ ranges over a finite set of "block constants,"
- $l$ ranges over the set of location constants that consists of the block constants and the constant *table*,
- $t$ ranges over the symbols representing an initial segment of integers $0, \ldots, T$,

except that in $move(b, l, t)$ we require $t < T$. One other kind of atoms used in the program is $above(b, l, t)$: "block $b$ is above location $l$ at time $t$". These atoms are used to express constraint (31) that requires every block to be

"supported by the table" and thus eliminates stacks and circular configurations of blocks flying in space.

The program consists of the following rules:[15]

$$\begin{array}{l} on(b,l,0) \leftarrow not\ \neg on(b,l,0) \\ \neg on(b,l,0) \leftarrow not\ on(b,l,0) \\ move(b,l,t) \leftarrow not\ \neg move(b,l,t) \\ \neg move(b,l,t) \leftarrow not\ move(b,l,t) \end{array} \tag{23}$$

$$\begin{array}{l} on(b,l,t) \leftarrow move(b,l,t) \\ on(b,l,t+1) \leftarrow on(b,l,t), not\ \neg on(b,l,t+1) \end{array} \tag{24}$$

$$\neg on(b,l,t) \leftarrow on(b,l',t) \qquad (l \neq l') \tag{25}$$

$$\leftarrow on(b,b'',t), on(b',b'',t) \qquad (b \neq b') \tag{26}$$

$$\leftarrow move(b,l,t), on(b',b,t) \tag{27}$$

$$\leftarrow move(b,l,t), move(b',l',t) \qquad (b \neq b' \text{ or } l \neq l') \tag{28}$$

$$\begin{array}{l} above(b,l,t) \leftarrow on(b,l,t) \\ above(b,l,t) \leftarrow on(b,b',t), above(b',l,t) \end{array} \tag{29}$$

$$\leftarrow above(b,b,t) \tag{30}$$

$$\leftarrow not\ above(b,table,t) \tag{31}$$

To illustrate the use of Theorem 3, in Section 9 we use it to prove the following proposition:

**Proposition 6** *Program (23)–(31) is tight on every set of literals that is closed under it.*

This proposition tells us that the program's answer sets are closed under and supported by the program. It follows that these answer sets can be computed by first eliminating classical negation from the program in favor of new atoms, then generating and clausifying the program's completion, and then running a satisfiability solver on the resulting set of clauses.

[15] This program is similar to the history program for the blocks world from [Lifschitz, 1999]. Instead of rules (23), the program in that paper contains a pair of disjunctive rules; according to Theorem 1 from [Erdem and Lifschitz, 1999], this difference does not affect the program's answer sets. The intuitive meaning of rules (24)–(27) is discussed in [Lifschitz, 1999], Section 6. Rule (28) prohibits concurrent actions. Rules (29) and (30) were suggested to us by Norman McCain and Hudson Turner on June 11, 1999; similar rules are discussed in [Lifschitz, 1999], Section 8.

The idea of the proof is to check first that program (23)–(28), (30), (31) is tight, and then use Theorem 3 to conclude that tightness is preserved when we add the definition (29) of *above*. There are two complications, however, that need to be taken into account.

First, *on* and *above* are ternary predicates, not binary. To relate them to the concept of transitive closure, we can say that any binary "slice" of *above* obtained by fixing its last argument is the transitive closure of the corresponding "slice" of *on*. Accordingly, Theorem 3 will need to be applied $T + 1$ times, once for each slice.

Second, the first two arguments of *on* do not come from the same set $C$ of object constants, as required in the framework of Theorem 3: the set of block constants is a proper part of the set of location constants. In the proof, we will introduce a program similar to (23)–(31) in which, syntactically, *table* is allowed as the first argument of both *on* and *above*.

# 9 Proofs

**Lemma 1** *Given a formula $F$ without negation as failure and two sets $Z, Z'$ of literals such that $Z' \subseteq Z$, if $Z' \models F$ then $Z \models F$.*

**Proof.** Immediate by structural induction.

The following lemma is the special case of Proposition 1 in which $\Pi$ is assumed to be a program without negation as failure.

**Lemma 2** *For any program $\Pi$ without negation as failure and any consistent set $X$ of literals, if $X$ is an answer set for $\Pi$ then $X$ is closed under and supported by $\Pi$.*

**Proof.** Let $\Pi$ be a program without negation as failure and $X$ be an answer set for $\Pi$. By the definition of an answer set for programs without negation as failure, $X$ is closed under $\Pi$. To prove supportedness, take any literal $L$ in $X$. Since $X$ is minimal among the sets closed under $\Pi$, $X \setminus \{L\}$ is not closed under $\Pi$. This means that $\Pi$ contains a rule (4) such that $X \setminus \{L\} \models \mathit{Body}$ but $\mathit{Head} \notin X \setminus \{L\}$. By Lemma 1, $X \models \mathit{Body}$. Since $X$ is closed under $\Pi$, it follows that $\mathit{Head} \in X$, so that $\mathit{Head} = L$.

The definition of the reduct $F^X$ of a formula $F$ is similar to the definition of the reduct of a program given in Section 2.

**Lemma 3** *For any formula $F$, any program $\Pi$, and any consistent set $X$ of literals,*

*(i)* $X \models F$ *iff* $X \models F^X$;

*(ii)* $X$ *is closed under* $\Pi$ *iff* $X$ *is closed under* $\Pi^X$;

*(iii)* $X$ *is supported by* $\Pi$ *iff* $X$ *is supported by* $\Pi^X$.

**Proof.** Part (i) is immediate by structural induction; parts (ii) and (iii) follow from (i).

**Proof of Proposition 1 (Section 3).** Consider a program $\Pi$ and an answer set $X$ for $\Pi$. By the definition of an answer set, $X$ is an answer set for $\Pi^X$. Then, by Lemma 2, $X$ is closed under and supported by $\Pi^X$. By Lemma 3(ii,iii), it follows that $X$ is closed under and supported by $\Pi$.

**Proof of Proposition 2 (Section 3).** Let $\Pi$ be a finite normal program. Recall that the completion of $\Pi$ consists of the equivalences (7) where $A$ is an atom or the symbol $\bot$. It is clear that a set $X$ of atoms satisfies the completion of $\Pi$ iff, for each $A$,

(a) for every rule (8) in $\Pi$ with the head $A$, if $X \models Body_i$ then $A \in X$, and

(b) if $A \in X$ then $X \models Body_i$ for some rule (8) in $\Pi$ with the head $A$.

Condition (a) expresses that $X$ is closed under $\Pi$, and condition (b) expresses that $X$ is supported by $\Pi$.

**Proof of Lemma from Set Theory (Section 4).** The "if" part follows from the well-foundedness of $<$ on sets of ordinals. To prove the "only if" part, consider the following transfinite sequence of subsets of the domain of $R$:

$$\begin{array}{l} S_0 = \emptyset, \\ S_{\alpha+1} = \{x : \ \forall y(yRx \Rightarrow y \in S_\alpha)\}, \\ S_\alpha = \bigcup_{\beta<\alpha} S_\beta \quad \text{if } \alpha \text{ is a limit ordinal.} \end{array}$$

For any $x \in \bigcup_\alpha S_\alpha$, define $\lambda(x)$ to be the smallest $\alpha$ such that $x \in S_\alpha$. From the well-foundedness of $R$ we can conclude that $\bigcup_\alpha S_\alpha$ is the whole domain of $R$.

**Lemma 4** *For any formula* $F$ *and any set* $X$ *of literals*

$$X \models F \text{ iff } X \cap lit(F) \models F.$$

**Proof.** Immediate by structural induction.

The following lemma is the special case of one half of Theorem 1 in which $\Pi$ is assumed to be a program without negation as failure.

**Lemma 5** *Let* $\Pi$ *be a program without negation as failure. For any consistent set* $X$ *of literals such that* $\Pi$ *is tight on* $X$*, if* $X$ *is closed under and supported by* $\Pi$ *then* $X$ *is an answer set for* $\Pi$*.*

**Proof.** Let $\Pi$ be a program without negation as failure and $X$ be a consistent set of literals such that $\Pi$ is tight on $X$. Suppose that $X$ is closed under and supported by $\Pi$. By the definition of an answer set for programs without negation as failure, we need to show that no proper subset of $X$ is closed under $\Pi$. Let $Y$ be a proper subset of $X$. Note first that there exists a literal $L$ in $X \setminus Y$ that does not have a parent in $X \setminus Y$ relative to $\Pi$ and $X$. Indeed, assume that there is no such literal, so that every literal in $X \setminus Y$ has a parent relative to $\Pi$ and $X$ in this set; then there exists an infinite sequence $L_0, L_1, \ldots$ of elements of $X \setminus Y$ such that $L_{i+1}$ is a parent of $L_i$, which contradicts the assumption that $\Pi$ is tight on $X$.

Take such a literal $L$. Since $X$ is supported by $\Pi$, there is a rule

$$L \leftarrow \mathit{Body}$$

in $\Pi$ such that

$$X \models \mathit{Body}. \tag{32}$$

By the definition of the parent relation, the elements of $X \cap \mathit{lit}(\mathit{Body}^p)$ are parents of $L$ relative to $\Pi$ and $X$. By the choice of $L$, no parent of $L$ relative to $\Pi$ and $X$ belongs to $X \setminus Y$, so that $X \cap \mathit{lit}(\mathit{Body}^p)$ is disjoint from $X \setminus Y$. Consequently, $X \cap \mathit{lit}(\mathit{Body}^p) \subseteq Y$. Since $\Pi$ does not contain negation as failure, $\mathit{lit}(\mathit{Body}) = \mathit{lit}(\mathit{Body}^p)$, so that

$$X \cap \mathit{lit}(\mathit{Body}) \subseteq Y. \tag{33}$$

By Lemma 4, we can conclude from (32) that

$$X \cap \mathit{lit}(\mathit{Body}) \models \mathit{Body}.$$

In view of (33), it follows by Lemma 1 that $Y \models \mathit{Body}$. Since $L \in Y$, it follows that $Y$ is not closed under $\Pi$.

**Lemma 6** *For any program* $\Pi$ *and any consistent set* $X$ *of literals, if* $\Pi$ *is tight on* $X$ *then so is* $\Pi^X$*.*

**Proof.** Let $\Pi$ be a program and $X$ be a consistent set of literals. Suppose that $\Pi^X$ is not tight on $X$. We want to show that $\Pi$ is not tight on $X$. Note

that, due to the definition of reduct of a program, every rule of $\Pi^X$ has the form

$$Head \leftarrow Body^X \tag{34}$$

for some rule (4) in $\Pi$. Then there is an infinite sequence $L_0, L_1, \ldots$ in $X$ such that, for every $i$, there is a rule (34) in $\Pi^X$ with $L_i = Head$, $L_{i+1} \in lit((Body^p)^X) \cap X$ and $X \models Body^X$.

By Lemma 3(i), $X \models Body^X$ iff $X \models Body$. From the definition of the reduct, $lit((Body^p)^X) \subseteq lit(Body^p)$. Therefore, there is an infinite sequence $L_0, L_1, \ldots$ in $X$ such that, for every $i$, there is a rule (4) in $\Pi$ with $L_i = Head$, $L_{i+1} \in lit(Body^p) \cap X$ and $X \models Body$. This contradicts the assumption that $\Pi$ is tight on $X$.

**Proof of Theorem 1 (Section 5).** Consider a program $\Pi$ and a consistent set $X$ of literals such that $\Pi$ is tight on $X$. Assume that $X$ is closed under and supported by $\Pi$. Then, by Lemma 3(ii,iii), $X$ is closed under and supported by $\Pi^X$. By Lemma 6, since $\Pi$ is tight on $X$, so is $\Pi^X$. Hence, by Lemma 5, $X$ is an answer set for $\Pi^X$, and consequently an answer set for $\Pi$. In the other direction, the assertion of the theorem follows from Proposition 1.

**Proof of Theorem 2 (Section 5).** Let $\Pi$ be a finite normal program, and let $X$ be a set of literals disjoint from $pos(\Pi)$. Condition (*) from the definition of tightness (Section 5) applied to $\Pi$ holds trivially for any choice of $\lambda$. Then, by Theorem 1, $X$ is an answer set for $\Pi$ iff $X$ is closed under and supported by $\Pi$.

**Proof of Proposition 4 (Section 7).** We will first prove the special case when $\Pi$ doesn't contain negation as failure. Let $X$ be an answer set for $\Pi \cup Def$; denote set (20) by $R$, and its transitive closure by $R^\infty$. We need to prove that for all $x$ and $y$, $tc(x, y) \in X$ iff $\langle x, y\rangle \in R^\infty$.

*Left-to right.* Since there is no negation as failure in $\Pi$, $X$ can be characterized as the union $\bigcup_i X_i$ of the sequence of sets of literals defined as follows: $X_0 = \emptyset$; $X_{i+1}$ is the set of all literals $L$ such that $\Pi \cup Def$ contains a rule $L \leftarrow Body$ with $Body \subseteq X_i$. We will show by induction on $i$ that $tc(x, y) \in X_i$ implies $\langle x, y\rangle \in R^\infty$. If $i = 0$, the assertion is trivial because $X_0$ is empty. Assume that for all $x$ and $y$, $tc(x, y) \in X_i$ implies $\langle x, y\rangle \in R^\infty$, and take an atom $tc(x, y)$ from $X_{i+1}$. Take a rule $tc(x, y) \leftarrow Body$ in $\Pi \cup Def$ such that $Body \subseteq X_i$. Since $\Pi$ doesn't contain atoms of the form $tc(x, y)$ in the heads of rules, this rule belongs to $Def$. *Case 1:* $Body = \{p(x, y)\}$. Then $p(x, y) \in X_i \subseteq X$, so that $\langle x, y\rangle \in R \subseteq R^\infty$. *Case 2:* $Body = \{p(x, v), tc(v, y)\}$. Then $p(x, v) \in X_i \subseteq X$, so that $\langle x, v\rangle \in R \subseteq R^\infty$;

also, $tc(v, y) \in X_i$, so that, by the induction hypothesis, $\langle v, y\rangle \in R^\infty$. By the transitivity of $R^\infty$, it follows that $\langle x, y\rangle \in R^\infty$.

*Right-to-left.* Since $R^\infty = \bigcup_{j>0} R^j$, it is sufficient to prove that for all $j > 0$, $\langle x, y\rangle \in R^j$ implies $tc(x, y) \in X$. The proof is by induction on $j$. When $j = 1$, $\langle x, y\rangle \in R$, so that $p(x, y) \in X$; since $X$ is closed under $Def$, it follows that $tc(x, y) \in X$. Assume that for all $x$ and $y$, $\langle x, y\rangle \in R^j$ implies $tc(x, y) \in X$, and take a pair $\langle x, y\rangle$ from $R^{j+1}$. Take $v$ such that $\langle x, v\rangle \in R$ and $\langle v, y\rangle \in R^j$. Then $p(x, v) \in X$ and, by the induction hypothesis, $tc(v, y) \in X$. Since $X$ is closed under $Def$, it follows that $tc(x, y) \in X$.

We have proved the assertion of Proposition 4 for programs without negation as failure. Now let $\Pi$ be any program that does not contain atoms of the form $tc(x, y)$ in heads of rules, and let $X$ be an answer set for $\Pi \cup Def$. Clearly, the reduct $\Pi^X$ is a program without negation as failure that does not contain atoms of the form $tc(x, y)$ in the heads of rules, and $X$ is an answer set for $\Pi^X \cup Def$. By the special case of the theorem proved above, applied to $\Pi^X$, (19) is the transitive closure of (20).

**Proof of Theorem 3 (Section 7).** Assume (i)–(iii), and assume that $\Pi \cup Def$ is not tight on $X$. Then there exists an infinite sequence $L_0, L_1, \ldots$ of elements of$X$ such that for every $i$, $L_{i+1}$ is a parent of $L_i$ relative to $\Pi \cup Def$ and $X$. Consider two cases.

*Case 1:* Sequence $L_0, L_1, \ldots$ contains only a finite number of terms of the form $tc(x, y)$. Let $L_n$ be the last of them. Then for every $i > n$, $L_{i+1}$ is a parent of $L_i$ relative to $\Pi$ and $X$. Sequence $L_{n+1}, L_{n+2}, \ldots$ shows that $\Pi$ is not tight on $X$, contrary to (i).

*Case 2:* Sequence $L_0, L_1, \ldots$ contains infinitely many terms of the form $tc(x, y)$. By (iii), it follows that this sequence has no terms of the form $p(x, y)$. The examination of rules $Def$ shows that every $tc(x, y)$ in this sequence is immediately followed by a term of the form $tc(v, y)$ such that $p(x, v) \in X$. Consequently, sequence $L_0, L_1, \ldots$ consists of some initial segment followed by an infinite sequence of literals of the form

$$tc(v_0, y), tc(v_1, y), \ldots$$

such that, for every $i$, $p(v_i, v_{i+1}) \in X$. This is impossible by (ii).

**Proof of Proposition 5 (Section 7).** Let $\Pi$ be a program containing constraint (22), with finite $C$, and let $X$ be a set of literals closed under $\Pi \cup Def$. Assume that $\{\langle x, y\rangle : \ p(y, x) \in X\}$ is not well-founded. Take

$x_1, \ldots, x_n \in C$ that satisfy (21). Since $X$ is closed under $Def$, $tc(x_1, x_1) \in X$. But this is impossible because $X$ is closed under (22).

**Proof of Proposition 6 (Section 8).** Let $\Pi$ be the program that differs from (23)–(31) in that

- its underlying set of atoms includes, additionally, expressions of the forms $on(table, l, t)$ and $above(table, l, t)$, and
- rules (29) and (30) are replaced by

$$\begin{array}{l} above(l, l', t) \leftarrow on(l, l', t), \\ above(l, l', t) \leftarrow on(l, l'', t), above(l'', l', t) \end{array} \tag{35}$$

and

$$\leftarrow above(l, l, t). \tag{36}$$

Let $X$ be a set of literals that does not contain any of the newly introduced atoms or their negations and is closed under the original program (23)–(31). We will prove that $\Pi$ is tight on $X$. It will follow then that the original program is tight on $X$ as well, because that program is a subset of $\Pi$.

For every $k = 0, \ldots, T+1$, let $\Pi_k$ be the subset of the rules of $\Pi$ in which rules (35) are restricted to $t < k$. Since $\Pi_{T+1} = \Pi$, it is sufficient to prove that, for all $k$, $\Pi_k$ is tight on $X$. The proof is by induction on $k$. *Basis:* $k = 0$. The rules of $\Pi_0$ are (23)–(28), (31) and (36). To see that this program is tight, define

$$\begin{array}{l} \lambda(on(l, l', t)) = t + 1, \\ \lambda(\neg on(l, l', t)) = t + 2, \\ \lambda(move(b, l, t)) = \lambda(\neg move(b, l, t)) = 0, \\ \lambda(above(l, l', t)) = \lambda(\neg above(l, l', t)) = 0. \end{array}$$

*Induction step:* Assume that $\Pi_k$ is tight on $X$. Let $C$ be the set of location constants, and let functions $p$ and $tc$ be defined by

$$\begin{array}{l} p(l, l') = on(l, l', k+1), \\ tc(l, l') = above(l, l', k+1). \end{array}$$

Then $\Pi_{k+1} = \Pi_k \cup Def$. Let us check that all conditions of Theorem 3 are satisfied. Condition (i) holds by the induction hypothesis. Since $X$ is closed under the original program (23)–(31) and does not contain any of the newly introduced literals, it is closed under $\Pi_{k+1}$ as well; in view of the fact that $\Pi_{k+1}$ contains constraint (36), condition (ii) follows by Proposition 5. By inspection, (iii) holds also. By Theorem 3, it follows that $\Pi_{k+1}$ is tight on $X$.

## 10 Conclusion

For tight logic programs, the answer set semantics is equivalent to the completion semantics. Answer sets for a tight program can be found by running a satisfiability solver on the program's completion. This method of computing answer sets is applicable to rules with nested expressions, including choice rules in the sense of SMODELS.

This approach to answer set programming is not directly applicable to disjunctive programs, but in some cases the disjunction in the head of a rule can be eliminated in favor of nested expressions in its body [Lifschitz *et al.*, 1999], [Lifschitz *et al.*, 2001]. For instance, the disjunctive rule

$$p; not\ q \leftarrow r$$

in any program can be replaced by

$$p \leftarrow r, not\ not\ q$$

without changing the program's answer sets.

## Acknowledgments

We are grateful to Yuliya Babovich, Victor Marek, Norman McCain, Emilio Remolina and Hudson Turner for useful discussions related to the subject of this paper. This work was partially supported by National Science Foundation under grant IIS-9732744. The first author was also supported by a NATO Science Fellowship and by a grant from the Texas Higher Education Coordinating Board.

## References

[Apt *et al.*, 1988] Krzysztof Apt, Howard Blair, and Adrian Walker. Towards a theory of declarative knowledge. In Jack Minker, editor, *Foundations of Deductive Databases and Logic Programming*, pages 89–148. Morgan Kaufmann, San Mateo, CA, 1988.

[Babovich *et al.*, 2000] Yuliya Babovich, Esra Erdem, and Vladimir Lifschitz. Fages' theorem and answer set programming.[16] In *Proc. Eighth Int'l Workshop on Non-Monotonic Reasoning*, 2000.

[16] `http://arxiv.org/abs/cs.ai/0003042` .

[Baral and Gelfond, 1994] Chitta Baral and Michael Gelfond. Logic programming and knowledge representation. *Journal of Logic Programming*, 19,20:73–148, 1994.

[Clark, 1978] Keith Clark. Negation as failure. In Herve Gallaire and Jack Minker, editors, *Logic and Data Bases*, pages 293–322. Plenum Press, New York, 1978.

[Erdem and Lifschitz, 1999] Esra Erdem and Vladimir Lifschitz. Transformations of logic programs related to causality and planning. In *Logic Programming and Non-monotonic Reasoning: Proc. Fifth Int'l Conf. (Lecture Notes in Artificial Intelligence 1730)*, pages 107–116, 1999.

[Erdem and Lifschitz, 2001] Esra Erdem and Vladimir Lifschitz. Fages' theorem for programs with nested expressions. In *Proc. 2001 Int'l Conf. on Logic Programming*, 2001. To appear.

[Fages, 1994] François Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of Logic in Computer Science*, 1:51–60, 1994.

[Ferraris and Lifschitz, 2001] Paolo Ferraris and Vladimir Lifschitz. Weight constraints as nested expressions. Submitted for publication, 2001.

[Gelfond and Lifschitz, 1988] Michael Gelfond and Vladimir Lifschitz. The stable model semantics for logic programming. In Robert Kowalski and Kenneth Bowen, editors, *Logic Programming: Proc. Fifth Int'l Conf. and Symp.*, pages 1070–1080, 1988.

[Gelfond and Lifschitz, 1991] Michael Gelfond and Vladimir Lifschitz. Classical negation in logic programs and disjunctive databases. *New Generation Computing*, 9:365–385, 1991.

[Giunchiglia *et al.*, 2001] Enrico Giunchiglia, Joohyung Lee, Vladimir Lifschitz, and Hudson Turner. Causal laws and multi-valued fluents.[17] Unpublished draft, 2001.

[Inoue and Sakama, 1998] Katsumi Inoue and Chiaki Sakama. Negation as failure in the head. *Journal of Logic Programming*, 35:39–78, 1998.

[Lifschitz and Turner, 1994] Vladimir Lifschitz and Hudson Turner. Splitting a logic program. In Pascal Van Hentenryck, editor, *Proc. Eleventh Int'l Conf. on Logic Programming*, pages 23–37, 1994.

[17] `http://www.cs.utexas.edu/users/vl/papers/clmvf-long.ps` .

[Lifschitz *et al.*, 1999] Vladimir Lifschitz, Lappoon R. Tang, and Hudson Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence*, 25:369–389, 1999.

[Lifschitz *et al.*, 2001] Vladimir Lifschitz, David Pearce, and Agustin Valverde. Strongly equivalent logic programs. *ACM Transactions on Computational Logic*, 2:526–541, 2001.

[Lifschitz, 1996] Vladimir Lifschitz. Foundations of logic programming. In Gerhard Brewka, editor, *Principles of Knowledge Representation*, pages 69–128. CSLI Publications, 1996.

[Lifschitz, 1999] Vladimir Lifschitz. Answer set planning. In *Proc. ICLP-99*, pages 23–37, 1999.

[Lloyd and Topor, 1984] John Lloyd and Rodney Topor. Making Prolog more expressive. *Journal of Logic Programming*, 3:225–240, 1984.

[Niemelä *et al.*, 1999] Ilkka Niemelä, Patrik Simons, and Timo Soininen. Stable model semantics for weight constraint rules. In *Logic Programming and Non-monotonic Reasoning: Proc. Fifth Int'l Conf. (Lecture Notes in Artificial Intelligence 1730)*, pages 317–331, 1999.