Semantics for Dynamic Logic Programming: A Principle-Based Approach[*]

José J. Alferes[1], Federico Banti[1], Antonio Brogi[2], and João A. Leite[1]

[1] CENTRIA, Universidade Nova de Lisboa, Portugal,
{jja,banti,jleite}@di.fct.unl.pt
[2] Dipartimento di Informatica, Università di Pisa, Italy,
brogi@di.unipi.it

Abstract. Over recent 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 most adequate to model updates. A systematic approach to face this question is to identify general principles against which such semantics could be evaluated. In this paper we motivate and introduce a new such principle – the *refined extension principle* – which is complied with by the stable model semantics for (single) logic programs. It turns out that none of the existing semantics for logic program updates, even though based on stable models, complies with this principle. For this reason, we define a refinement of the dynamic stable model semantics for Dynamic Logic Programs that complies with the principle.

1 Introduction and Motivation

Most of the research in the field of logic programming for representing knowledge that evolves with time has focused on changes in the extensional part of knowledge bases (factual events or observations). This is what happens with the event calculus [10], logic programming forms of the situation calculus [15,17] and logic programming representations of action languages [9]. In all of these, the problem of updating the intensional part of the knowledge base (rules or action descriptions) remains basically unexplored.

In recent years, some amount of effort was devoted to explore the problem of updates in a logic programming setting leading to different framework proposals and semantics [1,4,6,11,13,14,19,21]. According to these proposals, knowledge is given by a sequence of logic programs (or a Dynamic Logic Program) where each is to be viewed as an update to the previous ones. Most of the existing semantics are based on the notion of causal rejection of rules [13] i.e., the rejection of a prior rule if there is a newer one that conflicts with it, and on a notion of default assumptions that can be added to the theory. Different notions of rejection

---

and default assumptions lead to different semantics, namely the justified update semantics [13,14], the dynamic stable model semantics [1,2,11] and the update answer-set semantics [6][1]. While such existing semantics based on causal rejection coincide on a large class of program updates, they essentially differ on the extent to which they are immune to some apparently inoffensive updates (e.g. tautologies[2]). Take, for example, the program $P_1 = \{a.\}$ and update it with $P_2 = \{not\,a \leftarrow not\,a\}$. Intuitively one would expect the update of P_1 with P_2 not to change the semantics because the only rule of P_2 is a tautology. This is not the case according to the semantics of justified updates which admits, after the update, the models $\{a\}$ and $\{\}$. Similar behaviours are exhibited by the update answer-set semantics [6]. Examples such as this one were one of the main reasons for the introduction of the dynamic stable model semantics [1,2] which properly deals with them. Unfortunately, there still remain examples involving tautological updates where none of the existing semantics behaves as expected. Let us now show an example to illustrate the problem.

Example 1. Consider the program P_1 describing some knowledge about the sky. At 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.

$$P_1 : day \leftarrow not\,night. \qquad stars \leftarrow night, not\,cloudy.$$
$$night \leftarrow not\,day. \qquad not\,stars.$$

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

$$P_2 : stars \leftarrow stars.$$

This tautological update introduces the new dynamic stable model $\{night, stars\}$. Furthermore these results are shared by all other semantics for updates based on causal rejection [1,4,6,11,13,14]. We argue that this behaviour is counterintuitive as the addition of the tautology in P_2 should not add new models.

This alone should be enough to have us start a quest for a semantics for updates that is immune to tautologies. But the problem runs deeper. Typically, these tautological updates are just particular instances of more general updates that should be ineffective but, in reality, cause the introduction of new models e.g. those with a rule whose head is *self-dependent*[3] as in the following example:

Example 2. Consider again program P_1 of Example 1, and replace P_2 with

$$P_2 : stars \leftarrow venus. \qquad venus \leftarrow stars.$$

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

---

[1] In this paper, we only consider semantics based on the notion of causal rejection.

[2] By a tautology we mean a rule of the form $L \leftarrow Body$ with $L \in Body$.

[3] For the definition of self-dependent literals in a logic program, see [3].

$\{night, stars, venus\}$. 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.

On the basis of these considerations, it is our stance that, besides the principles used to analyze and compare these semantics, described in [6,11], another important principle is needed to test the adequacy of semantics of logic program updates in some important situations, in particular those concerning the unwanted generation of new dynamic stable models when certain sets of rules are added to a dynamic logic program. It is worth noting that an update with the form of P_2 in Example 2 may have the effect of eliminating previously existing models, this often being a desired effect, as illustrated by the following example:

Example 3. Consider program P_1 with the obvious intuitive reading: one is either alone or with friends, and one is either happy or depressed.

$$P_1 : friends \leftarrow not\, alone. \qquad happy \leftarrow not\, depressed.$$
$$alone \leftarrow not\, friends. \qquad depressed \leftarrow not\, happy.$$

This program has four dynamic stable models namely, $\{friends, depressed\}$, $\{friends, happy\}$, $\{alone, happy\}$ and $\{alone, depressed\}$. Suppose now the program is updated with the following program (similar to P_2 used in Example 2):

$$P_2 : depressed \leftarrow alone. \qquad alone \leftarrow depressed.$$

This update specified by P_2 eliminates two of the dynamic stable models, leaving only $\{friends, happy\}$ and $\{alone, depressed\}$, this being a desirable effect.

In this paper we propose a new principle, that we call the *refined extension principle*, which can be used to compare different semantics for updates based on the stable model semantics — as is the case of all the above mentioned.

To this purpose, we start with the simple case of a single logic program and set forth the *refined extension principle* which, if complied with by a semantics, specifies some conditions under which rules can be safely added without introducing new models according to that semantics. Notably, the stable model semantics [8] complies with this principle. Informally, the semantics based on stable models can be 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.

Subsequently, we generalize this principle by lifting it to the case of semantics for dynamic logic programs. Not unexpectedly, given the examples above, it turns out that none of the existing semantics for updates based on causal rejection complies with such principle, which leads us to introduce a new semantics for dynamic logic programs, namely the *refined dynamic stable model semantics*, which complies with the refined extension principle. The *refined dynamic stable model semantics* is obtained by refining the dynamic stable model semantics [1,2,

11] which, of all the existing semantics, as shall be seen, is the one that complies with 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 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 complies with 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   Preliminaries

In this paper we extensively use the concept of generalized logic programs [16] i.e. logic programs that allow for default negation both in the bodies as well as in the heads of their rules. A generalization of the stable models semantics for normal logic programs [8] to the class of generalized programs was defined by Lifschitz and Woo [16]. Here we present such semantics differently from [16], the equivalence of both definition being proven in [2].

Let $\mathcal{A}$ be a set of propositional **atoms**. A **default literal** is an atom preceded by not. A **literal** is either an atom or a default literal. A **rule** r is an ordered pair $H(r) \leftarrow B(r)$ where $H(r)$ (dubbed the head of the rule) is a literal and $B(r)$ (dubbed the body of the rule) is a finite set of literals. A rule with $H(r) = L_0$ and $B(r) = \{L_1, \dots, L_n\}$ will simply be written as $L_0 \leftarrow L_1, \dots, L_n$. A rule with $H(r) = L_0$ and $B(r) = \{\}$ is called a fact and will simply be written as L_0. A **generalized logic program** (GLP) P, in $\mathcal{A}$, is a finite or infinite set of rules. By $P_\emptyset$ we mean an empty set of rules. If $H(r) = A$ (resp. $H(r) = not\,A$) then $not\,H(r) = not\,A$ (resp. $not\,H(r) = A$). Two rules r and r' are **conflicting**, denoted by $r \bowtie r'$, iff $H(r) = not\,H(r')$.

An **interpretation** M of $\mathcal{A}$ is a set of atoms ($M \subseteq \mathcal{A}$). An atom A is true in M, denoted by $M \vDash A$, iff $A \in M$, and false otherwise. A default literal $not\,A$ is true in M, denoted by $M \vDash not\,A$, iff $A \notin M$, and false otherwise. A set of literals B is true in M, denoted by $M \vDash B$, iff each literal in B is true in M.

An interpretation M of $\mathcal{A}$ is a **stable model** (or answer set) of a generalized logic program P iff

$$M' = least\,(P \cup \{not\,A \mid A \notin M\})$$

where $M' = M \cup \{not\_A \mid A \notin M\}$, A is an atom, and $least(.)$ denotes the least model of the definite program obtained from the argument program by replacing every default literal $not\,A$ by a new atom not_A[4].

A **dynamic logic program** (DLP) is a sequence of generalized logic programs. Let $\mathcal{P} = (P_1, ..., P_s)$ and $\mathcal{P}' = (P'_1, ..., P'_s)$ be two DLPs. We use $\rho\,(\mathcal{P})$ to denote the multiset of all rules appearing in the programs $P_1, ..., P_s$, and $\mathcal{P} \cup \mathcal{P}'$

---

[4] This amounts to determining the least model of the argument program, treating default literals as positive atoms.

to denote the DLP $(P_1 \cup P_1', ..., P_s \cup P_s')$. According to all semantics based on causal rejection of rules, an interpretation M models a DLP iff

$$M' = \Gamma(\mathcal{P}, M)$$

where

$$\Gamma(\mathcal{P}, M) = least\,(\rho(\mathcal{P}) - Rej(\mathcal{P}, M) \cup Def(\mathcal{P}, M))$$

and where $Rej(\mathcal{P}, M)$ stands for the set of rejected rules and $Def(\mathcal{P}, M)$ for the set of default assumptions, both given $\mathcal{P}$ and M. Intuitively, we first determine the set of rules from $\mathcal{P}$ that are not rejected, i.e. $\rho(\mathcal{P}) - Rej(\mathcal{P}, M)$, to which we add a set of default assumptions $Def(\mathcal{P}, M)$. Note the similarity to the way stable models of generalized logic programs are obtained, where default assumptions of the form $not\,A$ are added for every $A \notin M$.

From [11] it is easy to see that all existing semantics for updates based on causal rejection are parameterizable using different definitions of $Rej(\mathcal{P}, M)$ and $Def(\mathcal{P}, M)$. The **dynamic stable model** semantics for DLP [1,11] is obtained with the following definitions:

$$Rej(\mathcal{P}, M) = \{r \mid r \in P_i, \exists r' \in P_j, i < j, r \bowtie r', M \vDash B(r')\}$$
$$Def(\mathcal{P}, M) = \{not\,A \mid \nexists r \in \rho(\mathcal{P}), H(r) = A, M \vDash B(r)\}$$

3   Refined Extensions

As mentioned in the Introduction, we are interested in exploring conditions guaranteeing that the addition of a set of rules to a (dynamic) logic program does not generate new (dynamic) stable models. In this Section, we motivate and introduce the notion of refined extension for both the case of single logic programs and dynamic logic programs, which, together with the results proven, constitute a step in such direction.

3.1   Refined Extensions of Generalized Logic Programs

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 i.e., the stable models of a generalized logic program P are those interpretations M such that M' coincides with the least Herbrand model of the program[5] $P \cup \{not\,A \mid A \notin M\}$. In general, several semantics share the characteristic that the models of a program P can be characterized as the least Herbrand model of the program $P \cup Assumptions\,(P, M)$, where $Assumptions\,(P, M)$ is simply a set of default literals whose definition depends on the semantics in use. Note that all of the stable models [8], the well-founded [7] and the weakly perfect model semantics [18] can be defined in this

---

[5] Here and elsewhere, whenever we mention the Herbrand model of a GLP, we mean the Herbrand model of the definite logic obtained from the GLP by replacing every default literal $not\,A$ by a new atom not_A.

way. As mentioned above, the stable model semantics of normal and generalized programs can be obtained by establishing that

$$Assumptions\,(P, M) = \{\,not\,A \mid A \notin M\,\}\,.$$

In the sequel, if Sem is a semantics definable in such a way, by $Sem(P)$ we denote the set of all models of a given program P, according to Sem.

With the intention of defining the refined extension principle, we first need to set forth some intermediate notions, namely that of *syntactic extension* of a program. Intuitively 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. Formally:

Definition 1. *Let P and E be generalized logic programs. We say that $P \cup E$ is a **syntactic extension** of P iff $least\,(P) = least\,(P \cup E)$.*

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 $P \cup E$. We can then apply the concept of syntactical extension to verify whether the addition of the rules in E does not influence the computation of M. If this is the case for all models of the program $P \cup E$, according to Sem, we say that $P \cup E$ is a *refined extension* of the original program P. Formally:

Definition 2. *Let P and E be generalized logic program, M an interpretation, and Sem a semantics for generalized logic programs. We say that $P \cup E$ is an **extension** of P with respect to Sem and 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 with respect to Sem iff $P \cup E$ is an extension of P with respect to Sem and all models in $Sem(P \cup E)$.*

Example 4. Let P_1 and P_2 be the programs:

$$
\begin{aligned}
P_1 : \ &day \leftarrow not\,night. \qquad &&stars \leftarrow night, not\,cloudy.\\
&night \leftarrow not\,day. \qquad &&not\,stars.\\
P_2 : \ &stars \leftarrow stars
\end{aligned}
$$

It is clear that no matter what the added assumptions $(Assumptions\,(P_1 \cup P_2, M))$ are, given any model M, the least model of $P_1 \cup Assumptions\,(P_1 \cup P_2, M)$ is the same as the least model of $P_1 \cup Assumptions\,(P_1 \cup P_2, M) \cup P_2$. Thus, $P_1 \cup P_2$ is a refined extension of P_1.

We are now ready to formulate the refined extension principle for semantics of generalized logic programs. Intuitively, a semantics complies with the refined extension principle iff a refined extension of a program P does not have more models than P.

Principle 1 (Refined extension – static case) *A semantics Sem for generalized logic programs complies with the refined extension principle iff for any two generalized logic programs P and 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 complies with the principle, the addition of tautologies does not generate new models.

Proposition 1. *Let Sem be a semantics for generalized programs, P a generalized program, and τ a tautology. If Sem complies with the refined extension principle then*

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

Most importantly, the stable model semantics complies with the refined extension principle, as stated in the following proposition:

Proposition 2. *The stable model semantics complies with 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. The converse is also true i.e., the addition of tautologies to a generalized program does not eliminate existing stable models.

3.2 Refined Extensions of Dynamic Logic Programs

We now generalize the refined extension principle to the case of dynamic logic programs, so as to guarantee that certain updates do not generate new models.

Definition 3. *Let $\mathcal{P}$ and $\mathcal{E}$ be two dynamic logic programs[6], Sem a semantics for dynamic logic programs and M an interpretation. We say that $\mathcal{P} \cup \mathcal{E}$ is an **extension** of $\mathcal{P}$ with respect to M iff*

$$[\rho\,(\mathcal{P}) - Rej\,(\mathcal{P} \cup \mathcal{E}, M)] \cup Def\,(\mathcal{P} \cup \mathcal{E}, M) \cup [\rho\,(\mathcal{E}) - Rej\,(\mathcal{P} \cup \mathcal{E}, M)]$$

is a syntactical extension of

$$[\rho\,(\mathcal{P}) - Rej\,(\mathcal{P} \cup \mathcal{E}, M)] \cup Def\,(\mathcal{P} \cup \mathcal{E}, M)$$

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

Note that the above definition is the straightforward lifting of Definition 2 to the case of dynamic logic programs. Roughly speaking, we simply replaced P and E with $\rho\,(\mathcal{P}) - Rej\,(\mathcal{P} \cup \mathcal{E}, M)$ and $\rho\,(\mathcal{E}) - Rej\,(\mathcal{P} \cup \mathcal{E}, M)$, respectively.

The refined extension principle is then formulated as follows.

---

[6] Here, and elsewhere, we assume that the sequences of GLPs (or DLPs) $\mathcal{P}$ and $\mathcal{E}$ are of the same length.

Principle 2 (Refined extension principle) *A semantics Sem for dynamic logic programs complies with the refined extension principle iff for any dynamic logic programs $\mathcal{P}$ and $\mathcal{P} \cup \mathcal{E}$, if $\mathcal{P} \cup \mathcal{E}$ is a refined extension of $\mathcal{P}$ then*

$$Sem(\mathcal{P} \cup \mathcal{E}) \subseteq Sem(\mathcal{P}).$$

Unfortunately, as we already pointed out in the Introduction, none of the existing semantics for dynamic logic programs based on causal rejection comply with the refined extension principle.

Example 5. Consider again the program P_1 and P_2 of Example 2. The presence of the new model contrasts with the refined extension principle. Indeed, if we consider the empty update $P_\emptyset$, then the dynamic logic program $(P_1, P_\emptyset)$ has only one stable model (viz., $\{day\}$). Since, as the reader can check, (P_1, P_2) is a refined extension of $(P_1, P_\emptyset)$ then, according to the principle, all models of (P_1, P_2) should also be models of $(P_1, P_\emptyset)$. This is not the case for existing semantics.

As for the case of generalized programs, if we consider a semantics *Sem* for dynamic logic programs that complies with the principle, the addition of tautologies does not generate new models. This is stated in the following proposition that lifts Proposition 1 to the case of dynamic logic programs.

Proposition 3. *Let Sem be a semantics for dynamic logic programs, $\mathcal{P}$ a dynamic logic program, and $\mathcal{E}$ a sequence of sets of tautologies. If Sem complies with the refined extension principle then*

$$Sem(\mathcal{P} \cup \mathcal{E}) \subseteq Sem(\mathcal{P}).$$

4   Refined Semantics for Dynamic Logic Programs

In this Section we define a new semantics for dynamic logic programs that complies with the refined extension principle. Before proceeding we will take a moment to analyze the reason why the dynamic stable model semantics fails to comply with the refined extension principle in Example 1. In this example, the extra (counterintuitive) stable model $\{night, stars\}$ is obtained because the tautology $stars \leftarrow stars$ in P_2 has a true body in that model, hence rejecting the fact $not\,stars$ of P_1. After rejecting this fact, it is possible to consistently conclude $stars$, and thus verify the fixpoint condition, via the rule $stars \leftarrow night, not\,cloudy$ of P_1.

Here lies the matrix of the undesired behaviour exhibited by the dynamic stable model semantics: One of the two conflicting rules in the same program (P_1) is used to support a later rule (of P_2) that actually removes that same conflict by rejecting the other conflicting rule. 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 other. This can be obtained with a slight modification to

the notion of rejected rules of the dynamic stable model semantics, namely by also allowing rules to reject other rules in the same state. Since, according to the dynamic stable model semantics, rejected rules can reject other rules, two rules in the same state can reject each other, thus avoiding the above described behaviour.

Definition 4. *Let $\mathcal{P}$ be a dynamic logic program and M an interpretation. M is a* refined dynamic stable model *of $\mathcal{P}$ iff*

$$M' = \Gamma^S\left(\mathcal{P}, M\right)$$

where
$$\Gamma^S\left(\mathcal{P}, M\right) = least\left(\rho\left(\mathcal{P}\right) - Rej^S\left(\mathcal{P}, M\right) \cup Def\left(\mathcal{P}, M\right)\right)$$

and
$$Rej^S\left(\mathcal{P}, M\right) = \{r \mid r \in P_i, \exists r' \in P_j, i \leq j, r \bowtie r', M \vDash B(r')\}$$

At first sight this modification could seem to allow the existence of models in cases where a contradiction is expected (e.g. 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 could have undesirable models. Notably, the opposite is actually true (cf. theorem 4 below), and the refined dynamic stable models are always dynamic stable models, i.e., allowing the rejection of rules by rules in the same state does not introduce extra models. Consider a DLP $\mathcal{P}$ with two conflicting rules (with heads A and *not* A) in one of its programs P_i. Take an interpretation M where the bodies of those two rules are both true (as nothing special happens if a rule with false body is rejected) and check if M is a refined dynamic stable model. By definition 4, 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 cannot be considered as a default assumption, i.e., does not belong to $Def\left(\mathcal{P}, M\right)$. This means that all the information about A with origin in P_i or any previous state is deleted. Since M' must contain either A or *not_A*, 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 comply with the refined extension principle, as stated by the following theorem.

Theorem 3. *The refined dynamic stable model semantics complies with the refined extension principle.*

By Proposition 3, it immediately follows from this theorem that the addition of tautologies never adds models in this semantics. Note that the converse is also true: the addition of tautologies does not eliminate existing models in the refined semantics i.e., the refined dynamic stable model semantics is immune to tautologies. Moreover the refined semantics preserves all the desirable properties of the previous semantics for dynamic logic programs [6,11].

To give an insight view of the behaviour of the refined semantics, we now illustrate how the counterintuitive results of example 1 are eliminated.

Example 6. Consider again the DLP $\mathcal{P} = (P_1, P_2)$ of example 1

$$P_1 : day \leftarrow not\, night. \qquad stars \leftarrow night, not\, cloudy.$$
$$night \leftarrow not\, day. \qquad not\, stars.$$
$$P_2 : stars \leftarrow stars$$

This DLP has one refined dynamic 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 now show that M is a refined dynamic stable model. First of all we compute the sets $Rej^S(\mathcal{P}, M)$ and $Def(\mathcal{P}, M)$:

$$Rej^S(\mathcal{P}, M) = \{stars \leftarrow night, not\, cloudy.\}$$
$$Def(\mathcal{P}, M) = \{not\, night, not\, stars, not\, cloudy\}$$

Then we check whether M is a refined dynamic stable model according to definition 4. Indeed:

$$\Gamma^S(\mathcal{P}, M) = least\left((P_1 \cup P_2) - Rej^S(\mathcal{P}, M) \cup Def(\mathcal{P}, M)\right) =$$
$$= \{day, not\_night, not\_stars, not\_cloudy\} = M'$$

As mentioned before, the dynamic stable model semantics, besides M, also admits the interpretation $N = \{night, stars\}$ as one of its models, thus violating the refined extension principle. We now show that N is not a refined dynamic stable model. As above we compute the sets:

$$Rej^S(\mathcal{P}, N) = \{not\, stars; \quad stars \leftarrow night, not\, cloudy.\}$$
$$Def(\mathcal{P}, N) = \{not\, day, not\, cloudy\}$$

Hence:

$$\Gamma^S(\mathcal{P}, N) = least\left((P_1 \cup P_2) - Rej^S(\mathcal{P}, N) \cup Def(\mathcal{P}, N)\right) =$$
$$= \{night, not\_day, not\_cloudy\} \neq N'$$

From where we conclude, according to definition 4, that N is not a refined dynamic stable model.

5 Comparisons

Unlike the refined dynamic stable model semantics presented here, none of the other existing semantics based on causal rejection respect the refined extension principle and, consequently, none is immune to the addition of tautologies.

It is clear from the definitions that the refined semantics coincides with the dynamic stable model semantics [1,2,11] for sequences of programs with no conflicting rules in a same program. This means that the dynamic stable model semantics complies with the refined extension principle for such class of sequences of programs, and 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. Another limitation would result from the fact that updates also provide a tool to remove inconsistency in programs by rejecting conflicting rules. Such feature would be completely lost in that case.

With respect to the other semantics based on causal rejection, it is not even the case that the principles is satisfied by sequences in that restricted class. The update answer-set semantics [6] (as well as inheritance programs of [4]), and the justified update semantics [13,14] fail to be immune to tautologies even when no conflicting rules occur in the same program:

Example 7. Consider the DLP $\mathcal{P} = (P_1, P_2, P_3)$ where (taken from [11]):

$$P_1 : day. \qquad P_2 : not\,day. \qquad P_3 : day \leftarrow day.$$

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 [13,14] and the dynamic stable model semantics [1,11] select $\{day\}$ as the only model, the update answer set semantics of [6] (as well as inheritance programs of [4]) associates two models, $\{day\}$ and $\{\}$, with such sequence of programs[7].

While the semantics of justified updates [13,14] works properly for the above example, there are classes of program updates for which it does not comply with the refined extension principle:

Example 8. Consider the DLP $\mathcal{P} = (P_1, P_2)$ where (taken from [11]):

$$P_1 : day. \qquad P_2 : not\,day \leftarrow not\,day.$$

According to the semantics of justified updates, (P_1, P_2) has two models, $M_1 = \{day\}$ and $M_2 = \{\}$, whereas $(P_1, P_\emptyset)$ has the single model M_1, thus violating the refined extension principle.

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

Theorem 4. *Let $\mathcal{P}$ be a DLP, and M an interpretation. If M is a refined dynamic stable model then M is a dynamic stable model.*

This result generalizes to all the other semantics since the dynamic stable model is the most credulous of the existing semantics. Indeed, each dynamic stable model is also a model in the justified update semantics [11]. Inheritance programs are defined for disjunctive logic programs, but if we restrict to the non disjunctive case, this semantics coincides with the update answer-set semantics of [6], and each dynamic stable model is also an update answer-set [11].

---

[7] Notice that, strictly speaking, the semantics [4,6] 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.

The analysis of the semantics for updates that are not based on causal rejection (e.g. [19,21]) is beyond the scope of this paper. Even though the refined extension principle is not directly applicable to evaluate such semantics, they do not appear satisfactory in the way they deal with simple examples, as shown in [6,11] where a deeper analysis of such semantics can be found.

6   Concluding Remarks

We have started by motivating and introducing a new general principle – the refined extension principle – that can be used to compare semantics of logic programs that are obtainable from the least model of the original program after the addition of some (default) assumptions. For normal logic programs, both the well-founded and stable models are obtainable as such. The principles states that the addition of rules that do not change the least model of the program plus assumptions can never generate new models. A special case of this principle concerns the addition of tautologies. Not surprisingly, the stable model semantics for normal and generalized logic programs complies with this principles.

We have generalized this principle for the case of dynamic logic programs, noticing that none of the existing semantics complies with it. A clear sign of this, already mentioned in the literature [6,11], was the fact that none of these existing semantics is immune to tautologies. We have shown that, among these existing semantics, the dynamic stable model semantics [1,2,11] is the one that complies with the principle for a wider class, viz., the class of dynamic logic programs without conflicting rules in a same program in the sequence.

To remedy this problem exhibited by the existing semantics, and guided by the refined extension principle, we have introduced a new semantics for dynamic logic programs – the refined dynamic stable model semantics – and shown that it complies with such principle. Furthermore, we have obtained that this semantics is immune to tautologies. We have compared the new semantics with extant ones and have shown that, besides the difference regarding the principle, this semantics is more credulous than any of the others, in the sense of admitting a smaller set of stable models (thus yielding a larger intersection of stable models).

Future lines of research include extending this study to deal with multi-dimensional updates [11,12]. Multi-dimensional updates can be viewed as a tool for naturally encoding and combining knowledge bases that incorporate information from different sources, which may evolve in time. Existing semantics suffer from the same kind of problems we identified here for dynamic logic program.

Another important line for future work is the implementation of the refined semantics. We intend to do it by finding a transformation from dynamic programs to generalized programs, such that the stable models of the latter are in a one-to-one correspondence with the refined dynamic stable models of the former, similarly to what has been done for e.g., the dynamic stable semantics. The existence of such a transformation would directly provide a means to implement the refined dynamic stable model semantics of dynamic logic programs, by

resorting to an implementation for answer-set programming, such as SMODELS [20] or DLV [5].

References

1. J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski. Dynamic logic programming. In *Procs. of KR'98*. Morgan Kaufmann, 1998.
2. 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.
3. K. R. Apt and R. N. Bol. Logic programming and negation: A survey. *The Journal of Logic Programming*, 19 & 20:9–72, May 1994.
4. F. Buccafurri, W. Faber, and N. Leone. Disjunctive logic programs with inheritance. In *Procs. of ICLP'99*. MIT Press, 1999.
5. DLV. The DLV project - a disjunctive datalog system (and more), 2000. Available at http://www.dbai.tuwien.ac.at/proj/dlv/.
6. T. Eiter, M. Fink, G. Sabbatini, and H. Tompits. On properties of update sequences based on causal rejection. *Theory and Practice of Logic Programming*, 2(6), 2002.
7. 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.
8. M. Gelfond and V. Lifschitz. The stable model semantics for logic programming. In R. Kowalski and K. A. Bowen, editors, *Procs. of ICLP'88*. MIT Press, 1988.
9. M. Gelfond and V. Lifschitz. Representing actions and change by logic programs. *Journal of Logic Programming*, 17:301–322, 1993.
10. R. Kowalski and M. Sergot. A logic-based calculus of events. *New Generation Computing*, 4(1):67–95, 1986.
11. J. A. Leite. *Evolving Knowledge Bases*, volume 81 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, 2003.
12. J. A. Leite, J. J. Alferes, and L. M. Pereira. Multi-dimensional dynamic knowledge representation. In *Procs. of LPNMR'01*, volume 2173 of *LNAI*. Springer, 2001.
13. J. A. Leite and L. M. Pereira. Generalizing updates: From models to programs. In *Procs. of LPKR'97*, volume 1471 of *LNAI*. Springer Verlag, 1997.
14. J. A. Leite and L. M. Pereira. Iterated logic program updates. In *Procs. of JICSLP'98*. MIT Press, 1998.
15. H. Levesque, F. Pirri, and R. Reiter. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science*, 03, 1998.
16. V. Lifschitz and T. Woo. Answer sets in general non-monotonic reasoning (preliminary report). In *Procs of KR'92*. Morgan-Kaufmann, 1992.
17. 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.
18. H. Przymusinska and T. Przymusinski. Weakly perfect model semantics. In *Procs. of ICLP'88*. MIT Press, 1988.
19. C. Sakama and K. Inoue. Updating extended logic programs through abduction. In *Procs. of LPNMR'99*, volume 1730 of *LNAI*. Springer.
20. SMODELS. The SMODELS system, 2000. Available at http://www.tcs.hut.fi/Software/smodels/.
21. Y. Zhang and N.Y. Foo. Updating logic programs. In *Procs of ECAI'98*. John Wiley & Sons, 1998.