José Júlio Alferes
Federico Banti
Antonio Brogi
João Alexandre Leite

The Refined Extension Principle for Semantics of Dynamic Logic Programming

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*. Such principle 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 generalisations of the stable model semantics, comply 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.

Keywords: Logic Programming, Dynamic Logic Programming, Updates, Belief Change, Non-monotonic Reasoning, Answer-set Programming, Stable Model Semantics

1. Introduction and Motivation

Until recently, 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 [9], logic programming forms of the situation calculus [15, 17] and logic programming representations of action languages [8]. 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, 5, 10, 12, 13, 20, 21]. According to most of these proposals, knowledge is given by a sequence of logic programs (or a Dynamic Logic Program [1, 2, 10]) 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* introduced by Leite and Pereira in [12] i.e., the rejection of a prior rule if there is a newer one that conflicts with it, and on

Special Issue on Reasoning about Action and Change
Edited by **Gerhard Brewka and Pavlos Peppas**

Studia Logica **79**: 7–32, 2005.
© 2005 *Kluwer Academic Publishers. Printed in the Netherlands.*

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 [12, 13], the dynamic stable model semantics [1, 2, 10] and the update answer-set semantics [5][1].

While such existing semantics based on causal rejection coincide on a large class of dynamic logic programs, 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 $\{\}$[3].

Examples such as this one motivated 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 1.1. 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.

$$\begin{array}{ll} P_1: & day \leftarrow not\ night. \\ & night \leftarrow not\ day. \\ & stars \leftarrow night, not\ cloudy. \\ & not\ stars. \end{array}$$

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

$$P_2: \quad stars \leftarrow stars.$$

This tautological update introduces a new dynamic stable model. In fact, besides the intuitively correct dynamic stable model $\{day\}$, the dynamic stable model semantics also admits the model $\{night, stars\}$. Technically, this happens because the rule in P_2, which has a true body in the latter model, causally rejects the fact $not\ stars$ of P_1. Furthermore, these results

[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] Similar behaviours are exhibited by the update answer-set semantics [5].

are shared by all other existing semantics for updates based on causal rejection [1, 4, 5, 10, 12, 13]. 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 the 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*[4] as in the following example:

EXAMPLE 1.2. Consider again program P_1 of Example 1.1, and replace P_2 with

$$\begin{array}{rl} P_2: & stars \leftarrow venus. \\ & venus \leftarrow stars. \end{array}$$

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, $\{night, stars, venus\}$. Intuitively, 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 [5, 10], 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, however, that an update with the form of P_2 in Example 1.2 may have the effect of eliminating previously existing models, this often being a desired effect, as illustrated by the following example:

EXAMPLE 1.3. Consider program P_1 with the obvious intuitive reading: one is either alone or with friends, and one is either happy or depressed.

$$\begin{array}{rl} P_1: & friends \leftarrow not\, alone. \\ & alone \leftarrow not\, friends. \\ & happy \leftarrow not\, depressed. \\ & depressed \leftarrow not\, happy. \end{array}$$

This program has four dynamic stable models namely, $\{friends, depressed\}$, $\{friends, happy\}$, $\{alone, happy\}$ and $\{alone, depressed\}$. Suppose now

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

that the program is updated with the following program (similar to P_2 used in Example 1.2):

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

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 [7] complies with this principle. Informally, the semantics based on stable models can be obtained by taking the least model of the definite program obtained by adding some assumptions (default negations) to the initial program. Intuitively, 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, 10] 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. In Appendix, the reader can find all the proofs of the results presented throughout the paper.

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 [7] 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 $B(r) = \{L_1, \ldots, L_n\}$ will simply be written as $L_0 \leftarrow L_1, \ldots, L_n$. 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[5]. 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.

A rule r is satisfied by an interpretation M iff whenever $M \vDash B(r)$ then $M \vDash H(r)$. An interpretation M is a model of a program P iff M satisfies all rules in P.

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

$$\overline{M} = least\,(P \cup \{not\ A \mid A \notin M\}) \tag{1}$$

where $\overline{M} = M \cup \{not\_A \mid A \notin M\}$, 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[6].

---

[5] As usual, we assume that each rule is identified by a unique name. Consequently, we can have, in the same program, two distinct rules, r and r', such that $H(r) = H(r')$ and $B(r) = B(r')$.

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

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 set of all rules appearing in the programs $P_1, ..., P_s$, and $\mathcal{P} \cup \mathcal{P}'$ 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

$$\overline{M} = \Gamma(\mathcal{P}, M) \tag{2}$$

where

$$\Gamma(\mathcal{P}, M) = least(\rho(\mathcal{P}) - Rej(\mathcal{P}, M) \cup Def(\mathcal{P}, M)) \tag{3}$$

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 [10] 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 [1, 10] for DLP is defined as follows:

Definition 2.1. Let $\mathcal{P}$ be a dynamic logic program and M an interpretation. M is a *dynamic stable model* of $\mathcal{P}$ iff

$$\overline{M} = \Gamma^D(\mathcal{P}, M) \tag{4}$$

where

$$\Gamma^D(\mathcal{P}, M) = least\left(\rho(\mathcal{P}) - Rej^D(\mathcal{P}, M) \cup Def(\mathcal{P}, M)\right) \tag{5}$$

and

$$Rej^D(\mathcal{P}, M) = \left\{r \mid r \in P_i, \exists r' \in P_j, i < j, r \bowtie r', M \vDash B(r')\right\} \tag{6}$$

$$Def(\mathcal{P}, M) = \{not\ A \mid \nexists r \in \rho(\mathcal{P}), H(r) = A, M \vDash B(r)\} \tag{7}$$

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 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 $\overline{M}$ coincides with the least model of the program[7]

$$P \cup \{not\, A \mid A \notin M\}$$

In general, several semantics share the characteristic that the models of a program P can be characterized as the least 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 [7], the well-founded [6] and the weakly perfect model semantics [19] can be defined in this 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 model of P. Formally:

Definition 3.1. 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 $P \cup E$ according to Sem is computed as the least 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:

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

Definition 3.2. 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 3.3. Let P_1 and P_2 be the programs:

$$\begin{array}{ll} P_1: & day \leftarrow not\, night. \\ & night \leftarrow not\, day. \\ & stars \leftarrow night, not\, cloudy. \\ & not\, stars. \end{array}$$

$$P_2: \quad stars \leftarrow stars$$

It is clear that irrespective of what the set $Assumptions\,(P_1 \cup P_2, M)$ of added assumptions is, given any model M, we have that the least model of $P_1 \cup Assumptions\,(P_1 \cup P_2, M)$ coincides with 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 3.4 (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 3.5. 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 3.6. Let P be any generalized logic program, $P \cup E$ be a refined extension of P, and M a stable model of $P \cup E$. Then M is also a stable model of P.

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, according to the semantics that comply with it, certain updates do not generate new models.

DEFINITION 3.7. Let $\mathcal{P}$ and $\mathcal{E}$ be two dynamic logic programs[8], 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 3.2 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.

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

Principle 3.8 (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

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

Again, this principle amounts to the straightforward lifting of Principle 3.4 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 3.9. Consider again the program P_1 and P_2 of Example 1.2. 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.

If we consider infinite logic programs, there is also another class of examples where all the previously existing semantics based on causal rejection fail to satisfy the refined extension principle. This class of examples consists of sequences of updates containing an infinite number of conflicting rules whose bodies are satisfied i.e., an infinite number of potential contradictions where each is removed by a later update, of which the following example serves as illustration.

Example 3.10. Consider an infinite 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 q be an unary predicate and $\mathcal{A}$ the set of propositional atoms consisting of all the predicates of the form $q(n)$.

Consider the following programs:

$$P_1: \quad q(X). \\ \quad\quad not\ q(X).$$

$$P_2: \quad q(X) \leftarrow q(X+1).$$

Since P_1 and P_2 contain the variable X, in order to compute the stable model semantics, we have to consider the ground versions of the two

programs i.e., the following two infinite programs:

$$P_1^{ground}: \quad \begin{array}{l} q(n). \ \ \forall\, n \in \mathbb{N} \\ not\, q(n). \ \ \forall\, n \in \mathbb{N} \end{array}$$

$$P_2^{ground}: \quad q(n) \leftarrow \ q(n+1). \ \ \forall\, n \in \mathbb{N}$$

All existing semantics based on causal rejection admit the interpretation $I = \mathcal{A}$, consisting of all the predicates of the form $q(n)$, as a model of the program $(P_1^{ground}, P_2^{ground})$. Since the program $(P_1^{ground}, P_\emptyset)$ is clearly contradictory, hence having no stable models, and the program $(P_1^{ground}, P_2^{ground})$ is a refined extension of $(P_1^{ground}, P_\emptyset)$, then, according to the refined extension principle, it should be taken as contradictory and have no models.

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 3.5 to the case of dynamic logic programs.

PROPOSITION 3.11. 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 1.1. In this example, the extra (counterintuitive) dynamic 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 (Equation 5), 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 4.1. Let $\mathcal{P}$ be a dynamic logic program and M an interpretation. M is a *refined dynamic stable model* of $\mathcal{P}$ iff

$$\overline{M} = \Gamma^S(\mathcal{P}, M) \tag{8}$$

where

$$\Gamma^S(\mathcal{P}, M) = least\left(\rho(\mathcal{P}) - Rej^S(\mathcal{P}, M) \cup Def(\mathcal{P}, M)\right) \tag{9}$$

and

$$Rej^S(\mathcal{P}, M) = \left\{r \mid r \in P_i, \exists r' \in P_j, i \leq j, r \bowtie r', M \vDash B(r')\right\} \tag{10}$$

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 5.3 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. To better understand this behaviour, 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 4.1, 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, $not\ A$ cannot be considered as a default assumption, i.e., does not belong to $Def(\mathcal{P}, M)$ because of the rule with head A and true body. This means that all the information about A (a rule with head A or $not\ A$) with origin in P_i or any previous state is deleted. Since $\overline{M}$ 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. If such later update does not exist, then, even though the two rules that cause the contradiction reject each other, we still do not obtain any models.

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

By Proposition 3.11, 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, as stated by the following theorem.

THEOREM 4.3. *Let $\mathcal{P}$ be any DLP and $\mathcal{E}$ a sequence of sets of tautologies. M is a refined stable model of $\mathcal{P}$ iff M is a refined stable model of $\mathcal{P} \cup \mathcal{E}$.*

Moreover the refined semantics preserves all the desirable properties of the previous semantics for dynamic logic programs [5, 10].

To give an insight view on the behaviour of the refined semantics, we now illustrate how the counterintuitive results of example 1.1 are eliminated.

EXAMPLE 4.4. Consider again the DLP $\mathcal{P} = (P_1, P_2)$ of example 1.1

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

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 $Def(\mathcal{P}, M)$:

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

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

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

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:

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

From where we conclude, according to definition 4.1, 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, 10] 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 [5] (as well as inheritance programs of [4]), and the justified update semantics [12, 13] fail to be immune to tautologies even when no conflicting rules occur in the same program:

EXAMPLE 5.1. Consider the DLP $\mathcal{P} = (P_1, P_2, P_3)$ where (taken from [10]):

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

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 [12, 13] and the dynamic stable model semantics [1, 10] select $\{day\}$ as the only model, the update answer set semantics of [5] (as well as inheritance programs of [4]) associates two models, $\{day\}$ and $\{\}$, with such sequence of programs[9].

While the semantics of justified updates [12, 13] works properly for the above example, there are classes of programs where its behaviour shows its failure to comply with the refined extension principle:

EXAMPLE 5.2. Consider the DLP $\mathcal{P} = (P_1, P_2)$ where (taken from [10]):

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

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 5.3. *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.*

[9]Notice that, strictly speaking, the semantics [4, 5] 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 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 [10]. 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 [5], and each dynamic stable model is also an update answer-set [10].

The analysis of the semantics for updates that are not based on causal rejection (e.g. [20, 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 [5, 10] 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 principle states that the addition of rules that do not change the least model of the program together with the 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 [5, 10], was the fact that none of these existing semantics is immune to tautologies. We have illustrated how, among these existing semantics, the dynamic stable model semantics [1, 2, 10] 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 proved 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 [10, 11]. 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 all other semantics based on the causal rejection principle. 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 [18] or DLV [14].

Acknowledgments

A warm acknowledgment is due to Pascal Hitzler and Reinhard Kahle for helpful discussions. This work was partially supported by project FLUX (POSI/40958/SRI/2001), financed by FEDER, and by project SOCS (IST-2001-32530).

7. Appendix

In this Appendix, we present the proofs of the Theorems and Propositions found throughout the paper. Before we proceed, however, we introduce some notation and establish some properties of the least model, to be used subsequently.

7.1. Notation

Let L be a literal, τ a rule of the form $L_0 \leftarrow L_1, \ldots, L_n$, and P a program. We use:

- $\overline{L}$ to denote the literal not_A (resp. A) if L is the literal $not\ A$ (resp. A);
- $\overline{\tau}$ to denote the rule $\overline{L_0} \leftarrow \overline{L_1}, \ldots, \overline{L_n}$;
- $\overline{P}$ to denote the program $\{\overline{\tau} \mid \tau \in P\}$;
- $H(P)$ to denote the set of the heads of all rules in P i.e.,

$$H(P) = \{L \mid \exists \tau \in P,\ H(\tau) = L\} \tag{11}$$

Let $\mathcal{P}$ and $\mathcal{E}$ be two dynamic logic programs, and M an interpretation. We use:

- M^- to denote the set $\{not\ A \mid A \notin M\}$;
- $Generator^D(\mathcal{P}, M)$ to denote the program $\rho(\mathcal{P}) - Rej^D(\mathcal{P}, M) \cup Def(\mathcal{P}, M)$;
- $Generator^S(\mathcal{P}, M)$ to denote the program $\rho(\mathcal{P}) - Rej^S(\mathcal{P}, M) \cup Def(\mathcal{P}, M)$.

Let M be an interpretation and τ a rule.We say that τ is supported by M iff $M \models B(\tau)$.

7.2. Properties of the least model

Throughout the proofs of Theorems and Propositions we make use several properties of the least model of a definite logic program, that we now prove.

PROPERTY 1. Let P, U and R be definite logic programs, such that, for any rule $\tau \in U$, $B(\tau) \not\subseteq least(P)$, and $H(R) \subseteq least(P)$ then:

$$least(P \cup U \cup R) = least(P) \tag{12}$$

i.e., $P \cup R \cup U$ is a syntactic extension of P.

PROOF. The *least* operator is monotonous, hence

$$least(P) \subseteq least(P \cup U \cup R) \tag{13}$$

To prove the inverse inclusion, we start by proving that $least(P)$ is a model of $P \cup U \cup R$. First, $least(P)$ is the least model of P. Moreover, it is a model of U, since, by hypothesis, the body of any rule in U is not supported. Finally it is a model of R, since, by hypothesis, the head of any rule in R

is in $least(P)$. Thus, $least(P)$ is a model of $P \cup R \cup U$ and, as such, must include the least model of $P \cup R \cup U$, i.e.

$$least(P \cup R \cup U) \subseteq least(P) \tag{14}$$

From (13) and (14), (12) immediately follows. ■

PROPERTY 2. Let P be a definite logic program and Ta a program whose rules are tautologies i.e. rules of the form: $A \leftarrow Body$ where $A \in Body$. Then $P \cup Ta$ is a syntactic extension of P.

PROOF. Let U be the program consisting of all rules r' in Ta such that $H(r') \notin least(P)$ and R be the program obtained considering all rules r such that $H(r) \in least(P)$. Note that, since for all rules $r \in Ta$ we have that $H(r) \in B(r)$, the rules of U are such that $B(r) \not\subseteq least(P)$. From this and the definition of R, it follows that P, U and R satisfy the hypothesis of Property 1. Hence, by the thesis of such Property it follows immediately that $P \cup Ta$ is a syntactic extension of P. ■

7.3. Proofs of Theorems and Propositions

PROOF OF PROPOSITION 3.5. Let Sem be a semantics for generalized programs, P a generalized program, τ a tautology and M an interpretation such that $M \in Sem(P \cup \{\tau\})$. Then, by definition:

$$\overline{M} = least(P \cup \{\tau\} \cup Assumtions(P \cup \{\tau\}, M))$$

By Property 2 we have that

$$\overline{M} = least(P \cup Assumtions(P \cup \{\tau\}, M))$$

Hence $P \cup \{\tau\}$ is a refined extension of P. Since the semantics Sem complies with the refined extension principle, this implies $M \in Sem(P)$. Thus $Sem(P \cup \{\tau\}) \subseteq Sem(P)$ as desired. ■

PROOF OF PROPOSITION 3.6. Let P and E be two generalized programs and M an interpretation. Recall that, in the stable model semantics, independently of P, the set $Assumptions(P, M)$ is M^-. We simply have to prove that, if M is a stable model of $P \cup E$ and $P \cup E$ is an extension of P wrt. M, then M is a stable model of P. If M is a stable model of $P \cup E$, then, by definition of extension

$$\overline{M} = least(P \cup M^- \cup E) = least(P \cup M^-)$$

This, by (1), implies that M is a stable model of P. ■

PROOF OF PROPOSITION 3.11. Let Sem be a semantics for dynamic logic programs that complies with the refined extension principle, $\mathcal{P}$ a dynamic logic program, and $\mathcal{E}$ a sequence of sets of tautologies. Let M be an interpretation such that $M \in Sem(\mathcal{P} \cup \mathcal{E})$. Then:

$$\overline{M} = least(\rho(\mathcal{P} \cup \mathcal{E}) - Rej(\mathcal{P} \cup \mathcal{E}, M) \cup Assumption(\mathcal{P} \cup \mathcal{E}, M))$$

Splitting the set $\rho(\mathcal{P} \cup \mathcal{E}) - Rej(\mathcal{P} \cup \mathcal{E}, M)$ in two parts, namely $\rho(\mathcal{P}) - Rej(\mathcal{P} \cup \mathcal{E}, M)$ and $\rho(\mathcal{E}) - Rej(\mathcal{P} \cup \mathcal{E}, M)$ we obtain

$$\overline{M} = least(\rho(\mathcal{P}) - Rej(\mathcal{P} \cup \mathcal{E}, M) \cup Assumption(\mathcal{P} \cup \mathcal{E}, M) \cup \mathcal{E}')$$

where $\mathcal{E}' = \rho(\mathcal{E}) - Rej(\mathcal{P} \cup \mathcal{E}, M)$. Note that all the rules in $\mathcal{E}'$ are tautologies. Since the addition of tautologies does not change the least model (cf. Property 2) it follows that M is also the least model of

$$\rho(\mathcal{P}) - Rej(\mathcal{P} \cup \mathcal{E}, M) \cup Assumption(\mathcal{P} \cup \mathcal{E}, M)$$

Since Sem complies with the refined extension principle, from such principle it follows that $M \in Sem(\mathcal{P})$. ■

PROOF OF THEOREM 4.2 . Let $\mathcal{P}$ and $\mathcal{E}$ be the two dynamic logic programs $P_1, \ldots, P_n$ and $E_1, \ldots, E_n$ such that $\mathcal{P} \cup \mathcal{E}$ is a refined extension of $\mathcal{P}$, and M a refined stable model of $\mathcal{P} \cup \mathcal{E}$ i.e.

$$\overline{M} = least(\rho(\mathcal{P} \cup \mathcal{E}) - Rej(\mathcal{P} \cup \mathcal{E}, M) \cup Assumption(\mathcal{P} \cup \mathcal{E}, M))$$

We have to prove that M is also a refined stable model of $\mathcal{P}$ i.e.

$$\overline{M} = least(Generator^S(\mathcal{P}, M))$$

Since M is a refined stable model of $\mathcal{P} \cup \mathcal{E}$ then, by definition, $\overline{M}$ is the least model of the definite logic program: $\mathcal{G}en \cup \mathcal{R}es$ where $\mathcal{G}en$ is the program

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

and $\mathcal{R}es$ the program

$$\rho(\mathcal{E}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M)$$

From the hypothesis we know that $\mathcal{G}en \cup \mathcal{R}es$ is a syntactic extension of $\mathcal{G}en$. By the definition of syntactic extension we derive that

$$\overline{M} = least(\mathcal{G}en)$$

If a rule τ belongs to $Rej^S(\mathcal{P}, M)$, then it also belongs to $Rej^S(\mathcal{P} \cup \mathcal{E}, M)$, because there will be anyway a rule rejecting it, hence

$$Rej^S(\mathcal{P}, M) \subseteq Rej^S(\mathcal{P} \cup \mathcal{E}, M)$$

Furthermore, if *not* A belongs to $Def(\mathcal{P} \cup \mathcal{E}, M)$, there is no rule in $\mathcal{P}$ with head A and true body, then *not* $A \in Def(\mathcal{P}, M)$ and hence for any A, *not* $A \in Def(\mathcal{P} \cup \mathcal{E}, M)$ implies *not* $A \in Def(\mathcal{P}, M)$ i.e.

$$Def(\mathcal{P} \cup \mathcal{E}, M) \subseteq Def(\mathcal{P}, M)$$

It follows that

$$Def(\mathcal{P} \cup \mathcal{E}, M) \quad \subseteq \quad least(Generator^S(\mathcal{P}, M)) \tag{15}$$

Moreover, since

$$\rho(\mathcal{P}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M) \subseteq \rho(\mathcal{P}) - Rej^S(\mathcal{P}, M) \tag{16}$$

by definition of $\mathcal{G}en$ and the inclusions (15) and (16) we obtain

$$\mathcal{G}en \subseteq Generator^S(\mathcal{P}, M)$$

We will prove that $\overline{Generator^S(\mathcal{P}, M)} = \overline{\mathcal{G}en} \cup U \cup R$ where R and U are definite logic programs such that $\overline{\mathcal{G}en}$, R and U satisfy the hypothesis of Property 1. This, by Property 1, implies that:

$$least(Generator^S(\mathcal{P}, M)) = least(\mathcal{G}en) = \overline{M}$$

and proves the thesis.

Let us define R as the following set of rules:

$$R = \left\{\tau : \tau \notin \overline{\mathcal{G}en} \wedge H(\tau) \in \overline{M} \wedge \left(\tau \in \overline{Def(\mathcal{P}, M)} \vee \tau \in \overline{\rho(\mathcal{P}) - Rej^S(\mathcal{P}, M)}\right)\right\}$$

And U as the following set of rules:

$$U = \{\tau : \tau \notin \overline{\mathcal{G}en} \wedge \ \overline{H(\tau)} \notin \ \overline{M} \wedge \ \overline{\tau} \mid \tau \in \rho(\mathcal{P}) - Rej^S(\mathcal{P}, M)\}$$

By definition of $Generator^S(\mathcal{P}, M)$ it follows that

$$\overline{Generator^S(\mathcal{P}, M)} = \overline{\mathcal{G}en} \cup U \cup R$$

and, by definition of R it follows $H(R) \subseteq least(\mathcal{G}en)$. It remains to be proved that $\overline{M} \not\models B(\overline{\tau})$ for every $\overline{\tau} \in U$. Let L be the head of rule $\tau \in U$. Since, by definition of U, $H(\tau) \notin \overline{M}$, there are two possibilities:

- L is the positive literal A and $not\ A \in\ Def(\mathcal{P} \cup \mathcal{E}, M)$. Then there does not exist any rule in $\rho(\mathcal{P})$ whose head is A and whose body is supported by M, then $\overline{M} \not\models B(\overline{\tau})$.
- There exists a rule η in $\rho(\mathcal{P}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M)$ such that $H(\eta) = not\ L$ and $M \models B(\eta)$. Then, suppose by absurd that, $\overline{M} \models B(\overline{\tau})$ and hence $M \models B(\tau)$. Let i be the index such that $\tau \in P_i$. Then there exists P_j such that $\eta\ \in\ P_j$ and $i < j$, otherwise η would not belong to $\rho(\mathcal{P}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M)$. Then, τ is rejected by η and hence: $\tau\ \notin\ \rho(\mathcal{P}) - Rej^S(\mathcal{P}, M)$, against hypothesis.

We have proved that $\mathcal{G}en$, U and R satisfy the hypothesis of Property 1. Then, by such Property, we have that $least(Generator^S(\mathcal{P}, M)) = least(\mathcal{G}en) = \overline{M}$ ■

Proof of Theorem 4.3 . Let $\mathcal{P}$ be a dynamic logic program, $\mathcal{E}$ a sequence of sets of tautologies, and M a refined stable model of $\mathcal{P}$. We have to prove that M is also a refined stable model of $\mathcal{P} \cup \mathcal{E}$.

By definition, $\overline{M}$ is a refined stable model of $\mathcal{P} \cup \mathcal{E}$ iff

$$\overline{M} = least(Generator^S(\mathcal{P} \cup \mathcal{E}, M))$$

where

$$\begin{aligned} &Generator^S(\mathcal{P} \cup \mathcal{E}, M) = \\ &= \rho(\mathcal{P}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M)\ \ \cup Def(\mathcal{P} \cup \mathcal{E}, M) \cup \rho(\mathcal{E}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M) \end{aligned} \tag{17}$$

First, we find a simpler program G such that

$$least(Generator^S(\mathcal{P} \cup \mathcal{E}, M)) = least(G)$$

Since

$$\rho(\mathcal{E}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M)$$

is a tautological program, by Property 2 we obtain

$$\begin{aligned} &least(Generator^S(\mathcal{P} \cup \mathcal{E}, M)) = \\ &least\left(\rho(\mathcal{P}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M)\ \cup Def(\mathcal{P} \cup \mathcal{E}, M)\right) \end{aligned}$$

We will now prove that $Def(\mathcal{P} \cup \mathcal{E}, M) = Def(\mathcal{P}, M)$. Clearly

$$Def(\mathcal{P} \cup \mathcal{E}, M) \subseteq Def(\mathcal{P}, M)$$

Now suppose that $not\ A \in Def(\mathcal{P}, M)$. Then $A \notin M$. For each rule τ in $\rho(\mathcal{E})$ such that $H(\tau) = A$, since τ is a tautology, $A \in B(\tau)$ which implies $M \not\models B(\tau)$. Hence, it follows that $A \in Def(\mathcal{P} \cup \mathcal{E}, M)$. It follows that

$$Def(\mathcal{P} \cup \mathcal{E}, M) = Def(\mathcal{P}, M)$$

as stated. Hence we define the program G as

$$G = \rho(\mathcal{P}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M) \cup Def(\mathcal{P}, M)$$

By what was previously mentioned, it follows that:

$$least(Generator^S(\mathcal{P} \cup \mathcal{E}, M)) = least(G)$$

Hence, if we prove that $least(G) = \overline{M}$ we prove the thesis. By

$$\rho(\mathcal{P}) - Rej^S(\mathcal{P} \cup \mathcal{E}, M) \subseteq \rho(\mathcal{P}) - Rej^S(\mathcal{P}, M)$$

It follows

$$G \subseteq \rho(\mathcal{P}) - Rej^S(\mathcal{P}, M) \cup Def(\mathcal{P}, M)$$

Then clearly $least(G) \subseteq \overline{M}$.

It remains to prove the opposite inclusion. Suppose, by absurd, there exists a literal L such that $\overline{L} \in \overline{M}$ but $\overline{L} \notin least(G)$. This means there exists a rule $\tau \in Generator^S(\mathcal{P}, M)$ such that $\tau \notin G$ and $H(\tau) = L$. This means there exists a tautology η such that $H(\eta) = not\ L$ and $M \models B(\eta)$. The last condition implies, since η is a tautology, that $\overline{not\ L} \in \overline{M}$, or equivalently $\overline{L} \notin \overline{M}$ against hypothesis. Then $least(G) = \overline{M}$ and hence M is a refined stable model of $\mathcal{P} \cup \mathcal{E}$. ■

Proof of Theorem 5.3 . Let $\mathcal{P}$ be a dynamic logic program and M a refined stable model of $\mathcal{P}$. We have to prove that M is a dynamic stable model. Since M is a refined stable model of $\mathcal{P}$, it is such that:

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

We have to prove that

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

Following the outline of the proof of theorem 4.2 We will prove that

$$\overline{Generator^S(\mathcal{P}, M)} \cup R \cup U = \overline{Generator^D(\mathcal{P}, M)}$$

where R and U are definite logic programs such that $\overline{Generator^S(\mathcal{P}, M)}$, R and U satisfy the hypothesis of Property 1. This, by such Property, implies:

$$least(Generator^S(\mathcal{P}, M)) = \overline{M}$$

and prove the thesis. By the two definitions of of rejected rules we obtain

$$\rho(\mathcal{P}) - Rej^S(\mathcal{P}, M) \subseteq \rho(\mathcal{P}) - Rej^D(\mathcal{P}, M)$$

We define R in the following way

$$\{\overline{\tau} |\ \tau \notin \rho(\mathcal{P}) - Rej^S(\mathcal{P}, M)\ \wedge\ \tau \in \rho(\mathcal{P}) - Rej^D(\mathcal{P}, M) \wedge\ H(\tau) \in M\}$$

And U in the following way

$$\{\overline{\tau} |\ \tau \in \rho(\mathcal{P}) - Rej^D(\mathcal{P}, M)\ \wedge\ \tau \notin Generator^S(\mathcal{P}, M) \wedge\ \overline{H(\tau)} \notin \overline{M}\}$$

By definition of $Generator^S(\mathcal{P}, M)$ it follows

$$\overline{Generator^S(\mathcal{P}, M)} = \overline{Generator^D(\mathcal{P}, M)} \cup U \cup R$$

and, by definition of R it follows $H(R) \subseteq least(Generator^D(\mathcal{P}, M))$.

It remains to prove that $\overline{M} \not\models B(\overline{\tau})$ for any $\overline{\tau} \in U$. Let L be the head of τ. By definition of U it follows that $L \notin M$. There are two possibilities.

- L is the positive literal A and $not\ A \in Def(\mathcal{P}, M))$. Then it does not exists any rule in $\rho(\mathcal{P})$ whose head is A and whose body is supported by M, then $\overline{M} \not\models B(\tau)$.
- There exists a rule $\eta \in \rho(\mathcal{P}) - Rej^S(\mathcal{P}, M)$ such that $H(\eta) = not\ L$ and $M \models B(\eta)$. Suppose, by absurd, $\overline{M} \models B(\overline{\tau})$, which implies that $M \models B(\tau)$. Let i be the index such that $\tau \in P_i$. Then, there exists P_j such that $\eta \in P_j$ and $i < j$ otherwise η would not belong to $\rho(\mathcal{P}) - Rej^S(\mathcal{P}, M)$. Then, by definition, $\tau \notin \rho(\mathcal{P}) - Rej^D(\mathcal{P}, M)$, against hypothesis.

We proved that $\overline{Generator^S(\mathcal{P}, M)}$, U and R satisfy the hypothesis of Property 1. Hence we obtain

$$least(\overline{Generator^D(\mathcal{P}, M)}) = least(\overline{Generator^S(\mathcal{P}, M)} \cup U \cup R)$$

and so, $least(Generator^S(\mathcal{P}, M)) = least(Generator^D(\mathcal{P}, M))$. ∎

References

[1] Alferes, J. J., J. A. Leite, L. M. Pereira, H. Przymusinska, and T. C. Przymusinski, 'Dynamic logic programming', in A. Cohn, L. Schubert, and S. Shapiro, (eds.), *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)*, Morgan Kaufmann Publishers, 1998, pp. 98–111.

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

[4] Buccafurri, F., W. Faber, and N. Leone, 'Disjunctive logic programs with inheritance', in D. De Schreye, (ed.), *Proceedings of the 1999 International Conference on Logic Programming (ICLP-99)*, MIT Press, November 1999, pp. 79–93.

[5] Eiter, T., 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.

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

[7] Gelfond, M., and V. Lifschitz, 'The stable model semantics for logic programming', in R. Kowalski and K. A. Bowen, (eds.), *5th International Conference on Logic Programming*, MIT Press, 1988, pp. 1070–1080.

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

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

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

[11] Leite, J. A., J. J. Alferes, and L. M. Pereira, 'Multi-dimensional dynamic knowledge representation', in T. Eiter, M. Truszczynski, and W. Faber, (eds.), *Proceedings of the 6th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-01)*, volume 2173 of *LNAI*, Springer, 2001, pp. 365–378.

[12] Leite, J. A., and L. M. Pereira, 'Generalizing updates: From models to programs', in J. Dix, L. M. Pereira, and T. C. Przymusinski, (eds.), *Selected Extended Papers of the ILPS'97 3th International Workshop on Logic Programming and Knowledge Representation (LPKR-97)*, volume 1471 of *LNAI*, Springer Verlag, 1997, pp. 224–246.

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

[14] Leone, N., G. Pfeifer, W. Faber, F. Calimeri, T. Dell'Armi, T. Eiter, G. Gottlob, G. Ianni, G. Ielpa, S. Perri C. Koch, and A. Polleres, 'The DLV system', in *Procs. of JELIA'02*, volume 2424 of *LNAI*, Springer-Verlag, 2002.

[15] Levesque, H., F. Pirri, and R. Reiter, 'Foundations for the situation calculus', *Linkoping Electronic Articles in Computer and Information Science*, 3(18), 1998.

[16] Lifschitz, V., and T. Woo, 'Answer sets in general non-monotonic reasoning (preliminary report)', in B. Nebel, C. Rich, and W. Swartout, (eds.), *Proceedings of the 3th International Conference on Principles of Knowledge Representation and Reasoning (KR-92)*, Morgan-Kaufmann, 1992.

[17] McCarthy, J., and P. J. Hayes, 'Some philosophical problems from the standpoint of artificial intelligence', in B. Meltzer and D. Michie, (eds.), *Machine Intelligence 4*, Edinburgh University Press, 1969, pp. 463–502.

[18] Niemela, I., and P. Simons, 'Smodels: An implementation of the stable model and well-founded semantics for normal LP', in J. Dix, U. Furbach, and A. Nerode, (eds.), *Proceedings of the 4th International Conference on Logic Programing and Nonmonotonic Reasoning (LPNMR-97)*, volume 1265 of *LNAI*, Springer, 1997, pp. 420–429.

[19] Przymusinska, H., and T. Przymusinski, 'Weakly perfect model semantics for logic programs', in R. A. Kowalski and K. A. Bowen, (eds.), *Proceedings of the Fifth International Conference and Symposium on Logic Programming*, The MIT Press, 1988, pp. 1106–1120.

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

[21] Zhang, Y., and N. Y. Foo, 'Updating logic programs', in H. Prade, (ed.), *Proceedings of the 13th European Conference on Artificial Intelligence (ECAI-98)*, John Wiley & Sons, 1998, pp. 403–407.

José Júlio Alferes
CENTRIA
Universidade Nova de Lisboa
2829-516 Caparica,
Portugal
jja@di.fct.unl.pt

Federico Banti
CENTRIA
Universidade Nova de Lisboa
2829-516 Caparica,
Portugal
banti@di.fct.unl.pt

Antonio Brogi
Dipartimento di Informatica
Università di Pisa
Pisa, Italy
brogi@di.unipi.it

João Alexandre Leite
CENTRIA
Universidade Nova de Lisboa
2829-516 Caparica,
Portugal
jleite@di.fct.unl.pt