Transitive Closure, Answer Sets and Predicate Completion

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

We prove that the usual logic programming definition of transitive closure is correct under the answer set semantics, and investigate under what conditions it is correct under the completion semantics. That definition is allowed here to be combined with an arbitrary set of rules that may contain negation as failure, not merely with a set of facts. This work is motivated by applications to answer set programming.

Introduction

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 Π of facts (that is, ground atoms) 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 completion of $\Pi \cup Def$ in the sense of Clark (1978) may have models different from the minimal model. In these "spurious" models of completion, tc is weaker than the transitive closure of p. The existence of such models is often considered a blemish on the completion semantics. The absence of "spurious" models can be assured, however, by requiring that facts Π define relation p to be acyclic.

In this paper we consider 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 (Gelfond & Lifschitz 1990) instead of the minimal model. Program $\Pi \cup Def$ may have many answer sets. Is it true that, in each of them, the extent of tc is the transitive closure of the extent of p? Theorem 1 gives a positive answer to this question. Next, we would like to know under what conditions the completion of a program containing *Def* has no "spurious" models. Such conditions are provided in Theorem 2.

The questions discussed in this paper are important from the perspective of answer set programming. The idea of this programming method is to represent solutions to a computational problem by answer sets of a logic program, and to solve the problem using systems for generating answer sets, such as DLV[1] and SMODELS[2]. A logic program used in this process may define the transitive closure of one of its predicates—for instance, the transitive closure *above* of the relation *on* in the blocks world. In such cases, we expect that the definition will "operate correctly" in the context of a set of rules with negation as failure, not merely in combination with a set of facts. If every model of the completion of such a program is an answer set then the program's answer sets can be found using a propositional solver, as proposed in (Babovich, Erdem, & Lifschitz 2000).

In the next section, we prove the correctness of the definition of transitive closure under the answer set semantics. After a review of the concept of a tight programs, the correctness of the definition of transitive closure relative to the completion semantics is investigated. After that, this theory is illustrated by applying it to a formalization of the blocks world.

Transitive Closure and Answer Sets

The syntax of the class of logic programs studied in this paper is defined as follows. We begin with a set of propositional symbols, called *atoms*. A *literal* is an expression of the form A or $\neg A$, where A is an atom. A *rule element* is an expression of the form L or *not* L, where L is a literal. The symbol $\neg$ is called *classical negation*, and the symbol *not* is *negation as failure*. A *rule* is a pair *Head* $\leftarrow$ *Body* where *Head* is a literal or the symbol $\bot$, and *Body* is a finite set of rule elements. Thus a rule has the form

$$Head \leftarrow L_1, \dots, L_m, not\ L_{m+1}, \dots, not\ L_n \qquad (1)$$

where $n \geq m \geq 0$; we drop { } around the elements of the body. A rule (1) is a *constraint* if $Head = \bot$. A

[1] `http://www.dbai.tuwien.ac.at/proj/dlv/` .
[2] `http://www.tcs.hut.fi/Software/smodels/` .

program is a set of rules.

For the definition of an answer set for programs of this kind see (Lifschitz & Turner 1999), Section 3.

The syntax of logic programs defined above is propositional. 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.

Theorem 1 *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*

$$\{\langle x,y\rangle : tc(x,y) \in X\} \tag{2}$$

is the transitive closure of

$$\{\langle x,y\rangle : p(x,y) \in X\}. \tag{3}$$

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.[3] 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 $tc(x,y)$. This is easy to prove using the splitting set theorem (Lifschitz & Turner 1994).

Proof of Theorem 1. We will first prove the special case of the theorem when Π doesn't contain negation as failure. Let X be an answer set for $\Pi \cup Def$; denote set (3) 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 $\Pi \cup Def$ 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 $X_0 = \emptyset$. 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 Π 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^∞, 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 the theorem 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 $\Pi \cup Def$. 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 $\Pi^X \cup Def$. By the special case of the theorem proved above, applied to Π^X, (2) is the transitive closure of (3).

Tight Programs

The second question we want to investigate is under what conditions the completion of a program of the form $\Pi \cup Def$ has no "spurious" models, that is to say, no models different from the program's answer sets. Early work on the relationship between completion and answer sets was done by Fages (1994). We will review here the generalization of his theorem given in (Babovich, Erdem, & Lifschitz 2000).

A program Π is said to be *tight* on a set X of literals if there exists a function λ from X to ordinals such that for every rule (1) in Π, if $Head, L_1, \dots, L_m \in X$ then

$$\lambda(L_1), \dots, \lambda(L_m) < \lambda(Head). \tag{4}$$

As proved in (Babovich, Erdem, & Lifschitz 2000), for 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.[4] In the special case when Π is a finite program without classical negation, this theorem shows that X is an answer set for Π iff X satisfies the completion of Π.

For instance, program

$$\begin{array}{l} p \leftarrow not\ q, \\ q \leftarrow not\ p, \\ p \leftarrow p, r \end{array} \tag{5}$$

is tight on each of the two models $\{p\}$, $\{q\}$ of its completion

$$\begin{array}{l} p \equiv \neg q \lor (p \land r), \\ q \equiv \neg p, \\ r \equiv \bot \end{array}$$

(and even on their union: take $\lambda(p) = \lambda(q) = 0$). Accordingly, both models are answer sets for this program.

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

[4]Supportedness (Apt, Blair, & Walker 1988) is a model-theoretic counterpart of completion which can be applied to infinite programs and programs with classical negation (see (Lifschitz & Turner 1999), Section 7).

Theorem 2, discussed in the next section, tells us under what conditions the tightness of a program is preserved when the definition of transitive closure is added to it.

The proposition below gives a simple characterization of tightness that does not refer to ordinals (and is actually close to Fages' original formulation). 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 (1) in Π such that

- $L_1, \ldots, L_m \in X$,
- $L \in \{L_1, \ldots, L_m\}$, and
- $L' = Head$.

For instance, the parents of p relative to (5) and $\{p, q, r\}$ are p and r; on the other hand, p has no parents relative to (5) and $\{p, q\}$.

Proposition 1 *A program Π is tight on a set X of literals iff 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 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.[5]

Proposition 1 is a special case of the following general fact of set theory:

Lemma 1 *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 $\lambda(x) < \lambda(y)$.*

Proof (outline). 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.

Transitive Closure and Completion

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 $L_1, \ldots, L_n \in X$ $(n > 1)$ 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 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. 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

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

The first part of the theorem tells us that, under some conditions, the tightness of a program is preserved when the definition of the transitive closure of a predicate is added. The second part, in application to finite programs without classical negation, tells us that, under some conditions, the answer sets for $\Pi \cup Def$ can be characterized as the models of this program's completion, so that, in any model of completion, the extent of tc is the transitive closure of the extent of p.

Condition (ii) 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

$$p(x_1, x_2), \ldots, p(x_{n-1}, x_n), p(x_n, x_1) \in X. \tag{6}$$

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

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

Proposition 2 *If Π contains constraint*

$$\bot \leftarrow tc(x, x) \tag{7}$$

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 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) can be verified by checking, for instance, that p does not depend positively on tc in the dependency graph of Π. This condition 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 $\Pi \cup Def$, but is is not an answer set for $\Pi \cup Def$: the only answer set for this program is empty.

Proof of Theorem 2. Assume (i)–(iii). To prove the first assertion of the theorem, suppose that $\Pi \cup Def$ is not tight on X. By Proposition 1, there is an infinite sequence $L_0, L_1, \ldots \in X$ such that for every i, L_{i+1} is

[5] 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, $x_{i+1} R x_i$.

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 Π and X. Proposition 1, applied to sequence $L_{n+1}, L_{n+2}, \ldots$, shows that Π 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).

The second assertion of Theorem 2 follows from the first, in view of the theorem from (Babovich, Erdem, & Lifschitz 2000) reviewed above, and by Theorem 1.

Proof of Proposition 2. Let Π be a program containing constraint (7), 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 (6). Since X is closed under Def, $tc(x_1,x_1) \in X$. But this is impossible because X is closed under (7).

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 $0, \ldots, T$,

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 (16) 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:[6]

$$\begin{array}{l} on(b,l,0) \leftarrow not\ \neg on(b,l,0) \\ \neg on(b,l,0) \leftarrow not\ on(b,l,0) \\ move(b,l,t) \leftarrow not\ \neg move(b,l,t) \\ \neg move(b,l,t) \leftarrow not\ move(b,l,t) \end{array} \tag{8}$$

$$\begin{array}{l} on(b,l,t) \leftarrow move(b,l,t) \\ on(b,l,t+1) \leftarrow on(b,l,t), not\ \neg on(b,l,t+1) \end{array} \tag{9}$$

$$\neg on(b,l,t) \leftarrow on(b,l',t) \qquad (l \neq l') \tag{10}$$

$$\bot \leftarrow on(b,b'',t), on(b',b'',t) \qquad (b \neq b') \tag{11}$$

$$\bot \leftarrow move(b,l,t), on(b',b,t) \tag{12}$$

$$\bot \leftarrow move(b,l,t), move(b',l',t) \qquad (b \neq b' \text{ or } l \neq l') \tag{13}$$

$$\begin{array}{l} above(b,l,t) \leftarrow on(b,l,t) \\ above(b,l,t) \leftarrow on(b,b',t), above(b',l,t) \end{array} \tag{14}$$

$$\bot \leftarrow above(b,b,t) \tag{15}$$

$$\bot \leftarrow not\ above(b,table,t) \tag{16}$$

To illustrate the use of Theorem 2, we will prove the following proposition:

Proposition 3 *Program (8)–(16) is tight on every set of literals that is closed under it.*

Since program (8)–(16) contains classical negation, the completion process is not applicable to it directly. But classical negation can be eliminated from the program by replacing $\neg on$ with the auxiliary predicate on' and adding the constraint

$$\bot \leftarrow on(b,l,t), on'(b,l,t).$$

Proposition 3 tells us that, after this transformation, the program's answer sets can be computed by using a propositional solver to find models of the program's completion, as described in (Babovich, Erdem, & Lifschitz 2000).

The idea of the proof is to check first that program (8)–(13), (15), (16) is tight, and then use Theorem 2 to conclude that tightness is preserved when we add the definition (14) 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.

[6]This program is similar to the history program for the blocks world from (Lifschitz 1999). Instead of rules (8), the program in that paper contains a pair of disjunctive rules; according to Theorem 1 from (Erdem & Lifschitz 1999), this difference does not affect the program's answer sets. The intuitive meaning of rules (9)–(12) is discussed in (Lifschitz 1999), Section 6. Rule (13) prohibits concurrent actions. Rules (14) and (15) were suggested to us by Norman McCain and Hudson Turner on June 11, 1999; similar rules are discussed in (Lifschitz 1999), Section 8.

We need to introduce a program similar to (8)–(16) in which, syntactically, *table* is allowed as the first argument of both *on* and *above*.

Proof of Proposition 3. Let Π be the program that differs from (8)–(16) in that

- its underlying set of atoms includes, additionally, expressions of the forms $on(table, l, t)$ and $above(table, l, t)$, and
- rules (14) and (15) are replaced by

$$\begin{array}{l} above(l,l',t) \leftarrow on(l,l',t), \\ above(l,l',t) \leftarrow on(l,l'',t), above(l'',l',t) \end{array} \qquad (17)$$

and

$$\bot \leftarrow above(l,l,t). \qquad (18)$$

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 (8)–(16). 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 $k = 0, \ldots, T+1$, let Π_k be the subset of the rules of Π in which rules (17) are restricted to $t < k$. Since $\Pi_{k+1} = \Pi$, 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 (8)–(13), (16) and (18). 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

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

Then $\Pi_{k+1} = \Pi_k \cup Def$. Let us check that all conditions of Theorem 2 are satisfied. Condition (i) holds by the induction hypothesis. Since X is closed under the original program (8)–(16) 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 (18), condition (ii) follows by Proposition 2. By inspection, (iii) holds also. By Theorem 2, it follows that Π_{k+1} is tight on X.

Conclusion

To prove that the completion of a program has no models other than the program's answer sets, we can check that the program is tight. When the program contains the definition of the transitive closure of a predicate, it may be difficult to check its tightness directly. But our Theorem 2 can be sometimes used to show that the tightness of a program is not lost when such a definition is added to it.

Essential for the applicability of that theorem is the presence of the "irreflexivity" constraint from Proposition 2, such as constraint (15) from the blocks world example. There are cases, however, when such constraints cannot be included without distorting the meaning of the program, as, for instance, when the concept of transitive closure is used to talk about paths in an arbitrary graph.

Acknowledgments

We are grateful to Norman McCain and Hudson Turner for their suggestion, mentioned in Footnote ([6]), which served as a starting point for our work on this paper, and for other useful discussions related to the problem of transitive closure in logic programming. This work was partially supported by the National Science Foundation under Grant No. IIS-9732744. 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 Minker, J., ed., *Foundations of Deductive Databases and Logic Programming*. San Mateo, CA: Morgan Kaufmann. 89–148.

Babovich, Y.; Erdem, E.; and Lifschitz, V. 2000. Fages' theorem and answer set programming.[7] In *Proc. NMR-2000*.

Clark, K. 1978. Negation as failure. In Gallaire, H., and Minker, J., eds., *Logic and Data Bases*. New York: Plenum Press. 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.

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

Gelfond, M., and Lifschitz, V. 1990. Logic programs with classical negation. In Warren, D., and Szeredi, P., eds., *Logic Programming: Proc. Seventh Int'l Conf.*, 579–597.

Lifschitz, V., and Turner, H. 1994. Splitting a logic program. In Van Hentenryck, P., ed., *Proc. Eleventh Int'l Conf. on Logic Programming*, 23–37.

Lifschitz, V., and Turner, H. 1999. Representing transition systems by logic programs. In *Logic Programming and Non-monotonic Reasoning: Proc. Fifth Int'l Conf. (Lecture Notes in Artificial Intelligence 1730)*, 92–106.

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

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