Tight Logic Programs

Esra Erdem
Department of Computer Science
University of Toronto
6 King's College Road
Toronto, ON M5S 3H5, Canada
(*e-mail:* esra@cs.toronto.edu)

Vladimir Lifschitz
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78712, USA
(*e-mail:* 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 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.

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

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 2001a).

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

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

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 (Erdem and Lifschitz 2001b).

The concept of a tight program and generalizations of Fages' theorem may be interesting for two 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 conditions 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 satisfiability solver such as SATO[3] or MCHAFF[4] (Babovich et al. 2000). The idea of computing answer sets for a program using a satisfiability solver led to the creation of the system CMODELS[5] which, in some cases, finds answer sets faster than "general-purpose" answer set solvers such as SMODELS[6] and DLV[7]. Our generalization of Fages' theorem allows us to apply this idea to programs containing weight constraint rules (Simons et al. 2002), because such programs

[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.

[3] `http://www.cs.uiowa.edu/~hzhang/sato.html` .

[4] `http://www.ee.princeton.edu/~mchaff/` .

[5] `http://www.cs.utexas.edu/users/tag/cmodels.html` .

[6] `http://www.tcs.hut.fi/Software/smodels/` .

[7] `http://www.dbai.tuwien.ac.at/proj/dlv/` .

can be viewed as a special case of programs with nested expressions (Ferraris and Lifschitz 2003). For instance, rule $\{p\}$ can be treated as shorthand for

$$p \leftarrow not\ not\ p.$$

CMODELS uses this fact to handle programs with weight constraints, and our generalization of Fages' theorem allows CMODELS to decide whether the answer sets for such a program are identical to the models of its completion.

We begin by reviewing the definitions of answer sets, closure, supportedness and completion for programs with nested expressions (Sections 2, 3). After discussing the concept of tightness for such programs in Sections 4 and 5, we state our generalization of Fages' theorem (Section 6). As an example, we show how this generalized form applies to a formalization of the n-queens problem (Section 7). Then we study the tightness of programs containing the definition of the transitive closure of a relation (Section 8). A logic programming description of the blocks world is used as an example in Section 9. Proofs of theorems are given in Section 10.

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

$$Head \leftarrow Body \tag{3}$$

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 Π if, for every rule (3) in Π, $Head \in X$ whenever $X \models Body$.

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

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

$$\begin{array}{l} p \\ p \leftarrow p, q \end{array} \tag{4}$$

is $\{p\}$.

The *reduct* Π^X of a program Π relative to a set X of literals is obtained from Π by replacing every maximal occurrence of a formula of the form *not* F in Π (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 Π if it is the answer set for the reduct Π^X. For instance, $\{p\}$ is an answer set for program (1) since it is the answer set for the reduct (4) 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 every 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.

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 Π if for every literal $L \in X$ there exists a rule (3) in Π such that $\mathit{Head} = L$ and $X \models \mathit{Body}$.[10] For instance, each of the sets $\{p\}$, $\{\neg q\}$ is supported by the program

$$\begin{array}{l} p \leftarrow \mathit{not}\ \neg q \\ \neg q \leftarrow \mathit{not}\ p \end{array} \tag{5}$$

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 Π and any consistent set X of literals, if X is an answer set for Π then X is closed under and supported by Π.

For instance, the answer sets for program (5) are $\{p\}$ and $\{\neg q\}$; each of them is

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

[10] In (Baral and Gelfond 1994) and (Inoue and Sakama 1998) the definition of supportedness from (Apt et al. 1988) is generalized in a different direction—it is extended to disjunctive programs.

closed under and supported by (5). 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 Π be a finite normal program. The "completion" of Π 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

$$A \equiv Body_1 \vee \cdots \vee Body_k \tag{6}$$

where the disjunction extends over all rules

$$A \leftarrow Body_i \tag{7}$$

in Π with the head A. The *completion* of Π 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 Π, 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 Π, a set of atoms satisfies the completion of Π iff it is closed under and supported by Π.

From Propositions 1 and 2, we conclude:

Corollary 1
For any finite normal program Π and any set X of atoms, if X is an answer set for Π then X satisfies the completion of Π.

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, $lit(p; not\ \neg r) = \{p, \neg r\}$. For any formula G, by $poslit(G)$ we denote the set of all literals having a regular occurrence in G that is not in the scope of negation as failure. For instance, $poslit(p, not\ q, (not\ not\ r; \neg s)) = \{p, \neg s\}$.

[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 (6).

For any program Π and any set X of literals, we say about literals $L, L' \in X$ that L is a *parent* of L' relative to Π and X if there is a rule (3) in Π such that

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

For instance, the parents of p relative to the program

$$\begin{array}{l} p \leftarrow not\ q \\ q \leftarrow not\ p \\ p \leftarrow p, r \end{array} \tag{8}$$

and the set $\{p, q, r\}$ are p and r; on the other hand, p has no parents relative to (8) and the set $\{p, q\}$.

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

A program Π is *tight* on a set X of literals if there is no infinite sequence $L_1, L_2, \dots$ of elements of X such that for every i, L_{i+1} is a parent of L_i relative to Π and X.

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

If X is finite then the tightness condition can be reformulated as follows: there is no finite sequence $L_1, \dots, L_n$ of elements of X ($n > 1$) such that for every i ($1 \leq i < n$), L_{i+1} is a parent of L_i relative to Π and X, and $L_n = L_1$.

For instance, program (1) is tight on $\{p\}$: p does not have parents relative to (1) and $\{p\}$, so that 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 $p, p, \dots$ every element is followed by its parent.

The proposition below gives an equivalent characterization of tightness:

Proposition 3

A program Π is tight on a set X of literals iff there exists a function λ from X to ordinals satisfying the following condition:

(*) for every rule (3) in Π such that $Head \in X$ and $X \models Body$, and for every L in $X \cap poslit(Body)$, $\lambda(L) < \lambda(Head)$.

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

$$p \leftarrow q; not\ r \tag{9}$$

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

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

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

[12] A binary relation R is *well-founded* if there is no infinite sequence $x_1, x_2, \dots$ of elements of its domain such that, for all i, $x_{i+1} R x_i$.

Lemma from Set Theory
A binary relation R is well-founded iff there exists a function λ 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 Π have the form

$$Head \leftarrow L_1, \ldots, L_m, not\ L_{m+1}, \ldots, not\ L_n \tag{10}$$

where each L_i is a literal. In this case, condition (*) says: *for every rule (10) in* Π*, if*

$$Head \in X, \tag{11}$$

$$L_1, \ldots, L_m \in X \tag{12}$$

and

$$L_{m+1}, \ldots, L_n \notin X \tag{13}$$

then, for all $L \in X \cap \{L_1, \ldots, L_m\}$, $\lambda(L) < \lambda(Head)$. In view of (12), 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 (13). The additional generality gained by including (13) can be illustrated by the program

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

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

5 Absolutely Tight Programs

The following modification of the tightness condition is often useful. A program Π is *absolutely tight* if there is no infinite sequence $L_1, L_2, \ldots$ of literals such that for every i there is a rule (3) in Π for which $L_{i+1} \in poslit(Body)$ and $L_i = Head$. It is clear that an absolutely tight program is tight on any set of literals.

To prove that a program Π is absolutely tight, it is sufficient to find a function λ from literals to ordinals such that for every rule (3) in Π with $Head \neq \bot$ and for every literal $L \in poslit(Body)$, $\lambda(L) < \lambda(Head)$.

For a program containing finitely many atoms, absolute tightness can be characterized as follows. The *positive dependency graph* of a program Π is the directed graph G such that

- the vertices of G are the literals that have regular occurrences in Π, and
- G has an edge from L to L' if there is a rule (3) in Π for which $L \in poslit(Body)$ and $L' = Head$.

A program containing finitely many atoms is absolutely tight iff its positive dependency graph has no cycles.

In application to programs whose rules have the form (10) and contain neither $\bot$ nor classical negation, the definition of absolute tightness above turns into (the propositional case of) Fages' original definition of tightness (Fages 1994).

6 Generalization of Fages' Theorem

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

For instance, program (5) is tight on the sets $\{p\}$ and $\{\neg q\}$ that are closed under and supported by (5). By Proposition 1 and the theorem above, it follows that $\{p\}$ and $\{\neg q\}$ are the answer sets for (5).

By Proposition 2, we conclude:

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

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

By $pos(\Pi)$ we denote the set of literals L such that Π contains a rule (3) with $Head \neq \bot$ and $L \in poslit(Body)$. For instance, if Π is (1) then $pos(\Pi) = \{p, q\}$. If a set X of literals is disjoint from $pos(\Pi)$ then no literal in X has a parent relative to Π and X, and consequently Π is tight on X. We conclude:

Corollary 3
For any program Π and any consistent set X of literals disjoint from $pos(\Pi)$, X is an answer set for Π iff X is closed under and supported by Π.

By Proposition 2, it follows then:

Corollary 4
For any finite normal program Π and any set X of atoms disjoint from $pos(\Pi)$, X is an answer set for Π iff X satisfies the completion of Π.

Since an absolutely tight program is tight on every set of literals, we conclude from Theorem 1 and Corollary 2:

Corollary 5
For any absolutely tight program Π and any consistent set X of literals, X is an answer set for Π iff X is closed under and supported by Π.

Corollary 6
For any finite normal absolutely tight program Π and any set X of atoms, X is an answer set for Π iff X satisfies the completion of Π.

```
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).
```

Fig. 1. An SMODELS representation of the n-queens problem.

7 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, (a) no two queens may be on the same column, (b) no two queens may be on the same row, and (c) no two queens may be on the same diagonal.

A solution to the n-queens problem can be described by a set of atoms of the form $queen(R, C)$ $(1 \leq R, C \leq n)$ satisfying conditions (a)–(c); including $queen(R, C)$ in the set indicates that there is a queen in position (R, C).

The n-queens problem can be described by a program whose answer sets are solutions, as follows. The selections satisfying condition (a) correspond to the answer sets for the program consisting of the rules

$$queen(R, C) \leftarrow not\ not\ queen(R, C) \tag{14}$$

for all R, C in $\{1, \ldots, n\}$,

$$\leftarrow not\ queen(1, C), \ldots, not\ queen(n, C) \tag{15}$$

for all C in $\{1, \ldots, n\}$, and

$$\leftarrow queen(R, C), queen(R1, C) \tag{16}$$

for all $R, R1, C$ in $\{1, \ldots, n\}$ such that $R < R1$. Conditions (b) and (c) are represented by the constraints

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

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

$$\leftarrow queen(R, C), queen(R1, C1) \tag{18}$$

for all $R, R1, C, C1$ in $\{1, \ldots, n\}$ such that $C < C1$ and $|R - R1| = |C - C1|$.

The answer sets for program (14)–(18) are in a 1–1 correspondence with the possible arrangements of n queens. According to (Ferraris and Lifschitz 2003), rules (14)–(16) can be rewritten as weight constraints

$$1\{queen(1, C), \ldots, queen(n, C)\}1 \tag{19}$$

for all C in $\{1, \ldots, n\}$ Then the program can be presented to SMODELS as shown in Figure 1.

Program (14)–(18) is a finite normal absolutely tight program. By Corollary 6, its answer sets are identical to the models of its completion. (Note the use of nested negations in rule (14); this is the reason why the new generalization of Fages' theorem is needed here.) This fact can be used to find solutions to the n-queens problem using CMODELS, that is to say, by running a satisfiability solver on the program's completion. CMODELS transforms the input shown in Figure 1 into (14)–(18), computes the completion, clausifies it, and calls MCHAFF to find a model. For $n = 20$ CMODELS finds a solution in 2 seconds (for comparison, SMODELS, given the same input file, finds one in 55 seconds).[13] For $n = 25$, CMODELS finds a solution in 3 seconds, whereas SMODELS requires more than 2 hours.

This example confirms the conjecture underlying the design of CMODELS: 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.

8 Transitive Closure

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

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

If we combine this definition Def with any set Π 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. The absence of such "spurious" sets can be assured by requiring that facts in Π define relation p to be acyclic.

In this section we study the more general situation when Π 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 Π, 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 Π 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,

[13] We have used LPARSE 1.0.11, SMODELS 2.27, CMODELS 1.03, and MCHAFF SPELT3. All CPU times here are for a SunBlade 1000, with two 600MHz UltraSPARC-III processors and 5GB RAM.

however, to be a little more general. We assume p and tc 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 Π 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

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

is the transitive closure of

$$\{\langle x, y\rangle : p(x, y) \in X\}. \tag{21}$$

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

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

Theorem 2
Let Π 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) Π 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 Π and X,

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

By Theorem 1 and Proposition 4, we conclude:

Corollary 7
Let Π 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 2. If, in addition,

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

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

then X is an answer set for $\Pi \cup \mathit{Def}$, 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 Π is a finite program without classical negation then condition (iv) can be reformulated as follows:

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

Condition (ii) in the statement of Theorem 2 is similar to the acyclicity property mentioned at the beginning of this section. 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 $x_1, \dots, x_n \in C$ $(n > 1)$ such that

$$p(x_1, x_2), \dots, p(x_{n-1}, x_n) \in X \tag{22}$$

and $x_n = x_1$. 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 Π contains constraint

$$\leftarrow tc(x, x) \tag{23}$$

and C is finite then, for every set X of literals closed under $\Pi \cup \mathit{Def}$, 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 Π 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) is essential as well. Indeed, take Π 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 $\Pi \cup \mathit{Def}$, but is is not an answer set for $\Pi \cup \mathit{Def}$: the only answer set for this program is empty.

9 Example: The Blocks World

As an example of the use of Theorem 2, 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 $0, \dots, T$,

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 (32) that requires every block to be "supported by the table" and thus eliminates stacks of blocks flying in space.

The program consists of the following rules:[15]

$$\begin{array}{l} on(b,l,0) \leftarrow not\ \neg on(b,l,0) \\ \neg on(b,l,0) \leftarrow not\ on(b,l,0) \\ move(b,l,t) \leftarrow not\ \neg move(b,l,t) \\ \neg move(b,l,t) \leftarrow not\ move(b,l,t) \end{array} \tag{24}$$

$$\begin{array}{l} on(b,l,t+1) \leftarrow move(b,l,t) \\ on(b,l,t+1) \leftarrow on(b,l,t), not\ \neg on(b,l,t+1) \end{array} \tag{25}$$

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

$$\leftarrow on(b,b'',t), on(b',b'',t) \qquad (b \neq b') \tag{27}$$

$$\leftarrow move(b,l,t), on(b',b,t) \tag{28}$$

$$\leftarrow move(b,l,t), move(b',l',t) \qquad (b \neq b' \text{ or } l \neq l') \tag{29}$$

$$\begin{array}{l} above(b,l,t) \leftarrow on(b,l,t) \\ above(b,l,t) \leftarrow on(b,b',t), above(b',l,t) \end{array} \tag{30}$$

$$\leftarrow above(b,b,t) \tag{31}$$

$$\leftarrow not\ above(b, table, t) \tag{32}$$

To illustrate the use of Theorem 2, in Section 10 we use it to prove the following proposition:

Proposition 6
Program (24)–(32) is tight on every set of literals that is closed under it.

This proposition, in combination with Proposition 1 and Theorem 1, tells us that the answer sets for (24)–(32) can be characterized as the sets that are closed under this program and supported by it. These answer sets can be computed by eliminating classical negation in favor of new atoms and generating models of the completion of the resulting program.

[15] This program is similar to the history program for the blocks world from (Lifschitz 1999). Instead of rules (24), 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 (25)–(28) is discussed in (Lifschitz 1999), Section 6. Rule (29) prohibits concurrent actions. Rules (30) and (31) 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 (24)–(29), (31), (32) is tight, and then use Theorem 2 to conclude that tightness is preserved when we add the definition (30) 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 2 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 2: 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 (24)–(32) in which, syntactically, *table* is allowed as the first argument of both *on* and *above*.

10 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 Π is assumed to be a program without negation as failure.

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

Proof
Let Π be a program without negation as failure and X be an answer set for Π. By the definition of an answer set for programs without negation as failure, X is closed under Π. To prove supportedness, take any literal L in X. Since X is minimal among the sets closed under Π, $X \setminus \{L\}$ is not closed under Π. This means that Π contains a rule (3) 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 Π, 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 Π, and any consistent set X of literals,

(i) $X \models F$ iff $X \models F^X$;
(ii) X is closed under Π iff X is closed under Π^X;
(iii) X is supported by Π iff X is supported by Π^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 Π and an answer set X for Π. By the definition of an answer set, X is an answer set for Π^X. Then, by Lemma 2, X is closed under and supported by Π^X. By Lemma 3(ii,iii), it follows that X is closed under and supported by Π.

Proof of Proposition 2 (Section 3)

Let Π be a finite normal program. Recall that the completion of Π consists of the equivalences (6) where A is an atom or the symbol $\bot$. It is clear that a set X of atoms satisfies the completion of Π iff, for each A,

(a) for every rule (7) in Π 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 (7) in Π with the head A.

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

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:

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

For any $x \in \bigcup_\alpha S_\alpha$, define $\lambda(x)$ to be the smallest α 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, $X \models F$ 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 Π is assumed to be a program without negation as failure.

Lemma 5

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

Proof
Let Π be a program without negation as failure and X be a consistent set of literals such that Π is tight on X. Suppose that X is closed under and supported by Π. 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 Π. 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 Π and X. Indeed, assume that there is no such literal, so that every literal in $X \setminus Y$ has a parent relative to Π and X in this set; then there exists an infinite sequence $L_1, L_2, \dots$ of elements of $X \setminus Y$ such that L_{i+1} is a parent of L_i, which contradicts the assumption that Π is tight on X.

Take such a literal L. Since X is supported by Π, there is a rule

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

in Π such that

$$X \models \mathit{Body}. \tag{33}$$

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

$$X \cap \mathit{lit}(\mathit{Body}) \subseteq Y. \tag{34}$$

By Lemma 4, we can conclude from (33) that

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

In view of (34), it follows by Lemma 1 that $Y \models \mathit{Body}$. Since $L \notin Y$, it follows that Y is not closed under Π.

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

Proof
Let Π be a program and X be a consistent set of literals. Suppose that Π^X is not tight on X; we want to show that Π is not tight on X. It is clear from the definition of reduct of a program that every rule of Π^X has the form

$$\mathit{Head} \leftarrow \mathit{Body}^X \tag{35}$$

for some rule (3) in Π. Then there is an infinite sequence $L_1, L_2, \dots$ in X such that, for every i, there is a rule (35) in Π^X with $L_i = \mathit{Head}$, $L_{i+1} \in \mathit{poslit}(\mathit{Body}^X) \cap X$ and $X \models \mathit{Body}^X$.

By Lemma 3(i), $X \models \mathit{Body}^X$ iff $X \models \mathit{Body}$. From the definition of the reduct, $\mathit{poslit}(\mathit{Body}^X) \subseteq \mathit{poslit}(\mathit{Body})$. Therefore, there is an infinite sequence $L_1, L_2, \dots$ in X such that, for every i, there is a rule (3) in Π with $L_i = \mathit{Head}$, $L_{i+1} \in \mathit{poslit}(\mathit{Body}) \cap X$ and $X \models \mathit{Body}$. This contradicts the assumption that Π is tight on X.

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

Proof of Proposition 4 (Section 8)
We will first prove the special case when Π doesn't contain negation as failure. Let X be an answer set for $\Pi \cup \mathit{Def}$; denote set (21) by R, and its transitive closure by R^∞. 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 Π, 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 $\Pi \cup \mathit{Def}$ contains a rule $L \leftarrow \mathit{Body}$ with Body satisfied by 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 $tc(x,y) \leftarrow \mathit{Body}$ in $\Pi \cup \mathit{Def}$ such that $X_i \models \mathit{Body}$. Since Π doesn't contain atoms of the form $tc(x,y)$ in the heads of rules, this rule belongs to Def. *Case 1:* $\mathit{Body} = p(x,y)$. Then $p(x,y) \in X_i \subseteq X$, so that $\langle x,y\rangle \in R \subseteq R^\infty$. *Case 2:* $\mathit{Body} = p(x,v), tc(v,y)$. 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^∞, 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 Π 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 $\Pi \cup \mathit{Def}$. Clearly, the reduct Π^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 $\Pi^X \cup \mathit{Def}$. By the special case of the theorem proved above, applied to Π^X, (20) is the transitive closure of (21).

Proof of Theorem 2 (Section 8)
Assume (i)–(iii), and assume that $\Pi \cup \mathit{Def}$ is not tight on X. Then there exists an infinite sequence $L_1, L_2, \ldots$ of elements of X such that for every i, L_{i+1} is a parent of L_i relative to $\Pi \cup \mathit{Def}$ and X. Consider two cases.

Case 1: Sequence $L_1, L_2, \ldots$ 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 Π and X. Sequence $L_{n+1}, L_{n+2}, \dots$ shows that Π is not tight on X, contrary to (i).

Case 2: Sequence $L_1, L_2, \dots$ 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 $L_1, L_2, \dots$ consists of some initial segment followed by an infinite sequence of literals of the form

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

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

Proof of Proposition 5 (Section 8)

Let Π be a program containing constraint (23), with finite C, and let X be a set of literals closed under $\Pi \cup \mathit{Def}$. Assume that $\{\langle x,y\rangle : \ p(y,x) \in X\}$ is not well-founded. Take $x_1, \dots, x_n \in C$ that satisfy (22) and $x_n = x_1$. Since X is closed under *Def*, $tc(x_1,x_1) \in X$. But this is impossible because X is closed under (23).

Proof of Proposition 6 (Section 9)

Let Π be the program that differs from (24)–(32) in that

- its underlying set of atoms includes, additionally, expressions of the forms $on(table,l,t)$ and $above(table,l,t)$, and
- rules (30) and (31) are replaced by

$$\begin{array}{l} above(l,l',t) \leftarrow on(l,l',t) \\ above(l,l',t) \leftarrow on(l,l'',t), above(l'',l',t) \end{array} \tag{36}$$

  and

$$\leftarrow above(l,l,t). \tag{37}$$

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 (24)–(32). We will prove that Π 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 Π.

For every $k = 0, \dots, T+1$, let Π_k be the subset of the rules of Π in which rules (36) are restricted to $t < k$. Since $\Pi_{T+1} = \Pi$, it is sufficient to prove that, for all k, Π_k is tight on X. The proof is by induction on k. *Basis:* $k = 0$. The rules of Π_0 are (24)–(29), (32) and (37). 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 Π_k is tight on X. Let C be the set of location constants, and let functions p and tc be defined by

$$p(l, l') = on(l, l', k),$$
$$tc(l, l') = above(l, l', k).$$

Then $\Pi_{k+1} = \Pi_k \cup Def$. Let us check that all conditions of Theorem 2 are satisfied. Condition (i) holds by the induction hypothesis. Since X is closed under the original program (24)–(32) and does not contain any of the newly introduced literals, it is closed under Π_{k+1} as well; in view of the fact that Π_{k+1} contains constraint (37), condition (ii) follows by Proposition 5. By inspection, (iii) holds also. By Theorem 2, it follows that Π_{k+1} is tight on X.

11 Conclusion

For absolutely tight logic programs, the answer set semantics is equivalent to the completion semantics. Answer sets for a finite normal absolutely tight program can be found by running a satisfiability solver on the program's completion. Defining tightness relative to a set of literals extends the applicability of this method to some programs that are not absolutely tight. This method of computing answer sets is applicable to rules with nested expressions and with weight constraints.

Although this method is not directly applicable to disjunctive programs, disjunction in the head of a rule can be sometimes eliminated in favor of nested expressions in its body (Lifschitz et al. 1999), (Lifschitz 2002). 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, Selim T. Erdoğan, Paolo Ferraris, Joohyung Lee, Victor Marek, Norman McCain and Emilio Remolina for comments and discussions related to the subject of this paper. Special thanks to Hudson Turner for many useful suggestions, including the use of *poslit* in the definition of tightness instead of the more complicated condition from (Erdem and Lifschitz 2001a), and the possibility of restriction (13) from Section 4.

This work was partially supported by National Science Foundation under grant IIS-9732744 and by the Texas Higher Education Coordinating Board under grant 003658-0322-2001. The first author was also supported by a NATO Science Fellowship.

References

Apt, K., Blair, H., and Walker, A. 1988. Towards a theory of declarative knowledge.

In *Foundations of Deductive Databases and Logic Programming*, J. Minker, Ed. Morgan Kaufmann, San Mateo, CA, 89–148.

Babovich, Y., Erdem, E., and Lifschitz, V. 2000. Fages' theorem and answer set programming.[16] In *Proc. NMR-2000*.

Baral, C. and Gelfond, M. 1994. Logic programming and knowledge representation. *Journal of Logic Programming 19,20*, 73–148.

Clark, K. 1978. Negation as failure. In *Logic and Data Bases*, H. Gallaire and J. Minker, Eds. Plenum Press, New York, 293–322.

Erdem, E. and Lifschitz, V. 1999. 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)*. 107–116.

Erdem, E. and Lifschitz, V. 2001a. Fages' theorem for programs with nested expressions. In *Proceedings of the Seventeenth International Conference on Logic Programming*. 242–254.

Erdem, E. and Lifschitz, V. 2001b. Transitive closure, answer sets, and predicate completion. In *Working Notes of the AAAI Spring Symposium on Answer Set Programming*.

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

Ferraris, P. and Lifschitz, V. 2003. Weight constraints as nested expressions. In progress.

Gelfond, M. and Lifschitz, V. 1988. The stable model semantics for logic programming. In *Logic Programming: Proc. Fifth Int'l Conf. and Symp.*, R. Kowalski and K. Bowen, Eds. 1070–1080.

Gelfond, M. and Lifschitz, V. 1991. Classical negation in logic programs and disjunctive databases. *New Generation Computing 9*, 365–385.

Inoue, K. and Sakama, C. 1998. Negation as failure in the head. *Journal of Logic Programming 35*, 39–78.

Lifschitz, V. 1996. Two components of an action language. In *Working Papers of the Third Symposium on Logical Formalizations of Commonsense Reasoning*.

Lifschitz, V. 1999. Answer set planning. In *Proc. ICLP-99*. 23–37.

Lifschitz, V. 2002. Answer set programming and plan generation. *Artificial Intelligence 138*, 39–54.

Lifschitz, V., Tang, L. R., and Turner, H. 1999. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence 25*, 369–389.

Lloyd, J. and Topor, R. 1984. Making Prolog more expressive. *Journal of Logic Programming 3*, 225–240.

Simons, P., Niemelä, I., and Soininen, T. 2002. Extending and implementing the stable model semantics. *Artificial Intelligence 138*, 181–234.

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