A principled semantics for logic program updates[*]

F. Banti and **J. J. Alferes**
CENTRIA
Univ. Nova de Lisboa, Portugal
{banti|jja}@di.fct.unl.pt

A. Brogi
Department of Computer Science
University of Pisa, Italy
brogi@di.unipi.it

Abstract

Over the last years various semantics have been proposed for dealing with updates in the setting of logic programs. The availability of different semantics naturally raises the question of which are the most adequate to model updates. A systematic approach to face this question is to identify general principles against which such semantics should be tested. In this paper we introduce a new such principle – the *refined extension principle* – which is satisfied by the stable model semantics for (static) logic programs. It turns out that none of the existing semantics for logic program updates, eventhough based on stable models, satisfy this principle. For this reason, we define a refinement of the semantics of Dynamic Logic Programs that is proved to satisfy the principle.

1 Introduction and motivations

Most of the work done in the field of logic programming for representing knowledge that evolves over time, mainly deals with changes in the extensional part of knowledge bases (factual events or observations). This is what happens, e.g., with the event calculus [Kowalski and Sergot, 1986], with logic programming forms of the situation calculus [Levesque *et al.*, 1998; McCarthy and Hayes, 1969] , or with logic programming representations of action languages [Gelfond and Lifschitz, 1993] . In all these works, the problems of updating the intensional part of the knowledge base (rules or descriptions of actions) remained basically unexplored. However, in the last years various attempts have been made for dealing with such updates in the setting of logic programs, and different semantics have been proposed [Alferes *et al.*, 1998; Buccafurri *et al.*, 1999; Eiter *et al.*, 2002; Leite and Pereira, 1997; 1998; Leite, 2003; Sakama and Inoue, 1999; Zhang and Foo, 1998].

[*]This work was partially supported by project FLUX (POSI/40958/SRI/2001/) and by the Information Society Technologies programme of the European Commission under the IST-2001-32530 project SOCS. Special thanks are due to João Leite and Reinhard Kahle for helpful discussions.

In logic program updates, a meaning is given to sequences of sets of rules, where later sets in the sequence are read as updates to the previous ones. It is hence natural that most of these semantics [Alferes *et al.*, 1998; Buccafurri *et al.*, 1999; Eiter *et al.*, 2002; Leite and Pereira, 1997; 1998; Leite, 2003; Zhang and Foo, 1998] are based on causal rejection, i.e., on the rejection of a prior rule if there is a more recent one that conflicts with it.

While the existing semantics based on causal rejection coincide on a large class of program updates, there are situations in which they differ one another, and there is no common agreement on which is the "right" semantics to take in such cases. A systematic approach to answer the above question is described in [Eiter *et al.*, 2002; Leite, 2003], where some principles to analyze and compare these semantics are set forth. It is our stance that, besides the principles described in [Eiter *et al.*, 2002; Leite, 2003], another important principle is needed to test the adequacy of semantics of logic program updates in some important situations. In this paper we propose a new principle, that we call the *refined extension principle* and that can be used to compare different semantics for updates based on the stable model semantics — as it is the case for all the above mentioned ones.

Informally, the semantics based on stable models are obtained by taking the least Herbrand model of the definite program obtained by adding some assumptions (default negations) to the initial program. Intuitively speaking, the refined extension principle states that the addition of rules that do not change that least model should not lead to obtaining more (stable) models.

Before providing an example, let us try to explore a bit further the intuition behind this principle. It is now well known that the stable model semantics, and its corresponding implementation systems (e.g., smodels [SMODELS, 2000] and DLV [DLV, 2000]), can be fruitfully used to compute solutions for NP-complete problems. The idea is to write a logic program whose stable models are exactly the solutions of the problem [Marek and Truszczyński, 1987; Niemelä, 1999] . For doing so, a widely used technique consists of first writing a set of rules that would generate a larger amount of stable models, and then add extra rules to impose integrity constraints over this larger set of models. In this perspective, it is important to guarantee that the rules used for imposing constraints have no other effects apart from pruning models, i.e., they do not introduce new models.

Remarkably, while in (static) stable models semantics the introduction of such rules does not add new models, this is not the case in general for any of the existing semantics for updates. For instance, all the semantics fail to satisfy this principle in examples where the undesired behaviour is determined by a tautological update. We claim that this is clearly counterintuitive, since one would expect tautologies to be completely irrelevant for the semantics of a logic program (update). Let us now show an example to illustrate the problem.

Example 1.1 *Consider the program P_1 describing some knowledge about the sky. In each moment it is either day time or night time, we can see the stars whenever it is night time and there are no clouds, and currently it is not possible to see the stars.*

$$\begin{array}{lrcl} P_1 : & Day & \leftarrow & not\ Night. \\ & Night & \leftarrow & not\ Day. \\ & See\_stars & \leftarrow & Night,\ not\ Cloudy. \\ & not\ See\_stars. & & \end{array}$$

The only stable model of this program is $\{Day\}$. Suppose now the program is updated with the following tautology:

$$P_2 : \quad See\_stars \leftarrow See\_stars.$$

It is worth noting that this tautological update introduces a new stable model, namely $\{Night, See\_stars\}$, in all the semantics based on causal rejection [Alferes et al.*, 1998; Buccafurri* et al.*, 1999; Eiter* et al.*, 2002; Leite and Pereira, 1997; 1998; Leite, 2003]. We argue that this behaviour is counterintuitive as the addition of the tautology in P_2 should not add new models.*

In this paper we will formally introduce the refined extension principle in order to avoid counterintuive modelizations, such as the one illustrated in the previous example. Since none of the existing semantics for logic program updates will turn out to fully comply with the refined extension principle, we shall present here a new compliant semantics for logic program updates. Such semantics will be obtained by refining the semantics of Dynamic Logic Programming [Alferes *et al.*, 2000]. The choice of this extant semantics as the basis for our definition is justified by the fact that, among all the existing semantics, it is the one that satisfies the refined extension principle on the largest class of programs.

The rest of the paper is organized as follows. Section 2 recalls some preliminary notions and establishes the notation. Section 3 is devoted to motivate and present the refined extension principle, while in Section 4 a refined semantics for logic program updates that satisfies the principle is presented. Section 5 is devoted to compare the new semantics with other existing semantics, and to analyze these with respect to the refined extension principle. Finally, some concluding remarks are drawn.

2 Background: Concepts and notation

In the paper we extensively use the concept of generalized logic programs [Lifschitz and Woo, 1992]. Generalized logic programs are sets of rules that admit default negated literals both in the heads and in the bodies. A generalization of the stable models semantics for normal logic programs [Gelfond and Lifschitz, 1988] to the class of generalized programs was defined by Lifschitz and Woo [Lifschitz and Woo, 1992]. Here we present this semantics differently from [Lifschitz and Woo, 1992], the equivalence of both definition being proven in [Alferes *et al.*, 2000].

In the following, given a generalized program P, we will use the notation $least(P)$ to denote the least Herbrand model of the definite program obtained by interpreting all default negated literals as new atoms. An interpretation I over a language $\mathcal{L}$ is a set of literals such that for each atom A in $\mathcal{L}$, exactly one of the two literals A, $not\ A$ belongs to I. For simplicity we sometimes omit the negative literals from the extensive enumeration of the members of I. Finally, given an interpretation I, we use the expression I^- for the set of negative literals in I. With an abuse of notation, we will also use I^- to refer to the set of facts $not\ A \leftarrow$ such that $notA$ belongs to I. Given a generalized program P and an interpretation M, we say M is a stable model of P iff $M = least(P \cup M^-)$.

As mentioned in the Introduction, we are interested in studying updates in logic programs. For the language to express such updates, we follow [Alferes *et al.*, 1998; Leite and Pereira, 1998; Leite, 2003] where updates are expressed by sequences $P_1, \ldots, P_n$ of generalized logic programs. For notational convenience, we often refer to one such sequence by $P_1 \oplus \ldots \oplus P_n$, or simply by $\oplus P_i$. We also use the notation P for $\bigcup P_i$, E for $\bigcup E_i$, and $\oplus P_i^{E_i}$ for a sequence $\bigoplus (P_i \cup E_i)$, where all the P_i and E_i are sets of rules. Finally, by $P_\emptyset$ we simply mean an empty set of rules.

Two literals L_1 and L_2 in the language of a given program are called *conjugate* iff L_1 is some atom A and L_2 is $not\ A$, or viceversa. We say that two rules τ and η are *conflicting*, and denote it by $\tau \bowtie \eta$, iff their heads are conjugate.

A sequence $P_1 \oplus \ldots \oplus P_n$ is viewed as starting from the initial program P_1, then updating it with the rules in P_2, ..., then updating it with the rule in P_n. In all the above mentioned semantics that are based on causal rejection of rules, the meaning of such a sequence can be obtained as follows. Given an interpretation I:

- First choose from the union of all the rules in the sequence $\oplus P_i$ the subset $Residue(\oplus P_i, I)(P)$ of those that are not rejected by any update,
- Then add to the resulting logic program a set of default assumptions $Assumptions(\oplus P_i, I)$, similarly to what is done in the stable models semantics of generalized programs, where assumptions of the form $not\ A$ are added for every $A \notin I$.

An interpretation M is a stable model of a sequence

$\oplus P_i$ iff

$$M = \Gamma(\oplus P_i, M)$$

where

$$\Gamma(\oplus P_i, M) = least \left(\begin{array}{r} Residue(\oplus P_i, M)(P) \\ \cup \ Assumptions(\oplus P_i, M) \end{array} \right).$$

From [Leite, 2003] it is easy to see that all the existing semantics for updates based on causal rejection are parameterizable using different definitions of $Residue$[1] and $Assumptions$. The dynamic logic programming semantics [Alferes *et al.*, 1998] is obtained by putting:

$$\begin{array}{rcl} Assumptions(\oplus P_i, I) & = & Default(\oplus P_i, I) \\ Residue(\oplus P_i, I)(X) & = & X \setminus Rej(\oplus P_i, I) \end{array}$$

where $Default(\oplus P_i, I)$ is the set:

$$\{not\ A \leftarrow \mid\ \nexists A \leftarrow \overline{B} \in P_j \ and \ I \models \overline{B}\}$$

and $Rej(\oplus P_i, I)$ is the set

$$\{\tau \in P_i \mid \exists\ \eta\ \in P_j,\ i < j :\ \tau \bowtie \eta \ and \ I \models body(\eta)\}.$$

3 Refined extensions

We are interested in finding conditions guaranteeing that the addition of a set of rules E to a logic program does not generate new stable models. The concept of refined extension and the results proven in this paper are a step in this direction. We start by defining these concepts on single (static) programs, and then proceed for sequences of programs.

3.1 Refined extensions of a single program

We first define the notion of *syntactic extension* of a program. Namely, we say that $P \cup E$ is a syntactic extension of P iff the rules in E have no effect on the least Herbrand model of P. A rule is deemed *uneffective* in case its addition yields a syntactic extension of the original program.

Definition 3.1 *Let P be a definite logic program, and let E be a set of rules. We say $P \cup E$ is a* syntactic extension *of P iff P and $P \cup E$ have the same least Herbrand model.*

For normal and generalized programs, in this paper we only focus on those semantics whose models can be computed as the least Herbrand model of the definite logic program $P \cup Assumptions(P, M)$, where $Assumptions(P, M)$ is simply a set of default negated literals (viewed as new atoms), whose definition depends on the semantics in use. Note that all of the stable models [Gelfond and Lifschitz, 1988] , the well-founded [Gelder *et al.*, 1991] and the weakly perfect model semantics [Przymusinska and Przymusinski, 1988] can be defined in this way. For example, the stable model semantics of normal and generalized programs can be obtained by putting

$$Assumptions(P, M) = \{notA \mid A \notin M\}.$$

[1] $Residue(\oplus P_i, I)$ is viewed as a function from a set of rules X to the set of rules of X that are not rejected.

In the sequel, by $Sem(P)$ we denote the set of all models of a given program P, obtained by a semantics Sem that can be defined as above.

Consider now a generalized program P, and a set of rules E. A model M of $P \cup E$ in the semantic Sem is computed as the least Herbrand model of the definite logic program obtained by adding the set of default assumptions to $P \cup E$. It is then possible to apply the concept of syntactical extension to verify whether the addition of the rules of E does not influence the computation of M. When this happens for all Sem models of the program $P \cup E$, we call such program a refined extension of the original program P.

Definition 3.2 *Let P be a generalized program, M an interpretation of P, E a set of rules, and Sem a semantics for generalized logic programs. We say that $P \cup E$ is an* extension *of P with respect to M iff*

$$P \cup Assumptions(P \cup E, M) \cup E$$

is a syntactic extension of

$$P \cup Assumptions(P \cup E, M).$$

We say that $P \cup E$ is a refined extension *of P iff $P \cup E$ is an extension of P with respect to all the models in $Sem(P)$.*

Example 3.1 *Consider the programs P_1 and P_2 from Example 1.1. It is clear that no matter what are the negative assumption S added given any model M, the least model of $P_1 \cup S$ will be the same as the least model of $P_1 \cup S \cup P_2$. Thus, $P_1 \cup P_2$ is a refined extension of P_1.*

We can now formulate the refined extension principle for the case of generalized logic programs by stating that a refined extension of a program P should not have more models than P.

Principle 1 (Refined extension — static case) *A semantic Sem for generalized logic programs complies with the refined extension principle iff for any program P and set of rules E, if $P \cup E$ is a refined extension of P then*

$$Sem(P \cup E) \subseteq Sem(P).$$

As one may expect, the principle properly deals with the case of adding tautologies, i.e., for any semantics Sem that satisfies the principle, the addition of tautologies (i.e., rules of the form $L \leftarrow Body$ with $L \in Body$) does not generate new models.

Proposition 3.1 *Let Sem be a semantics for generalized programs, P a generalized program, and τ a tautology. If Sem satisfies the refined extension principle then*

$$Sem(P \cup \{\tau\}) \subseteq Sem(P).$$

Most importantly, the stable semantics does satisfy the refined extension principle, as stated in the following proposition.

Proposition 3.2 *The stable model semantics satisfies the refined extension principle.*

As an immediate consequence of these two propositions, we get that the addition of tautologies to a generalized program does not introduce new stable models.

3.2 Refined extensions of program updates

It would be desirable to have some form of refined extension principle also for the semantics of updates, in order to guarantee that updates of integrity constraints do not generate new models. Unfortunately, it turns out that none of the existing semantics for logic program updates fully satisfies such a principle. Intuitively speaking, the reason of the difficulty is that the rules in a program sequence are both used to derive conclusions and to reject other rules. To formally state the problem, let us first lift the definition of (syntactic) extension to the case of program updates.

Definition 3.3 *Let $\oplus P_i$ be a sequence of generalized logic programs, Sem a semantics for updates and M an interpretation of $\oplus P_i$.*

We say that $\oplus P_i^{E_i}$ is an extension *of $\oplus P_i$ with respect to M iff*

$$\begin{array}{ll} Residue(\oplus P_i^{E_i}, M)(P) & \bigcup \\ Assumptions(\oplus P_i^{E_i}, M) & \bigcup \\ Residue(\oplus P_i^{E_i}, M)(E) & \end{array}$$

is a syntactical extension of

$$Residue(\oplus P_i^{E_i}, M)(P) \cup \; Assumptions(\oplus P_i^{E_i}, M).$$

We say that $\oplus P_i^{E_i}$ is a refined extension *of $\oplus P_i$ iff $\oplus P_i^{E_i}$ is an extension of $\oplus P_i$ with respect to all models in $Sem(\oplus P_i^{E_i})$.*

Note that the above definition is the straightforward lifting of Definition 3.2 to the case of updates. Roughly speaking, we simply replaced P and E with $Residue(\oplus P_i, M)(P)$ and $Residue(\oplus P_i, M)(E)$, respectively.

Example 3.2 *Consider again programs P_1 and P_2 from Example 1.1, the sequence $P_1 \oplus P_\emptyset$, $E_1 = \emptyset$, and $E_2 = P_2$. The residue of E, independently of the model M in consideration, can only be either the empty set of rules or P_2. In both cases, the addition of the residue does not change the least model. Thus $P_1 \oplus P_2$ is a refined extension of $P_1 \oplus P_\emptyset$.*

The refined extension principle is then formulated as follows.

Principle 2 (Refined extension principle) *A semantics Sem complies with the refined extension principle iff for any sequences of programs $\oplus P_i$ and $\oplus P_i^{E_i}$, if $\oplus P_i^{E_i}$ is a refined extension of $\oplus P_i$ then*

$$Sem(\oplus P_i^{E_i}) \subseteq Sem(\oplus P_i).$$

As for the case of generalized programs, for any semantics *Sem* that satisfies the principle, the addition of tautologies does not generate new models. This is stated by the following proposition that lifts Proposition 3.1 to the case of program updates.

Proposition 3.3 *Let Sem be a semantics for logic program updates, $\oplus P_i$ a sequence of generalized programs, and $\oplus E_i$ be a sequence of sets of tautologies. If Sem satisfies the refined extension principle then*

$$Sem(\oplus P_i^{E_i}) \subseteq Sem(\oplus P_i).$$

Unfortunately, as we already pointed out in the Introduction, the existing semantics for logic program updates based on causal rejection do not satisfy the refined extension principle. Indeed tautological updates may generate new models, as shown in Example 1.1. It is worth observing that tautological updates are not the only cases in which these semantics fail to satisfy the principle, as illustrated by the following example of an update containing a rule whose head is *self-dependent*[2].

Example 3.3 *Consider again program P_1 of Example 1.1, and suppose that P_1 is now updated with:*

$$\begin{array}{ll} P_2: & See\_stars \leftarrow See\_Venus. \\ & See\_Venus \leftarrow See\_stars. \end{array}$$

While P_1 has only one stable model (viz., $\{Day\}$). In all the existing semantics for updates based on causal rejection the update P_2 adds a second stable model:

$$\{See\_stars, See\_Venus, Night\}$$

Intuitively speaking, this new model arises since the update P_2 causally rejects the rule of P_1 which stated that it was not possible to see the stars.

The presence of the new model contrasts with the refined extension principle. Indeed, if we consider the empty update $P_\emptyset$, then the program $P_1 \oplus P_\emptyset$ has only one stable model (viz., $\{Day\}$). Since $P_1 \oplus P_2$ is a refined extension of $P_1 \oplus P_\emptyset$, then according to the principle all models of $P_1 \oplus P_2$ must also be models of $P_1 \oplus P_\emptyset$.

If we consider infinite logic programs, there is also another class of examples where all the existing semantics based on causal rejection fail to satisfy the principle. This second class of counterexamples consists of sequences of updates containing an infinite number of conflicting rules whose bodies are satisfied, i.e., an infinite number of potential contradictions, such that each potential contradiction is removed by a later update.

Example 3.4 *Consider a language consisting of the constant 0 and of the unary function successor. The set of terms of such a language has an obvious bijection to the set of natural numbers. In this context, given a term n, we will use the expression $n+1$ for $successor(n)$. Let A be an unary predicate.*

Consider the following sequence of programs:

$$\begin{array}{ll} P_1: & A(X). \\ & not\ A(X). \\ P_2: & A(X) \leftarrow \; A(X+1). \end{array}$$

[2]For the definition of self-dependent literals in a logic program, see [Apt and Bol, 1994].

In order to compute the stable model semantics, we have to consider the ground versions of the two programs:

$$\begin{array}{ll} P_1^{ground}: & A(n).\ \ \forall\, n \in \mathbb{N} \\ & not\ A(n).\ \ \forall\, n \in \mathbb{N} \\ P_2^{ground}: & A(n) \leftarrow\ A(n+1).\ \ \forall\, n \in \mathbb{N} \end{array}$$

The program $P_1^{ground} \oplus P_\emptyset$ is clearly contradictory, and hence has no stable models. However, in all of the above mentioned semantics based on causal rejection, $P_1^{ground} \oplus P_2^{ground}$ admits the whole Herbrand base as a stable model. The program $P_1^{ground} \oplus P_2^{ground}$ is a refined extension of $P_1^{ground} \oplus P_\emptyset$, and so, according to the refined extension principle, it should have no stable models.

4 Refined semantics for logic program updates

We are now going to define a new semantics for logic program updates that satisfies the refined extension principle.

Before defining the new semantics, let us analyze the reason why dynamic logic programming fails to satisfy the refined extension principle in Example 1.1. In this example the extra (counterintuitive) stable model $\{Night, See\_stars\}$ is obtained because the tautology

$$See\_stars \leftarrow See\_stars.$$

in P_2 has a true body in that model, and hence it rejects the fact

$$not\ See\_stars.$$

of P_1. After rejecting this fact, it is possible to consistently conclude See_stars, and thus verify the fixpoint condition, via the rule

$$See\_stars \leftarrow Night, not\ Cloudy.$$

of P_1.

Here lies the matrix of the undesired behaviour of dynamic logic programming: One of two conflicting rules (of P_1) is used to support a later rule (of P_2) that actually removes that same conflict. Informally, rules that should be irrelevant may become relevant because they can be used by one of the conflicting rules to defeat the other.

A simple way to inhibit this behaviour is to let conflicting rules in the same state inhibit each over. This can be obtained by slightly modifying the notion of rejected rules of dynamic logic programming, by also allowing rules to reject other rules in the same state. Since in dynamic logic programming rejected rules can reject rules, two rules in the same state can reject each other, thus avoiding the above described behaviour.

Definition 4.1 *Let $\oplus P_i$ be a sequence of programs and M an interpretation. The set of rejected rules $Rej^S(\oplus P_i, M)$ is defined as:*

$$\{\tau \in P_i \mid \exists\ \eta\ \in P_j,\ i \leq j:\ \tau \bowtie \eta \text{ and } M \models body(\eta)\}.$$

Then M is a stable model of $\oplus P_i$ iff:

$$M = \Gamma^S(\oplus P_i, M)$$

where $\Gamma^S(\oplus P_i, M)$ is

$$least\left(P \setminus Rej^S(\oplus P_i, M) \cup Default(\oplus P_i, M)\right).$$

At first sight this modification seems to allow the existence of models in cases where a contradiction is expected (for example in a sequence where the last program contains facts for both A and *not* A): If rules in the same state can reject each other then the contradiction is removed, and the program may have again undesired models. However, the converse is actually true (cf. theorem 5.1), and the stable models in the refined semantics are always stable models of the semantics of dynamic logic programming, i.e., by allowing rejection of rules in the same state no extra models are introduced. To see why this is so, consider a sequence $\oplus P_i$ with two conflicting rules (with heads A and $notA$) in one of its programs. Take an interpretation M where the bodies of those rules are both true (as nothing special happens if a rule with false body is rejected) and check if M is a stable model according to the new definition. By definition 4.1, these two rules reject each other, and reject all other rules with head A or *not* A in that or in any previous state. Moreover *not* A is also rejected as a default assumption, i.e., does not belong to $Default(\oplus P_i, M)$. This means that all the information about A is deleted. Since M is two valued, the only possibility for M to be a stable model is that there exists a rule τ in some later update whose head is either A or *not* A, and whose body is true in M. This means that a potential inconsistency can only be removed by some later update.

Finally, as this was the very motivation for introducing the refined semantics, it is worth observing that the refined semantics does satisfy the refined extension principle, as stated by the following theorem.

Theorem 4.1 *The refined semantics satisfies the refined extension principle.*

By Proposition 3.3, it immediately follows from this theorem that the addition of tautologies never adds models in this semantics. Moreover the refined semantics preserves all the desirable properties of the previous semantics for logic programs updates [Eiter *et al.*, 2002; Leite, 2003].

To give an insight view of the behaviour of the refined semantics we show how the counterintuitive results of dynamic logic programming in the example 1.1 are corrected.

Example 4.1 *Consider again the sequence $P_1 \oplus P_2$ of example 1.1*

$$\begin{array}{ll} P_1: & Day \leftarrow not\ Night. \\ & Night \leftarrow not\ Day. \\ & See\_stars \leftarrow Night,\ not\ Cloudy. \\ & not\ See\_stars. \\ P_2: & See\_stars \leftarrow See\_stars. \end{array}$$

This sequence has one single stable model, $M = \{Day\}$. Thus the conclusions of the semantics match with the intuition that it is day and it is not possible to see the stars. We show that M is a stable model according to the new definition. First of all we compute the sets $Rej^S(\oplus P_i, M)$ and $Default(\oplus P_i, M)$:

$$\begin{aligned} Rej^S(\oplus P_i, M) &= \emptyset \\ Default(\oplus P_i, M) &= \{not\, Night, not\, See\_Stars, \\ &\quad not\, Cloudy\} \end{aligned}$$

Then we check whether M is a stable model according to definition 4.1 . Indeed:

$$\begin{aligned} &\Gamma^S(\oplus P_i, M) \\ =\ & least((P_1 \cup P_2) \setminus Rej^S(\oplus P_i, M) \cup Default(\oplus P_i, M)) \\ =\ & \{Day, not\, Night, not\, See\_Stars, not\, Cloudy\} \\ =\ & M. \end{aligned}$$

We already observed that in dynamic logic programming, $P_1 \oplus P_2$ has a second stable model

$$N = \{Night, See\_stars\}.$$

The presence of this model violates the refined extension principle. We show that N is not a stable model in the refined semantics. As above we compute the sets:

$$\begin{aligned} Rej^S(\oplus P_i, N) = \{ & not\ See\_stars. \ , \\ & See\_stars \leftarrow Night. \ , \\ & not\ Cloudy. \ \} \end{aligned}$$

and

$$Default(\oplus P_i, N) = \{not\ Day,\ not\ Cloudy\}.$$

Hence:

$$\begin{aligned} &\Gamma^S(\oplus P_i, N) \\ =\ & least((P_1 \cup P_2) \setminus Rej^S(\oplus P_i, N) \cup Default(\oplus P_i, N)) \\ =\ & \{Night, not\, Day, not\, not\, Cloudy\} \\ \neq\ & N. \end{aligned}$$

So, according to definition 4.1 , N is not a stable model.

5 Comparisons

Unlike the semantics presented here, as shown by the examples above, none of the semantics based on causal rejection respects the refined extension principle and, consequently, is not immune to the addition of tautologies.

It is clear from the definitions that the refined semantics coincides with dynamic logic programming [Alferes *et al.*, 1998] for sequences of programs with no conflicting rules in a same program. This means that the dynamic logic programming semantics obeys the refined extension principle for that class of sequences of programs, and it would have no problems if one restricts its application to that class. However such limitation would reduce the freedom of the programmer, particulary in the possibility of using conflicting rules to represent integrity constraints. Since we have introduced the refined extension principle as a property for programming with integrity constraints, it would be weird to limit the use of integrity constraints in order to satisfy such principle. Another limitation would result from the fact that updates are also a tool to remove inconsistency in programs by rejecting conflicting rules. This feature would be completely lost in tha case.

As for the other semantics, it is not even the case that the principles is satisfied by sequences in that restricted class. Update sequences [Eiter *et al.*, 2002] and inheritance programs [Buccafurri *et al.*, 1999] also fail to be immune to tautologies when no conflicting rules occur in the same program.

Example 5.1 *Consider the sequence of programs (taken from [Leite, 2003]):*

$$\begin{aligned} P_1: &\quad Day. \\ P_2: &\quad not\ Day. \\ P_3: &\quad Day \leftarrow\ Day. \end{aligned}$$

stating that initially it is day time, then it is no longer day time, and finally (tautologically) stating that whenever it is day time, it is day time. While the semantics of justified updates [Leite and Pereira, 1997; 1998] and the dynamic logic programming semantics of [Alferes et al.*, 1998; Leite, 2003] select $\{Day\}$ as the only model, updates sequences [Eiter* et al.*, 2002] and inheritance programs [Buccafurri* et al.*, 1999] associate two models: $\{Day\}$ and $\{not\ Day\}$ with such program sequence*[3]*.*

While the semantics of justified updates [Leite and Pereira, 1997; 1998] works properly for the above example, there are classes of program updates for which it does not satisfy the refined extension principle. To illusrate that, we show here a modified version of the example used in [Leite, 2003] for the same purpose.

Example 5.2 *Consider the sequence of programs*

$$\begin{aligned} P_1: &\quad Day. \\ P_2: &\quad not\ Day \leftarrow\ not\ Day. \end{aligned}$$

According to the semantics of justified updates, this program has two stable models, $M = \{Day\}$ and $S = \{not\ Day\}$. If justified updates would follow the refined extension principle, S should not be a stable model. In fact S is not a stable model of $P_1 \oplus P_\emptyset$ then, according to Proposition 3.3 , it can not be a stable model of $P_1 \oplus P_2$ as well.

Finally, observe that the refined semantics is more credulous than all the other semantics, in the sense that the set of its stable models is always a subset of the set of stable models obtained with any of the others. Comparing first with dynamic logic programming:

Theorem 5.1 *Let $\oplus P_i$ be a sequence of programs, and M an interpretation. If M is a stable model in the refined semantics then it is also a stable model according to dynamic logic programming.*

[3]Notice that, strictly speaking, the semantics [Buccafurri *et al.*, 1999; Eiter *et al.*, 2002] are actually defined for extended logic programs, with explicit negation, rather than for generalized programs. However, the example can be easily adapted to extended logic programs.

This result generalizes to all the other semantics since dynamic logic programming is the most credulous of the existing semantics. Indeed, each stable model in dynamic logic programming is also a stable model in the justified update semantics [Leite, 2003]. Inheritance programs are defined for disjunctive logic programs, but if we restrict to the non disjunctive case, this semantics coincides with the update sequences semantics of [Eiter *et al.*, 2002], and each stable model in dynamic logic programming is also a stable model in this one [Leite, 2003]. The analysis of the semantics for updates that are not based on causal rejection is beyond the scope of this paper. An analysis of such semantics can be found in [Eiter *et al.*, 2002; Leite, 2003].

6 Concluding remarks

We have introduced the refined extension principle as a property of the stable model semantics of generalized programs that is helpful for programming with integrity constraints. We have extended the principle to the semantics of logic programs with updates to assess the adequacy of the existing semantics for modelling real problems. We have shown by examples that none of the existing semantics satisfies the principle. Among the existing ones, the semantics of dynamic logic programming satisfies the principle for the wider class of sequences of programs, viz., for sequences without conflicting rules in the same program. Then we have introduced a new semantics that completely satisfies the principle and briefly compared it with the existing ones. It turns out that the new semantics is the one that admits the smallest set of stable models and, moreover, it coincides with dynamic logic programming semantics for the class of programs without conflicting rules in the same program.

Future lines of research include extending this study to deal with multi-dimensional updates. Such an extension for the semantics of dynamic logic programming is already defined in [Leite, 2003]. Multi-dimensional updates can be viewed as a tool for naturally encoding knowledge bases that incorporate information from different sources. In this scenario, the presence of contradictory information is a common fact. Preliminary investigations show that, whenever there is no hierarchy among conflicting information, the existing semantics exhibits counterintuitive behaviours, This problem raises the question of a proper formulation of the refined extension principle for the multi-dimensional case, together with the definition of a semantics that satisfies the principle.

Another line of research is the definition of a well-founded based semantics for logic programs updates. Preliminary results suggest the possibility to combine the concepts of dynamic logic programming and the new refined stable model semantics presented here, for defining the well-founded semantics via an alternating fixpoint operator.

Another important line for future work is the implementation of the refined semantics using a transformational approach. On top of this theoretical work, we want to establish new techniques for solving real problems, like software specification, legal reasoning, multi-agent architecture and updates of web-pages.

References

[Alferes *et al.*, 1998] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic logic programming. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, pages 98–111, San Francisco, 1998. Morgan Kaufmann Publishers.

[Alferes *et al.*, 2000] J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, 45(1–3):43–70, September/October 2000.

[Apt and Bol, 1994] K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *The Journal of Logic Programming*, 19 & 20:9–72, May 1994.

[Buccafurri *et al.*, 1999] F. Buccafurri, W. Faber, and N. Leone. Disjunctive logic programs with inheritance. In D. De Schreye, editor, *Proceedings of the 1999 International Conference on Logic Programming (ICLP-99)*, pages 79–93, Cambridge, November 1999. MIT Press.

[DLV, 2000] DLV. The DLV project - a disjunctive datalog system (and more), 2000. Available at `http://www.dbai.tuwien.ac.at/proj/dlv/`.

[Eiter *et al.*, 2002] T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On properties of semantics based on causal rejection. *Theory and Practice of Logic Programming*, 2:711–767, November 2002.

[Gelder *et al.*, 1991] A. Van Gelder, K. A. Ross, and J. S. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, 1991.

[Gelfond and Lifschitz, 1988] M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1070–1080. MIT Press, 1988.

[Gelfond and Lifschitz, 1993] M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.

[Kowalski and Sergot, 1986] R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.

[Leite and Pereira, 1997] J. A. Leite and L. M. Pereira. Generalizing updates: from models to programs. In *LPKR'97: ILPS'97 workshop on Logic Programming and Knowledge Representation*, 1997.

[Leite and Pereira, 1998] J. A. Leite and L. M. Pereira. Iterated logic program updates. In J. Jaffar, editor, *Proceedings of the 1998 Joint International Conference and Symposium on Logic Programming (JICSLP-98)*, pages 265–278, Cambridge, 1998. MIT Press.

[Leite, 2003] J. A. Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers in Artificial Intelligence and Applications.* IOS Press, December 2003.

[Levesque *et al.*, 1998] Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science*, 03, 1998.

[Lifschitz and Woo, 1992] V. Lifschitz and T. Woo. Answer sets in general non-monotonic reasoning (preliminary report). In B. Nebel, C. Rich, and W. Swartout, editors, *Proceedings of the 3th International Conference on Principles of Knowledge Representation and Reasoning (KR-92).* Morgan-Kaufmann, 1992.

[Marek and Truszczyński, 1987] Victor Marek and Miroslaw Truszczyński. *Foundations of Logic Programming.* Springer–Verlag, Berlin, 2 edition, 1987.

[McCarthy and Hayes, 1969] J. McCarthy and P. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Michie, editors, *Machine Intelligence 4*. Edinburgh University Press, 1969.

[Niemelä, 1999] Ilkka Niemelä. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence*, 1999.

[Przymusinska and Przymusinski, 1988] H. Przymusinska and T. Przymusinski. Weakly perfect model semantics. In R. Kowalski and K. A. Bowen, editors, *5th International Conference on Logic Programming*, pages 1106–1122. MIT Press, 1988.

[Sakama and Inoue, 1999] C. Sakama and K. Inoue. Updating extended logic programs through abduction. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Proceedings of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-99)*, volume 1730 of *LNAI*, pages 147–161, Berlin, 1999. Springer.

[SMODELS, 2000] SMODELS. The SMODELS system, 2000. Available at `http://www.tcs.hut.fi/Software/smodels/`.

[Zhang and Foo, 1998] Y. Zhang and N. Y. Foo. Updating logic programs. In Henri Prade, editor, *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, pages 403–407, Chichester, 1998. John Wiley & Sons.