On the Equivalence between Answer Sets and Models of Completion for Nested Logic Programs

Jia-Huai You and Li-Yan Yuan
Department of Computing Science
University of Alberta, Canada

Mingyi Zhang
Guizhou Academy of Sciences
RR. China

Abstract

We present a sufficient as well as a necessary condition for the equivalence between answer sets and models of completion for logic programs with nested expressions in the bodies of rules. This condition is the weakest among all that we are aware of even for normal logic programs. To obtain this result, we present a polynomial time reduction from this class of nested logic programs to extended programs. Consequently, answer sets for these nested programs can be computed by an answer set generator for extended programs on the one hand, and characterized in terms of models of completion on the other.

1 Introduction

We consider logic programs with nested expressions in the bodies of rules , called *nested logic programs* in this paper. Nested expressions arc formulas built from literals (atoms with or without classic negation), connectives such as negation as failure *not,* disjunction ";", and conjunction ",". This is a proper subset of the class of programs considered in iLifschitz *et al,* 1999], where the head of a rule can also be an arbitrary formula. Lloyd and Topor [Lloyd and Topor, 1984] defined nested logic programs without negation as failure, and argued for the higher expressive power of the extended language, while Lifschitz et al. further showed the role of negation as failure in nested logic programs.

Our goal in this paper is to provide a characterization of answer sets for nested logic programs in terms of *tightness* on the one hand, and to use an answer set generator, such as Smodels or DLV to compute answer sets for nested logic programs on the other. The studies on characterizations of answer sets have proved useful in relating the completion semantics iClark, 1978] with the answer set semantics [Gelfond and Lifschitz, 1988]. Fages [Fages, 1994] defined a syntactic condition on logic programs, called *positive-order-consistent,* which guarantees the equivalence of the two semantics. Positive-order-consistent requires a program to have no positive loops. For example, the program $\{a \leftarrow a\}$ is not positive-order-consistent. Babovich et al. [Babovich *et al,* 2000] on the other hand formalized the notion, *tight on a set,* which says that an extended logic program II is *tight on X* if there exists a level mapping λ from A" to ordinals, such that for any rule

$$h \leftarrow l_1, ..., l_m, not\ l_{m+1}, ..., not\ l_n$$

in Π, if $\{h\} \cup \{l_1, ..., l_m\} \subseteq X$ then, for each $i \in [1..m]$,

$$\lambda(h) > \lambda(l_i).$$

Erdem and Lifschitz lErdem and Lifschitz, to appear] extended the work to nested logic programs, and weakened the condition so that the positive literals are required to be ordered by λ only for applicable rules w.r.t. A\

As shown in [Babovich *et al,* 2000; Erdem and Lifschitz, to appear], that a program is tight on *S* is a sufficient condition for the equivalence of *S* being an answer set and *S* satisfying the completion of the program. However, there are simple programs that are not tight even on their answer sets. For example, suppose we want to express that two or more propositions are equivalent, and they are true if certain conditions are satisfied. For instance, consider

$$\Pi = \{a \leftarrow b.\ b \leftarrow a.\ b \leftarrow not\ c.\}$$

Clearly, Π is not tight on its unique answer set {a, *b}*. Any level mapping λ that maps the domain {a, *b}* to non-negative integers will have to satisfy $\lambda(a) > \lambda(b)$, as well as $\lambda(b) > \lambda$(a), which is impossible.

Even for logic programs without negation as failure, tightness may fail to apply, e.g., with II = $\{a \leftarrow .\ \ a \leftarrow a.\}$, II is not tight on its unique answer set *{a}*.

In this paper, we define the notion of *weak tightness,* which is a sufficient as well as a necessary condition for a model of completion to be an answer set. Consequently, an answer set is characterized by three independent properties: *weak tightness, supportedness,* and *closedness.* This characterization of answer sets can be generalized to nested programs via a linear time transformation to extended programs. This yields a new characterization of answer sets for nested programs. Although the transformation uses a linear number of extra variables, we will see that these extra variables are *non-split,* in the sense that in a Davis-Putnam procedure [Davis and Putnam, 1960], they need not create choice points during search for answer sets, and thus they need not enlarge the search space.

Our results have some practical implications. First, they widen the application range of systems like Cmodels

[Babovich, 2002], a system implemented for the computation of answer sets for tight logic programs. Secondly, the transformation from nested programs to extended programs allows answer sets for nested programs to be computed by an answer set generator for extended programs, or indirectly via the models of their completed programs.

The next section introduces logic programs with nested expressions. In Section 3 we define weak tightness and show that answer sets are characterized by this condition along with models of completion. In Section 4 we present a polynomial time transformation from nested programs to extended programs. Section 5 remarks on related work.

2 Nested Logic Programs

We consider a propositional language *L*. A *literal* is an atom, or an atom with the sign ¬ in front, called the *classical negation. Elementary formulas* are literals and constants $\top$ **("true") and** $\bot$ **("false").** Formulas are built from elementary formulas using the unary connective *not* and the binary connectives, (conjunction) and ; (disjunction). A nested logic program (or, just program if not otherwise specified) is a set of rules of the form

$$H \leftarrow G$$

where *H* is a literal or $\bot$, and *G* is a formula. *H* is called the *head* of the rule and *G* the *body.* If $G = \top$, we will drop it from the body; rules with the body $\top$ are called *facts.*

Here are some special cases. A formula of the form *not* ϕ, where ϕ is a literal, is called a *default negation.* When *G* is a conjunction of literals and default negations, the rule is called an *extended rule.* In an extended rule, the literals that do not appear in the scope of *not* are called *positive literals.* The programs that consist of only extended rules are called *extended programs;* and extended logic programs without classic negation arc called *normal programs.*

An extended rule may be denoted by $H \leftarrow B^+, B^-$, where B^+ is the set of positive literals and B- the set of default negations.

We denote by $lit(\Phi)$ the set of literals occurring in Φ, where Φ is any syntactic entity, such as a formula or a program. Let A" be a set of literals. $X_{|V}$ denotes the subset of A^r, restricted to the literals in *V,* i.e., $X_{|V} = X \cap V$. A set of elementary formulas A' is *consistent* if i. *tf X* and it contains no complementary literals / and $\neg l$.

That a consistent set of literals *X* satisfies a formula F, denoted $X \models F$, is defined recursively as follows:

- for elementary formula 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 A^r of literals is *closed under* a program II if, for every rule $H \leftarrow G$ in Π, $H \in X$ whenever $X \models G$. *X* is *supported by* Π if, for any / $\in X$ there is a rule / $\leftarrow$ *G* such that $X \models G$.

We now define answer sets for a program. For a program Π without negation as failure, a consistent set of literals *X* is an answer set for II if and only if *X* is *minimal* among the consistent sets of literals closed under II.

A consistent set of literals A' is an answer set of program Π if and only if A" is *minimal* among the consistent sets of literals closed under Π^X, where the *reduct* Π^X is obtained from Π by replacing every *maximal* occurrence of a formula *not F* by $\bot$ if $X \models F$ and $\top$ otherwise.

We define the completion semantics for finite nested programs without the classic negation. Let II be such a program. The *completion* of Π, denoted *Comp*(Π), is the following set of propositional formulas: for each atom ϕ which is either an atom in L, or $\bot$,

- if ϕ does not appear as the head of any rule in IT, then $\phi \equiv \bot \in Comp(\Pi)$;
- otherwise, $\phi \equiv Body_1 \vee ... \vee Body_n \in Comp(\Pi)$ (with each default negation *not* ξ replaced by negative literal $\neg\xi$), if there are exactly *n* rules $\phi \leftarrow Body_i \in \Pi$ with ϕ as the head.

Finally, given a nested program Π without negation as failure, to construct the unique minimal set among the consistent sets of literals closed under II, we define the familiar T_Π operator as

$$T_\Pi(S) = \{l \mid l \leftarrow G \in \Pi \text{ and } S \models G\}$$

It is clear that T_Π is monotonic over the domain of sets of literals. The least fixpoint can then be computed by

$$T_\Pi \uparrow 0 = \emptyset$$
$$T_\Pi \uparrow i+1 = T_\Pi(T_\Pi \uparrow i), \quad \text{for all } i \geq 0$$
$$T_\Pi \uparrow \infty = \bigcup_{i=0}^{\infty} T_\Pi \uparrow i$$

The following proposition is needed later in this paper.

Proposition 2.1 *Let* Π *be a program without negation as failure. Then, X is an answer set for* Π *iff X is consistent and* $X = T_\Pi \uparrow \infty$.

3 Weakly Tight Logic Programs

Let Π be an extended program, A' a set of literals, and *h* a literal in A". We define the set of *applicable rules* of Π, with respect to *h* and A\ as follows;

$$\Pi(h, X) = \{h \leftarrow Body^+, Body^- \in \Pi \mid h \in X \\ Body^+ \subseteq X, \text{ and } \forall not\ \phi \in Body^-, \phi \notin X\}$$

A *level mapping* A is a partial function from literals to natural numbers such that the set A/* of natural numbers mapped from literals forms a partial order $\langle \mathcal{N}, > \rangle$.[1]

Definition 3.1 *Let X be a set of literals and* Π *an extended program.* Π *is said to be* weakly tight *on X if there exists a level mapping* λ *with the domain X such that for any nonempty* $\Pi(h, X)$ *where* $h \in X$, *there is a rule* $h \leftarrow$ *Body* \Body~ in* Π*(h,X) such that for each* $b \in$ *Body*,* $\lambda(h) > \lambda(b)$.

Example 3.2 *Let program* $\Pi = \{a \leftarrow b.\ b \leftarrow a.\ b \leftarrow$ *c, not d.* $c \leftarrow .\}$. *Then,* $S = \{a, b, c\}$ *is the only answer set for* Π. *Though* Π *is not tight on S,* n *is weakly tight on S. For example, a level mapping* A *satisfying the weak tightness condition can be:* $\lambda(a) = 3$, $\lambda(b) = 2$, $\lambda(c) = 1$.

[1] By a partial order, we mean a relation that is transitive and irreflexive.

Lemma 3.3 *Let Π be an extended program, and S an answer set for Π. Then Π is weakly tight on S.*

Proof. Since *S* is an answer set for Π, $S — T_{\Pi^S} \uparrow \infty$. It can be seen that the following mapping λ exists: for each literal *q* such that $q \in T_{\Pi^S} \uparrow n$ but $q \notin T_{\Pi^S} \uparrow n-1$, let $\lambda(q) = n$. Clearly, λ satisfies the condition for weak tightness. □

Theorem 3.4 *For any extended program Π and any consistent set of literals S, S is an answer set for Π iff Π is weakly tight on S, and S is closed under and supported by* II.

Proof ($\Rightarrow$) Let *S* be an answer set for Π,. Then $T_{\Pi^S} \uparrow \infty = 5$. It's obvious that *S* is closed under and supported by IT From Lemma 3.3 we know that TI is also weakly tight on *S.*

($\Leftarrow$) Assume Π is weakly tight on S, and *S* is closed under and supported by II. We show that *S* is an answer set for Π. Let $M = T_{M^S} \uparrow \infty$. The condition that *S* is closed implies $M \subseteq S$. We are done if M - *S.* Otherwise we have $M \subset$ S. Let $q \in S \backslash M$. Since *S* is supported by II, there is a rule $r : q \leftarrow Body^+, Body^-$ in Π such that $q \leftarrow Body^+ \in \Pi^S$ and $Body^+ \subseteq S$. Thus, the same rule r is in II *(q,* S), which is nonempty. As II is weakly tight on 5, there is a mapping λ and a rule $q \leftarrow Body'^+, Body'^-$ in $\Pi(q, S)$ such that $q \leftarrow Body'^+ \in \Pi^S$ and $\lambda(q) > \lambda(b)$ for each $b \in Body^+$. Since *M* is also closed, there is $b \in Body^+$ such that $b \in S \backslash M$. Then, from the definition of λ, there is a literal $p \in S \backslash M$ such that $\lambda(p)$ is minimal, hence $p \in T_{\Pi^S} \uparrow 1$. This contradicts the assumption that *M* is the least fixpoint of the operator T_{Π^S} □

Note that neither supportcdness nor closedness can be removed from the theorem for it still to hold. E.g. the program $\Pi = \{a \leftarrow b\}$ is tight as well as weakly tight on {a,/;}, which is not supported by Π. For closedness, we note that any program is tight as well as weakly tight on the empty set.

Corollary 3.5 *Let Π be a finite normal program and S a set of atoms satisfying Comp(Π). Then, Π is weakly tight on S ifj'S is an answer set for Π.*

To compare two conditions that guarantee a model of completion to be an answer set, we have the following definition.

Deiinilion 3.6 *Let Cond*\ (A\ *Y) and Cond'2(X,Y) be conditions where X ranges over the set of all extended programs and Y over the set of all sets of literals. Cond**(X, Y) is said to be* weaker than *Cond2(X, Y) if*

- *for all X and Y, Cond2(X, Y) implies Comll(X, Y), and*
- *there exist some X and Y such that Condi* (A[r], V) *does not imply Cond2{X, Y).*

Proposition 3.7 *For any extended program Π and set of literals X, if Π is tight on X then it is weakly tight on X. But the converse is not true.*

In Section 5, we will see that weak tightness is also weaker than other conditions used to characterize answer sets in the literature. Then, the question arises as whether the weak tightness condition is the weakest. Technically, the answer is *no,* since the weak tightness condition can be made weaker in trivial ways, e.g., one can make it weaker by not requiring a level mapping to be applied to a rule with a body literal that does not appear in the head of any rule.

4 Transformation

When a program II is translated to another program II[7], new propositional symbols may be used. Thus, the language for Π' is enlarged. Below, we define a notion of equivalence between a program and its translation. Throughout this section, a program means a finite nested program.

Definition 4.1 *Let Π be a program in language L. Suppose the language V for Π' is a superset of L. Π'' is* equivalent *to Π, denoted $\Pi \sqsubseteq \Pi'$, if and only if both of the following statements hold.*

- *For any answer set S for Π, there is exactly one answer set S' for Π' such that $S'_{|L} = S$.*
- *For any answer set S' for Π', $S'_{|L}$ is an answer set for Π.*

Lemma 4.2 *The relation $\sqsubseteq$ is reflexive and transitive.*

Our transformation makes use of some transformations in [Lifschitz *et ai,* 19991 that preserve strong equivalence. Two programs Π_1 and Π_2 are said to be *strongly equivalent* if for any program P, $\Pi_1 \cup P$ and $\Pi_2 \cup P$ have the same set of answer sets. Strong equivalence entails our notion of equivalence.

Proposition 4.3 *If two programs Π_1 and Π_2 are strongly equivalent, then they are equivalent to each other, i.e., $\Pi_1 \sqsubseteq \Pi_2$ and $\Pi_2 \sqsubseteq \Pi_1$*

4.1 Transformation

In the following, we say that a formula is *basic* if it is built from [1]Y\ [4]7\ literals, constants, and *not L* and *not not L* where *L* is a literal or constant. A program is said to be *basic* if it consists of rules whose body formulas are basic. A rule

$$H \leftarrow G_1; \ldots; G_n.$$

is said to be *fiat* if every G\ is a conjunction of literals and default negations. A program *sflat* if every rule in it is flat.

Let II be a nested program. Our transformation consists of three steps, briefly described below.

- II is transformed to a basic program II' by some simplification rules.
- IT is transformed to a flat program II" by transformation rules that name some subexpressions by new atoms.
- II" is then transformed to Π^{nf} by splitting each flat rule with disjunctive body into several extended rules with the same head.

We now give the details. Below, $F \Rightarrow R$ denotes a rewrite rule.

Step 1. Simplification

1. $not\ (F;G) \Rightarrow not\ F, not\ G$
2. $not\ (F,G) \Rightarrow not\ F; not\ G$
3. $not\ not\ not\ F \Rightarrow not\ F$

4. $F, \top \Rightarrow F$ (and its symmetric case, $\top, F \Rightarrow F$; the same for all rewrite rules below)
5. $F; \top \Rightarrow \top$
6. $F, \bot \Rightarrow \bot$
7. $F; \bot \Rightarrow F$
8. $p, \neg p \Rightarrow \bot$
9. $not\ p; not\ \neg p \Rightarrow \top$

where Rule 3 has higher priority to be applied than Rules 1 and 2. □

The simplification rules given here are essentially their corresponding equivalences given in [Lifschitz^/a/., 1999]. The only difference is that, for the purpose of termination, we cannot directly adopt an equivalence for symmetry, such as $F, G \Leftrightarrow G, F$, as a rewrite rule. We therefore have to represent the symmetric cases by rules.

The simplification rules are similar to the ones found in propositional logic, except perhaps Rule 3, which simplifies negation as failure only when it is nested at least three levels deep. Among these rules, only 1-3 are essential for the purpose of producing a basic program. The rest are used to simplify basic formulas further.

That a rule has a higher priority to be applied than another rule means that when both rules are applicable, the former should be applied and the latter should not. The only intention here for Rule 3 over Rules 1 and 2 is to prevent transforming an expression of the form

$$\underbrace{not \cdots not}_{n} F$$

where $n > 2$, by distributing each *not* over *F* (which would result in higher complexity).

Step 2. Naming subexpressions

(a) $F, (G; H) \Rightarrow F, \phi$
and add the rule $\phi \leftarrow G; H$ where ϕ is a new atom.[2]

(b) $(G; H), F \Rightarrow \phi, F$
and add the rule $\phi \leftarrow G; H$ where ϕ is a new atom.

(c) $not\ not\ p \Rightarrow not\ \phi$
and add the rule $\phi \leftarrow not\ p$ where ϕ is a new atom. □

Step 3. Splitting

A flat rule of form $H \leftarrow G_1; ...; G_n$ is replaced by rules

$$H \leftarrow G_1.$$

$$H \leftarrow G_n.$$

The set of all transformation rules in Steps 1, 2 and 3 define a binary relation over programs. We will use $\Pi \succ_i \Pi'$ to denote a single transformation step that transforms program n to program Π' by an application of a transformation rule in Step *i* where $i \in [1..3]$. By a *rewrite sequence,* we mean a sequence of zero or more rewrite steps which takes the general form

$$\Pi_0 \succ_{i_0} \ldots \succ_{i_{n-1}} \Pi_n \succ_{i_n} \ldots$$

[2]Apparently, in this case all the occurrences of *{G\ H)* in the given program can be replaced by ϕ simultaneously.

A program Π is called an ω_i*-normal form* if Π cannot be reduced further by transformation rules in Step i where $i \in$ [1..3]. In addition, we denote by

$$\Pi \succ_i^* \Pi'$$

a rewrite sequence generated only by the transformation rules in Step i.

Example 4.4 *Suppose* Π *contains a single rule*

$$a \leftarrow ((not\ not\ b);\ c),\ not\ d.$$

Π *is an* ω_1*-normal form. We can have* $\Pi \succ_2 \Pi'$ *where* $\Pi' = \{a \leftarrow \xi,\ not\ d.\ \xi \leftarrow not\ not\ b;\ c.\}$, *and* $\Pi' \succ_2 \Pi''$ *where* $\Pi'' = \{a \leftarrow \xi,\ not\ d.\ \xi \leftarrow not\ \eta;\ c.\ \eta \leftarrow not\ b.\}$. *Finally,* $\Pi'' \succ_3 \Pi'''$ *where* $\Pi''' = \{a \leftarrow \xi,\ not\ d.\ \xi \leftarrow not\ \eta.\ \xi \leftarrow c.\ \eta \leftarrow not\ b.\}$.

Lemma 4.5 *There is no infinite sequence*

$$\Pi_0 \succ_i \ldots \succ_i \Pi_n \succ_i \ldots$$

where Π_0 *is a nested program when* $i = 1$, Π_0 *is an* ω_1*-normal form when* $i = 2$, *and* Π_0 *is an* ω_2*-normal form when* $i = 3$.

Lemma 4.6 *Let n be the size of the program* Π_0 *mentioned below.*

1. *For any nested program* Π_0, *and any rewrite sequence* $\Pi_0 \succ_1 \ldots \succ_1 \Pi_m$ *where* Π_m *is an* ω_1*-normal form,* Π_m *is a basic program, and m is bounded by* $O(n)$.
2. *For any* ω_1*-normal form* Π_0, *and any rewrite sequence* $\Pi_0 \succ_2 \ldots \succ_2 \Pi_m$ *where* Π_m *is an* ω_2*-normal form,* Π_m *is a flat program, and m is bounded by* $O(n)$.
3. *For any* ω_2*-normal form* Π_0, *and any rewrite sequence* $\Pi_0 \succ_3 \ldots \succ_3 \Pi_m$ *where* Π_m *is an* ω_3*-normal form,* Π_m *is an extended program, and m is bounded by* $O(n)$.

We can show that every transformation in Step 2 preserves equivalence.

Lemma 4.7 *If* $\Pi \succ_2 \Pi'$ *then* $\Pi \sqsubseteq \Pi'$.

For any program II, Lemmas 4.5 and 4.6 together guarantee the existence of a finite rewrite sequence

$$\Pi \succ_1^* \Pi' \succ_2^* \Pi'' \succ_3^* \Pi'''$$

where Π' is an ω_1-normal form, Π'' is an u>2-normal form, and Π''' is an extended program. Let us call such a Π''' the *transformation of* Π, denoted by $Tr(\Pi)$.[3]

We are ready to show the main theorem of this section.

Theorem 4.8 *For any program* Π, $\Pi \sqsubseteq Tr(\Pi)$.

Proof. Assume $\Pi \succ_1^* \Pi' \succ_2^* \Pi'' \succ_3^* \Pi''' = Tr(\Pi)$. From [Lifschitz *et al.*, 1999], Π and Π' are strongly equivalent, so are Π'' and Π'''. It follows that $\Pi \sqsubseteq \Pi'$ and $\Pi'' \sqsubseteq \Pi'''$. Lemma 4.7 shows that every individual rewrite step in $\Pi' \succ_2^* \Pi''$ preserves equivalence. By transitivity (Lemma 4.2) we get $\Pi' \sqsubseteq \Pi''$, and therefore $\Pi \sqsubseteq Tr(\Pi)$. □

[3]Although transformation rules can be applied nondcterministically, it is easy to show that the rewrite relation $\succ_i^*$ for any $i \in [1..3]$ is *confluent* [Dershowitz and Jouannaud, 1990J so that the final extended program is unique, up to atom renaming.

Corollary 4.9 *For any programs* II *and P,*

1. $\Pi \cup P \sqsubseteq Tr(\Pi) \cup P$
2. $\Pi \cup P \sqsubseteq Tr(\Pi) \cup Tr(P)$, *if the sets of new atoms,* $lit(Tr(\Pi)) - lit(\Pi)$ *and* $lit(Tr(P)) - lit(P)$, *are disjoint.*

The claim in 1 above is considered by some authors as a form of *strong equivalence within the same language.*

It follows from Lemma 4.6 that

Theorem 4.10 *For any program* Π, *there is a linear time reduction of* Π *to an extended program* 11' *so that the answer sets for* Π *can he identified in linear time from those for* Π'.

We remark that the extra atoms introduced in our transformation are *non-split.* They are just "connecting atoms" serving the purpose of propagating values from subexpressions. Their values can be determined solely by constraint propagation from the values of the original atoms.

Non-split variables can be accommodated easily in systems like Smodels by applying the procedure *Heuristic* only to the set of original variables, so that non-split variables will not be picked up for guessing a value. In this way, non-split variables do not create *choice points* during the search for answer sets.

4.2 Weakly tight nested logic programs

We give a characterization of answer sets for nested programs, in terms of their transformations, and show the counterpart of Theorem 3.4 for nested programs.

Definition 4.11 *Let* Π *be a nested program and* $Tr(\Pi)$ *its transformation. Let* $X \subseteq lit(\Pi)$ *and* V *satisfy*

$$V = \{\phi \in lit(Tr(\Pi)) \mid \phi \notin lit(\Pi), \phi \in T_{Tr(\Pi)^{X \cup V}} \uparrow \infty\}.$$

Then, Π *is said to be weakly tight on* X *if and only if* $Tr(\Pi)$ *is weakly tight on* $X \cup V$.

Theorem 4.12 *Let* Π *be a nested program, and X a consistent set of literals. X is an answer set for* II *iff* Π *is weakly tight on X, and X is closed under and supported by* II.

Corollary 4.13 *Let* Π *be a finite nested program without the classic negation and S a set of atoms satisfying Co7np(U). Then, U is weakly tight on SiffS is an answer set for* II.

5 Related Work

Well-Supported Model

Definition 5.1 [Fages, 1991] *Let* Π *be a normal program. A set of atoms S is* well-supported *iff there exists a level mapping* λ *with S as the domain such that for any atom* $a \in S$ *there exists a rule* $a \leftarrow Body^+, Body^- \in \Pi$, *which is applicable w.r.t. S, and for each* $p \in Body^+$, $\lambda(a) > \lambda(p)$.

Ben-Eliyahu and Dechter's characterization [Ben-Eliyahu and Dechter, 1994] of answer sets for head cycle-free disjunctive programs reduces to that of Fages for normal programs.

Theorem 5.2 [Fages, 1991] *For any normal program* II, *the well-supported models ofU are exactly the answer sets for* II.

The level mapping in Fages' definition depends on the supportedness: one needs to find a rule that supports an atom, and then determines if a level mapping applies to the same rule. The two conditions in our case are independent.

Example 5.3 *Let* $\Pi = \{a \leftarrow b;\ b \leftarrow c\}$, *and* $S = \{a, b, c\}$. *S is a model of* Π. Π *is weakly tight (as well as tight) on S, since we can assign a level mapping as:* $\lambda(a) = 2$, $\lambda(b) = 1$, *and* $\lambda(c) = 0$. *But S is not well-supported by* Π.

To summarize the relationships among tightness, weak tightness and well-supportedness for normal programs, we have, for any normal program Π and set of atoms 5,

- If 5 is well-supported, then Π is weakly tight on 5; but the converse is not true.
- If Π is tight on 5, then Π is weakly tight on 5; but the converse is not true.
- That Π is tight on *S* is not a sufficient condition for *S* to be well-supported; that *S* is well-supported is not a sufficient condition for Π to be tight on *S.*

Therefore, the weak tightness is the weakest among all three, bridging the gap between tightness and well-supportedness.

Strong Compatibility of Logic Programs

In [Zhang, 1992], Zhang showed a characterization of default extensions. Here, we re-formulate it for logic programs.

For an extended logic program Π, let us define

$$Con(\Pi) = \{H \mid H \leftarrow B^+, B^- \in \Pi\}$$
$$Pre(\Pi) = \{L_i \mid 1 \le i \le m, \quad H \leftarrow L_1, ..., L_m, not\ L_{m+1}, ..., not\ L_n \in \Pi\}$$
$$Just(\Pi) = \{\neg L_i \mid m+1 \le i \le n, \quad H \leftarrow L_1, ..., L_m, not\ L_{m+1}, ..., not\ L_n \in \Pi\}$$

A subprogram $\Pi' \subseteq \Pi$ is said to be *compatible* (w.r.t. Π) if $Con(\Pi') \cup Just(\Pi')$ is consistent and $\bot \notin Con(\Pi')$.

For any $\Pi' \subseteq \Pi$, we let $\Lambda(\Pi') = \bigcup_{i=0}^{\infty} R_i(\Pi')$, where

$$R_0(\Pi') = \{H \leftarrow not\ L_1, ..., not\ L_n \in \Pi' \mid n \ge 0\},$$
and for all $k \ge 0$,
$$R_k(\Pi') = \{H \leftarrow B^+, B^- \in \Pi' \mid B^+ \subseteq \bigcup_{i=0}^{k-1} Con(R_i(\Pi'))\}$$

The characterization of default extensions as given in [Zhang, 1992], when applied to logic programs, can be stated as follows.

Theorem 5.4 *Let U be an extended logic program. U has an answer set iff there is a subprogram* II' C 17 *such that*

P1. Π' *is compatible,*

P2. $\Lambda(\Pi') = \Pi'$, *and*

P3. for any rule $r \in \Pi - \Pi'$, *either* $\Lambda(\Pi' \cup \{r\}) \neq \Pi' \cup \{r\}$, *or* $\Pi' \cup \{r\}$ *is not compatible.*

A subprogram $\Pi' \subseteq \Pi$ is said to be *strongly compatible* if it is compatible and $\Lambda(\Pi') = \Pi'$. We can show the following relationship.

Proposition 5.5 *Let* Π *be an extended program. For any* $\Pi' \subseteq \Pi$,

1. *if Π' is strongly compatible, then Π is weakly tight on* Con(n');

2. *Π is weakly tight on $Con(\Pi')$, and $Con(\Pi')$ is closed under and supported by Π, iff Π' is strongly compatible and satisfies P3 as well.*

Since no proper subset of $\{P1, P2, P3\}$ corresponds to the satisfaction of completion, to see whether a model of completion is also an answer set one needs all three conditions. The key lies in condition *P2* which embodies both supportedness and an order of derivation. Thus, Zhang's characterization does not provide a nontrivial condition for a model of completion to be an answer set.

Note that $P2$ alone does not imply weak tightness. For example, with $\Pi = \{a \leftarrow a;\ a \leftarrow not\ a\}$, we have $\Lambda(\Pi) = \Pi$. But Π is not weakly tight on $Con(\Pi) = \{a\}$, due to the fact that $a \leftarrow a$ is the only applicable rule w.r.t. $\{a\}$.

ASSAT

In the system ASSAT iLin and Zhou, 2002], the completion of a normal program is executed by a SAT solver, possibly repeatedly. When a model *M* of the completion from program II is generated, if $M \neq \Pi^M$ then *M* is not an answer set for II. ASSAT relies on $M - \Pi^M$ to compute loop formulas which say that the atoms in $M - \Pi^M$ that are involved in a (maximal) loop should be false if they cannot be derived otherwise. However, $M - \Pi^M$ does not provide a complete picture of the loops involved. There could be loops whose interpretation in *M* is faked due to the loops on A7 - $\Pi^{M/}$. For example, let $\Pi = \{c \leftarrow c.\ c \leftarrow not\ b.\ b \leftarrow not\ a.\ a \leftarrow a\}$, and let the computed model of completion be $M = \{a, c\}$. In this case, $M - \Pi^M = \{a\}$. Though the loop on *a* is captured correctly, the loop on *c* is captured incorrectly. It remains to be seen whether stronger loop formulas can be computed, based on the idea of level mapping.

Transformations of Nested Programs

The transformation given in [Lifschitz *et al,* 19991 deals with arbitrary nested programs where rules may have arbitrary formulas in the heads. The transformation preserves strong equivalence but is not polynomial.

Independently in [Pearce *et al.,* 2002], arbitrary nested programs are translated to disjunctive programs where extra variables arc used to avoid an exponential blow-up, an idea similar to the one adopted in this paper. Our transformation however departs from that of [Lifschitz *et al,* 1999] in a minimal way. By partitioning the transformation into three stages (cf. Theorem 4.8), we are able to identify exactly where extra variables are required. This is only when dealing with a disjunction or an expression like *not not L* where *L* is a literal. In [Pearce *et al.,* 2002], other forms of subformulas, such as the whole body of a rule and a conjunction, can also be labeled by extra variables. Our transformation is thus simpler and more compact. While our transformation takes linear time, the one in [Pearce *et al,* 2002] is only claimed to be polynomial time. Finally, the extra variables introduced in our case are non-split. It is not clear whether this is possible for the class of all nested programs where the heads of rules are arbitrary formulas. It is interesting to see whether our approach, combined with some of the ideas in [Pearce *et al,* 2002], can be extended in order to be able to handle all nested programs.

References

[Babovich et al., 20001 Y. Babovich, E. Erdem, and V. Lifschitz. Fages' theorem and answer set programming. In *Proc. 8th International Workshop on Non-Monotonic Reasoning,* 2000.

[Babovich, 2002] Y. Babovich. Cmodels home papge at www.cs.utexas.edu/users/tag/cmodcls.html. 2002.

iBen-Eliyahu and Dechter, 1994] R. Ben-Eliyahu and R. Dechter. Propositional semantics for disjunctive logic programs. *Annals of Math, and Artificial Intelligence,* 14:53-87, 1994.

[Clark, 1978] K.L. Clark. Negation as failure. In H. Gallaire and J. Minker, editors, *Logic and Data Bases,* pages 293-322. Plenum Press, 1978.

[Davis and Putnam, 1960] M. Davis and H. Putnam. A computing procedure for quantification theory. *J ACM,* 7(3):201-215, 1960.

[Dershowitz and Jouannaud, 19901 N. Dershowitz and P Jouannaud. Rewrite systems. In *Handbook of Theoretical Computer Science, Vol B: Formal Methods and Semantics,* pages 243-320. North-Holland, 1990.

[Erdem and Lifschitz, to appear] E. Erdem and V. Lifschitz. Tight logic programs. *Theory and Practice of Logic Programming,* to appear.

[Fages, 1991] F. Fages. A fixpoint semantics for general logic programs compared with the well-supported and stable model semantics. *New Generation Computing,* 9:425-443,1991.

[Fages, 1994] F. Fages. Consistency of Clark's completion and existence of stable models. *Journal of Methods of U)gic In Computer Science,* 1:51-60, 1994.

[Gelfond and Lifschitz, 1988] M. Gclfond and V. Lifschitz. The stable model semantics for logic programming. In *Proc. 5th ICLP,* pages 1070-1080. MIT Press, 1988.

[Lifschitz *etal,* 1999] V. Lifschitz, L. Tang, and H. Turner. Nested expressions in logic programs. *Annals of Mathematics and Artificial Intelligence,* 25:369-389, 1999.

[Lin and Zhou, 20021 F. Lin and Y. Zhou. ASSAT, computing answer set of a logic program by SAT solvers. In *Proc. AAAI V2,* pages 112-118, 2002.

[Lloyd and Topor, 1984] J Lloyd and R. Topor. Making Prolog more expressive. *J. Logic Programming,* 1(3):225-240, 1984.

[Pearce *etal.,* 20021 D. Pearce, V. Sarsakov, T. Schaub, H. Tompits, and S. Woltran. A polynomial translation of logic programs with nested expressions into disjunctive logic programs (preliminary report). In *Proc. 19th ICLP,* 2002.

[Zhang, 1992] M. Zhang. A characterization of extensions of general default theories. In *Proc.9th Canadian Artificial Intelligence Conference,* pages 134-139, 1992.