Answer Sets through G_3 Logic

Juan Antonio Navarro Pérez
Universidad de las Américas
ma108907@email.com

Abstract. We review some interesting connections between the notion of answer sets for logic programs and some intermediate propositional logics. In particular the G_3 logic is used to study several properties of answer sets as a new approach to understand and solve ASP problems.

There is a characterization of answer sets in terms of the G_3 logic. We discuss how this characterization can be used to generalize the notion of answer sets for programs with arbitrary propositional formulas as rules. Finally we prove how the notion of strong equivalence between programs corresponds to equivalence in G_3 under this generalized definition.

1 Introduction

Logic programming offers a natural way to represent declarative knowledge using the language of logic. Facts and relations between abstract objects are stated in logic programs in order to answer queries about given information and find solutions to particular problems.

The answer set semantics, or stable model semantics, was introduced in (Gelfond and Lifschitz 1988) to provide a declarative meaning for logic programs with negation. The list of practical applications of answer sets has been growing fast since then. Efficient software for computing answer sets is now available offering an alternative logic programming parading. Common examples of such systems are DLV and SMODELS[1].

The availability of "negation as failure" in the answer set semantics makes it also easy to specify default knowledge and produce nonmonotonic reasoning in logic programming. There has been great success using answer sets to, for example, state and solve planning problems, model the behaviour of logical agents and artificial intelligence in general.

Most research on answer sets supposes syntactically simple rules, such as disjunctive rules, to construct logic programs. This is justified since, most of the times, those restricted syntaxes are enough to represent a wide class of

[1] http://www.dbai.tuwien.ac.at/proj/dlv/ http://saturn.hut.fi/pub/smodels/

Proceedings of the Seventh ESSLLI Student Session
Malvina Nissim (editor)

interesting and relevant problems. It could seem unnecessary to generalize the notion of answer sets to some more complicated formulas.

A broader syntax for rules will bring, however, some benefits. The use of nested expressions, for example, could simplify the task of writing logic programs and improve their readability. It will allow us to write more concise rules and in a more natural way. It could even, in some cases, improve computability of models reducing redundant information and avoiding calculation of intermediate auxiliary predicates that might not be needed.

A definition of answer sets for rules containing nested expressions is provided in (Lifschitz, Tang, and Turner 1999). This definition follows the same spirit of the original definition computing answer sets as the fixed points of some operator applied over models of the logic program. We propose to use a characterization of answer sets in terms of the Gödel's 3-valued G_3 logic to generalize a bit more the syntax of programs allowing arbitrary propositional formulas. Formulas could now contain, for instance, embedded implications inside rules.

Consider for example the following pair of rules:

$$p(X) \leftarrow not\, o(X).$$
$$o(X) \leftarrow p(Y) \wedge (X \neq Y).$$

They are commonly used in answer set programming to state that the property p is only satisfied by one unique element. This expression has, however, several disadvantages. The property of uniqueness is written in a very non-standard way, it is almost unreadable for a non-experienced logic programmer. The need of an auxiliary predicate o to define the behaviour of p also seems unreasonable. A more natural way to denote such property, taking advantage of embedded implications, could be for example

$$p(X) \leftarrow \forall y(p(Y) \rightarrow X = Y).$$

where $\forall y F(Y)$ is an abbreviation of $F(y_1) \wedge \cdots \wedge F(y_n)$ for each constant y_i in the language of the program.

Surprisingly, relations between logic programming and logic are rather new. A first characterization of answer sets, for disjunctive programs, in terms of intermediate logics was given in (Pearce 1999). This result was generalized later for programs with nested expressions in (Osorio, Navarro, and Arrazola 2002a) where answer sets are identified with consistent and complete extensions of logic programs obtained by adding negated and twice negated literals to the program.

Results of this nature are very interesting, and one could even expect them, since intermediate logics were intended to model concepts like "knowledge" and "provability" rather that "truth" from classical logic. Answer sets, as we know, are also intended to be a tool for knowledge representation.

We would like to study then some properties of answer sets under this new proposed definition. There is some particular interest on equivalence

between programs. Strong equivalence, as defined in (Lifschitz, Pearce, and Valverde 2001), was shown to be closely related to equivalence with respect to G_3 logic. It this paper we will prove that, as one would like to expect, this result still holds for answer sets with respect to our new definition.

In a recent conference on logic programming R. Kowalski stated that "Logic and LP need to be put into place: Logic within the thinking component of the observation-thought-action cycle of a single agent, and LP within the belief component of thought" (Kowalski 2001). Results on this paper follow this same direction, showing how logic can provide a different point of view to study answer set related problems.

As usual in answer set programming we restrict our attention to propositional programs. The semantics of answer sets is extended to programs with variables by grounding. As Lifschitz noted in (Lifschitz 1996) "We treat programs as propositional objects; rules with variables are viewed as *schemata* that represent their ground instances." Function symbols are not allowed, however, so that the ground instance of a program is always finite.

2 Background

2.1 Syntax of Programs

We consider a *signature* $\mathcal{L}$, which consists of a finite set of *literals*. A literal is either an atom a or atom with explicit negation $\neg a$. A set of literals A is said to be consistent if there is no atom a such that both a and $\neg a$ are contained in A.

Formulas are built as usual in logic from the 0-place connective $\bot$ and the binary connectives $\wedge$, $\vee$, $\leftarrow$. The default negation $not\, F$ is introduced as an abbreviation of $\bot \leftarrow F$, and equivalence $F \equiv G$ as an abbreviation of $(F \leftarrow G) \wedge (G \leftarrow F)$. Also $\top$ abbreviates $not\, \bot$ and $F \rightarrow G$ is just another way of writing $G \leftarrow F$.

Given a set of formulas P we will use $not\, P$ to denote the set of formulas $\{not\, F \mid F \in P\}$. Also, if $P = \{F_1, \ldots, F_n\}$ is a finite set of formulas, we define $\bigwedge P = F_1 \wedge \cdots \wedge F_n$ and $\bigvee P = F_1 \vee \cdots \vee F_n$.

A *logic program* is a finite set of formulas. Formulas that are contained in a program are also referred as *rules*. If P is a program then $\mathcal{L}_P$ denotes its signature, the set of literals occurring in P. For a set of literals $M \subset \mathcal{L}_P$ its complement with respect to P is defined as the set $\widetilde{M} = \mathcal{L}_P \setminus M$.

Typical rules in logic programs have the form $H \leftarrow B$ where H and B are known as the *head* and the *body* of the rule respectively. A rule with an empty head, $\leftarrow B$, is know as a *constraint* and abbreviates the rule $\bot \leftarrow B$.

A *disjunctive rule* is one where the body contains no more than a conjunction of (possibly) negated literals and a simple disjunction of literals in the head. Current popular software implementations to compute answer sets support disjunctive programs with constraints.

A *nested formula* is a formula built (only) with the connectives $\wedge$, $\vee$ and *not* arbitrarily nested. A *nested program* allows nested formulas at the head and the body of rules. A definition of answer sets for this kind of programs is given in (Lifschitz, Tang, and Turner 1999).

One of the purposes of this paper is to define answer sets for programs having arbitrary propositional formulas as rules. The general term *logic program* will be used to denote this unrestricted class of programs. Observe that the main difference between nested and logic programs is that the later allows the insertion of embedded implications as subformulas.

2.2 Answer Sets

Since the actual definition of answer sets, given in (Lifschitz, Tang, and Turner 1999), will not be needed for the purpose of this paper, just a brief description of it is given now. For a basic program P, which does not contain default negation (*not*) and within the class of nested formulas, it is said that a consistent set of literals M is an answer set of P if M is a minimal model of P in the sense of classical logic.

To extend this definition for programs with default negation, the authors in (Lifschitz, Tang, and Turner 1999) introduced the notion of the *reduct* for a nested program P with respect to a consistent set of literals M. This reduct is a program, denoted by P^M, where each subformula *not* F contained in P is replaced by $\bot$ when M is a model of F in the sense of classical logic, and replaced by $\top$ otherwise. Finally M is said to be an *answer set* for a nested program P if it is an answer set for the reduct P^M.

2.3 Intermediate Logics

Intuitionistic logic I is a widely studied logic that can be defined in terms of an axiomatization, Kripke models or natural deduction. This logic, which intends to study the notion of "proof" or "knowledge", seems to be the weakest logic having interesting properties. Any logic whose set of theorems (provable formulas) is properly included in the set of classical theorems, but contains at least those of intuitionistic logic, is called an intermediate logic.

Some well know intermediate logics are the multivalued Gödel logics G_i. An interpretation I in one of such logics is a function that maps each literal in $\mathcal{L}$ to a value in the set $\{0, 1, \ldots, i-1\}$. The evaluation of an interpretation is extended recursively for arbitrary formulas as follows:

- $I(\bot) = 0$.
- $I(A \rightarrow B) = i - 1$ if $I(A) \leq I(B)$, and $I(A \rightarrow B) = I(B)$ otherwise.
- $I(A \vee B) = \max(I(A), I(B))$.
- $I(A \wedge B) = \min(I(A), I(B))$.

Recall that default negation $not\, F$, introduced as $\bot \leftarrow F$, corresponds to the usual negation defined in these logics. To introduce explicit negation we could consider $\neg a$ just as another "atom" conveniently named so that an answer set computer will automatically discard models where both a and $\neg a$ appear together. Formally we will assume all interpretations are consistent.

Observe that, by definition, an interpretation assigns values to each literal a and $\neg a$ independently. We say an interpretation is *consistent* if it evaluates the rule $\leftarrow a \wedge \neg a$ –for each pair $a, \neg a \in \mathcal{L}$– to the *designated* value $i-1$. In terms of valuations this condition states that both literals a and $\neg a$ can not simultaneously have valuations greater than 0.

An interpretation that assigns to literals only the values 0 and $i-1$ will be called *complete*, otherwise it will be referred as *incomplete*. If I is a consistent interpretation, F a formula and $I(F) = i-1$ we will say that I *is a model of* F, or that I *models* F. This definition is extended to programs: I is a model of a program P if it is a model for each rule $F \in P$. Finally we say F is a *tautology*, is *provable* in logic G_i, if every consistent interpretation I is a model of F.

Notice G_2 is classical logic, where all interpretations are complete. We found particularly useful the case G_3 that turns out to be the strongest intermediate logic. This logic can be alternatively defined using Kripke models as HT the logic with two worlds. Another way to introduce G_3 is through an axiomatization. The logic Sm, equivalent to G_3, is obtained by adding the axiom scheme $((not\, G \rightarrow F) \rightarrow (((F \rightarrow G) \rightarrow F) \rightarrow F)$ to a set of axioms for the logic I.

Some basic notation is now defined. As usual we denote by $\vdash_{\mathrm{X}} F$ the fact that the formula F is *provable* in the logic X. If Γ is a set of formulas, then $\Gamma \vdash_{\mathrm{X}} F$ means F is a *logical consequence* of formulas contained in Γ. Using the deduction theorem –valid in all intermediate logics– this consequence relation is equivalent, when Γ is a finite set of formulas, to $\vdash_{\mathrm{X}} \bigwedge \Gamma \rightarrow F$.

A program P is said to be *consistent* if it is not the case that $P \vdash_{\mathrm{X}} \bot$. Also P is said to be *(literal) complete* if either $P \vdash_{\mathrm{X}} a$ or $P \vdash_{\mathrm{X}} not\, a$ for all literals $a \in \mathcal{L}_P$. Observe this definitions have nothing to do with previous definitions for consistent set of literals and consistent interpretations that are related to explicit negation.

The following is not a standard definition, but the reader will see it is convenient for the purpose of this paper. If M is a set of formulas we will use $P \vdash_{\mathrm{X}} M$ to denote $P \vdash_{\mathrm{X}} F$ for all formulas $F \in M$. Also the notation $P \Vdash_{\mathrm{X}} M$ stands for the phrase: P is consistent and $P \vdash_{\mathrm{X}} M$. Now we define, as usual, that two programs P_1 and P_2 are equivalent under logic X, noted $P_1 \equiv_{\mathrm{X}} P_2$, when $P_1 \vdash_{\mathrm{X}} P_2$ and $P_2 \vdash_{\mathrm{X}} P_1$. We state now some basic results in G_3 that relate equivalence with some multivalued interpretations.

Definition 2.1 Let I be an interpretation. Define I' as the interpretation that evaluates each literal $a \in \mathcal{L}$ as $I'(a) = I(not\, not\, a)$.

Observe that, if I is a consistent interpretation then I' is also consistent. Also note that if $I(a) = 1$ then $I(not\, a) = 0$ and $I(not\, not\, a) = 2$. So I' is equivalent to replace each 1-assignment in I to the designated value 2. In particular I' is always a complete interpretation.

Lemma 2.2 (Osorio, Navarro, and Arrazola 2001) *Let F be a formula and I an interpretation, then $I'(F) = I(not\, not\, F)$. In particular, if I models F then I' models F too.*

Proposition 2.3 (Osorio, Navarro, and Arrazola 2001) *Let P_1 and P_2 be logic programs. If $P_1 \not\equiv_{\mathrm{G}_3} P_2$ then there is an interpretation I such that I models P_1 and does not model P_2 (or models P_2 but not P_1).*

3 Characterization of Answer Sets

Pearce provided a characterization of answer sets for disjunctive programs in terms of intermediate logics. He showed that a formula is entailed by a program in the answer set semantics if and only if it belongs to every intuitionistically complete and consistent extension of the program formed by adding only negated literals.

Theorem 3.1 (Pearce 1999) *Let P be a disjunctive program. A consistent set of literals M is an answer set of P if and only if $P \cup not\, \widetilde{M} \vdash_{\mathrm{I}} M$.*

There is a nice reading of previous theorem obtained in the context of answer sets modeling the behavior of a logical agent. Suppose P contains the "base knowledge" of our agent and let M be a consistent set of literals. If it is possible to assume that those literals not contained in M will never hold (the agent "believes" $not\, \widetilde{M}$) in such way that (i) these "beliefs" are consistent with previous "knowledge" and (ii) having this information is just enough to be sure about facts contained in M, then it makes sense that our agent could safely "believe" M.

Pearce also showed that, instead of logic I, any proper intermediate logic could be used. This result can be easily generalized for disjunctive programs containing constraints, but it does not hold any more if we allow negation in the head of rules. Take for example $P = \{a \vee not\, a\}$. It has two answer sets, namely $\{a\}$ and $\emptyset$, but only $M = \emptyset$ satisfies Pearce's condition.

A solution, given in (Osorio, Navarro, and Arrazola 2002a), is to include the set $not\, not\, M$ to complete the extension of the program. In the last example now it is possible to add twice negated literals like $not\, not\, a$ and recover the answer set $M = \{a\}$. The authors in that paper proved this proposed characterization for nested programs.

Theorem 3.2 (Osorio, Navarro, and Arrazola 2002a) *Let P be a nested program. A consistent set of literals M is an answer set of P if and only if $P \cup not\, \widetilde{M} \cup not\, not\, M \vdash_{\mathrm{I}} M$.*

Observe that, using our reading in terms of "knowledge" and "beliefs" of a logical agent, the agent is now allowed to, in some sense, "weakly believe" or "suspect" the facts contained in M (the agent believes $not\, not\, M$) and, if this is enough to be sure about those facts, it could then safely "believe" them. Under this reading we could identify answer sets as "belief sets" for a logical agent.

It is shown in a more recent paper that, just like Pearce noticed for disjunctive programs, it is possible to use, instead of intuitionistic logic, any proper intermediate logic and still obtain the same result. In particular the following theorem was proved.

Theorem 3.3 (Osorio, Navarro, and Arrazola 2002b) *If P is a logic program and M a consistent set of literals, then $P \cup not\, \widetilde{M} \cup not\, not\, M \Vdash_{\mathrm{I}} M$ if and only if $P \cup not\, \widetilde{M} \cup not\, not\, M \Vdash_{\mathrm{G_3}} M$.*

Previous results also suggest a natural way to define answer sets for programs containing arbitrary propositional formulas that, moreover, is equivalent to current definitions for classes of programs considered up to now.

Definition 3.4 Let P be a logic program. A consistent set of literals M is an answer set of P if and only if $P \cup not\, \widetilde{M} \cup not\, not\, M \Vdash_{\mathrm{G_3}} M$.

We will prove now some results, under this new definition of answer sets, which will be used in the next section to obtain a characterization of the so called strong equivalence between logic programs in terms of $\mathrm{G_3}$ logic.

Proposition 3.5 *Let P be a logic program and M a consistent set of literals. If $P \cup not\, \widetilde{M} \cup not\, not\, M$ has one unique $\mathrm{G_3}$ model then M is an answer set of P.*

Proof. Let $\Gamma = P \cup not\, \widetilde{M} \cup not\, not\, M$ and I its unique model. I must be complete, if not by Lemma 2.2 there would be at least two models I and I' for Γ. Now it is only left to verify that $\vdash_{\mathrm{G_3}} \bigwedge \Gamma \rightarrow a$ for all $a \in M$. Take any interpretation K. If $K(a) = 2$ it is immediate $K(\bigwedge \Gamma \rightarrow a) = 2$. If $K(a) = 1$ then, since $K \neq I$ and I is the unique model, $K(\bigwedge \Gamma) < 2$ and thus $K(\bigwedge \Gamma \rightarrow a) = 2$. If $K(a) = 0$ then $K(not\, not\, a) = 0$ and, since $not\, not\, a \in not\, not\, M \subseteq \Gamma$, we have $K(\bigwedge \Gamma) = 0$ and $K(\bigwedge \Gamma \rightarrow a) = 2$. In any case K models $\bigwedge \Gamma \rightarrow a$ concluding $\Gamma \vdash_{\mathrm{G_3}} a$. □

For the next proposition we need to define a relation between $\mathrm{G_3}$ interpretations and consistent sets of literals.

Definition 3.6 For a given interpretation I we define the set of literals $M_I = \{a \in \mathcal{L} \mid I(a) \neq 0\}$.

Note that, due to the nature of G_3, previous definition assigns to a consistent interpretation I one consistent set of literals M_I. Also, by construction, I is a model for both $not\ \widetilde{M_I}$ and $not\ not\ M_I$. The following proposition is useful to obtain information on the answer sets for a logic program under some particular circumstances.

Proposition 3.7 *Let P be a logic program.*

1. *If P has one unique G_3 model I then M_I is an answer set of P.*
2. *If P has no G_3 models then P is inconsistent and has no answer sets.*
3. *If I is an incomplete model of P then M_I is not an answer set of P.*

Proof. Let $\Gamma = P \cup not\ \widetilde{M_I} \cup not\ not\ M_I$.

(1) Note I is a model of Γ and, since it is the unique model for P, it will be the unique model of Γ. Proposition 3.5 proves M_I is an answer set of P.

(2) Take any interpretation K, then K must evaluate P to 0, otherwise the interpretation K' –as in Definition 2.1– would be a model of P, contradiction. So all interpretations make $K(P) = 0$ and then P is inconsistent.

(3) It is immediate since there is an atom $a \in M_I$ such that $I(a) = 1$, but $I(\bigwedge \Gamma) = 2$ so $I(\bigwedge \Gamma \rightarrow a) = 1$ and therefore $\bigwedge \Gamma \rightarrow a$ is not a tautology in G_3. In particular $\Gamma \not\vdash_{G_3} a$. □

4 Equivalence

Given two logic programs P_1 and P_2, a reasonable definition for equivalence would be that they are equivalent when they have the same answer sets. However, when developing a logic program, one would expect that replacing "equivalent" pieces of programs will lead globally to equivalent programs.

But this is not always true for simple equivalence as defined above. Take for example $P_1 = \{a \leftarrow not\ b\}$ and $P_2 = \{a\}$. These two programs are equivalent but replacing one rule by the other inside the larger program $P = \{a \leftarrow not\ b, \quad b\}$ will modify the resulting answer sets. The authors in (Lifschitz, Pearce, and Valverde 2001) offer a convenient definition of equivalence that satisfies the properties we are looking for.

Definition 4.1 (Lifschitz, Pearce, and Valverde 2001) Two programs P_1 and P_2 are said to be *strong equivalent* if, for every program P, $P_1 \cup P$ and $P_2 \cup P$ have the same answer sets.

They also show that G_3 characterizes this notion of strong equivalence for nested programs.

Theorem 4.2 (Lifschitz, Pearce, and Valverde 2001) *Given two nested programs P_1 and P_2, they are strongly equivalent if and only if $P_1 \equiv_{G_3} P_2$.*

Their proof, which is based in the HT logic, makes use of a characterization of answer sets in terms of *equilibrium models*. These equilibrium models are particular cases of HT models whose definition is closely related to the definition of answer sets for nested programs given in (Lifschitz, Tang, and Turner 1999). In particular the proof of Lemma 3 in (Lifschitz, Pearce, and Valverde 2001), which states this characterization, consists of a rephrase of the original definition for answer sets in terms of equilibrium models.

We will prove, as one would like to expect, that G_3 still characterizes the notion of strong equivalence for arbitrary logic programs with answer sets defined as in Definition 3.4. For one of the implications it will be assumed that $P_1 \not\equiv_{\mathrm{G}_3} P_2$ and then construct another program P such that $P_1 \cup P$ and $P_2 \cup P$ have different answer sets. The following definition will be helpful to construct such program P.

Definition 4.3 For a given interpretation I let $T(I)$ be the smallest program that satisfies:

- If $I(a) = 0$ then $not\, a \in T(I)$.
- If $I(a) = I(b) = 1$, and $a \neq b$, then $b \leftarrow a \in T(I)$.
- If $I(a) = 2$ then $a \in T(I)$.

Note that I if a model for $T(I)$, $not\, \widetilde{M_I}$ and $not\, not\, M_I$.

Lemma 4.4 *If $P_1 \not\equiv_{\mathrm{G}_3} P_2$ then there is a program P such that $P_1 \cup P$ and $P_2 \cup P$ have different answer sets.*

Proof. If I is an interpretation that models P_1 and not P_2 it will be called a witness interpretation. Since $P_1 \not\equiv_{\mathrm{G}_3} P_2$ by Proposition 2.3 we can suppose, without loss of generality, there is at least one witness interpretation. We will assume $\mathcal{L} = \mathcal{L}_{P_1 \cup P_2}$. The proof is separated in two cases.

(1) Suppose there is a witness interpretation I that is also complete (evaluates all atoms either to 0 or 2). Let $P = T(I)$ and $M = M_I$. Observe that I models $P_1 \cup P$ and it is its unique G_3 model since, in particular, I is the only interpretation that will model P. Then Proposition 3.7 states M is an answer set of $P_1 \cup P$. But $P_2 \cup P$ is inconsistent, since the only interpretation that models P does not model P_2, and thus cannot have answer sets.

(2) Suppose that all witness interpretations are incomplete. Let I be one of such interpretations and let $P = T(I)$. Since I is incomplete Proposition 3.7 states M_I is not an answer set of $P_1 \cup P$. Also observe I' –as in Definition 2.1– models P_2, otherwise I' would be a complete interpretation that models P_1 and not P_2, but all witness implementations were supposed to be incomplete.

Let $\Gamma = P_2 \cup P \cup not\, \widetilde{M_{I'}} \cup not\, not\, M_{I'}$. It will be shown I' is the unique model for Γ and, since $M_{I'} = M_I$ we have, by Proposition 3.5, that M_I is

an answer set for $P_2 \cup P$. Suppose there is another interpretation K that models Γ. Take any atom $a \in \mathcal{L}$.

If $I(a) = 2$ (or 0) then there is a clause a (or $not\, a$) in $P \subseteq \Gamma$ and, since K models Γ it must be the case $K(a) = 2$ (or 0). If $I(a) = 1$ then there is a clause $not\, not\, a \in not\, not\, M_I \subseteq \Gamma$, since K models Γ it must be the case $K(a) > 0$. We will show that, if $I(a) = I(b) = 1$ then $K(a) = K(b)$. If $I(a) = I(b) = 1$ then there are clauses $\{a \leftarrow b, b \leftarrow a\} \subseteq P \subseteq \Gamma$ and, since K models Γ, $K(a \equiv b) = 2$ or, equivalently, $K(a) = K(b)$.

Thus all atoms that evaluate to 1 in I must now evaluate to the same value in K, either 1 or 2. If $K(a) = 1$ then $I = K$, but I was not a model for Γ. So the only case left is when $K(a) = 2$, which implies $K = I'$ the unique model we found for Γ. □

Theorem 4.5 *Given two logic programs P_1 and P_2, they are strongly equivalent if and only if $P_1 \equiv_{\mathrm{G}_3} P_2$.*

Proof. One implication of the theorem is a direct from the characterization of answer sets. Namely if $P_1 \equiv_{\mathrm{G}_3} P_2$ then if we have $P_1 \cup P \cup not\, \widetilde{M} \cup not\, not\, M \vdash_{\mathrm{G}_3} M$ it is possible to replace the program P_1 with P_2 and obtain a proof for $P_2 \cup P \cup not\, \widetilde{M} \cup not\, not\, M \vdash_{\mathrm{G}_3} M$. For the other implication suppose $P_1 \not\equiv_{\mathrm{G}_3} P_2$, Lemma 4.4 says they are not strong equivalent. □

5 Conclusions

We show strong relations between logic programming and intermediate logics. We study, in particular, properties of the answer set semantics in terms of the G_3 logic. A generalized definition of answer sets for logic programs, which allows even the use of embedded implication in rules, is proposed.

Some benefits of having this extended definition are discussed. It is known that rules with nested expressions can be reduced to disjunctive programs. An interesting question, left open for future research, is to determine whether this broader syntax gives power to express a wider class of problems or if these logic programs can also be reduced to disjunctive ones.

The main contribution of this paper is to prove that the logic G_3 can be also used to characterize the notion of strong equivalence between logic programs under this generalized framework. We observe how the G_3 logic seems to be a new interesting approach to understand answer set problems.

An interesting line of research, for future work, is to search for further relations and applications of intermediate logics in answer sets. One could try to exploit properties from logics like G_3 to define new program transformations that could improve computability of answer sets. Results like this would allow to develop more efficient and comprehensive software systems to do logic programming.

Bibliography

Gelfond, M. and V. Lifschitz (1988). The Stable Model Semantics for Logic Programming. In R. Kowalski and K. Bowen (Eds.), *5th Conference on Logic Programming*, pp. 1070–1080. MIT Press.

Kowalski, R. (2001). Is logic really dead or just sleeping. In *Proceedings of the 17th ICLP*, pp. 2–3.

Lifschitz, V. (1996). Foundations of Logic Programming. *Principles of Knowledge Representation*, 69–137.

Lifschitz, V., D. Pearce, and A. Valverde (2001). Strongly Equivalent Logic Programs. *ACM Transactions on Computational Logic 2*, 526–541.

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

Osorio, M., J. A. Navarro, and J. Arrazola (2001). Equivalence in Answer Set Programming. In A. Pettorossi (Ed.), *LOPSTR'01: Logic-Based Program Synthesis and Transformation*, Paphos, Cyprus, pp. 18–28.

Osorio, M., J. A. Navarro, and J. Arrazola (2002a). Applications of intermediate logics in ASP. Submitted to TCS.

Osorio, M., J. A. Navarro, and J. Arrazola (2002b). A logical approach for A-Prolog. To appear in Proceedings of WOLLIC.

Pearce, D. (1999). Stable Inference as Intuitionistic Validity. *Logic Programming 38*, 79–91.