COMENIUS UNIVERSITY
FACULTY OF MATHEMATICS, PHYSICS AND INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS
BRATISLAVA, SLOVAKIA

Transformational Semantics and Implementation of Evolving Logic Programs

Master's Thesis

Martin Slota
author

João Alexandre Leite, MSc, PhD
doc. PhDr. Ján Šefránek, PhD
advisors

BRATISLAVA APRIL 2007

Transformational Semantics and Implementation of Evolving Logic Programs

Master's Thesis

Martin Slota

Comenius University
Faculty of Mathematics, Physics and Informatics
Department of Applied Informatics
Bratislava, Slovakia

Study Programme: 2508800 Informatics

João Alexandre Leite, MSc, PhD
doc. PhDr. Ján Šefránek, PhD

Bratislava, April 2007

By this I declare that I wrote this thesis by oneself, only with the help of the referenced literature, under the careful supervision of my thesis advisors.

Bratislava, April 2007 Martin Slota

Acknowledgements

A very big thanks goes to my advisors João Alexandre Leite and Ján Šefránek for their supervision and priceless help with this work.

I would also like to thank my parents and my brother for their gentle guidance, patience and support ever since I was born.

Last but not least, my thanks goes to Martin Baláž, Peter Klimo, Michal Malý and Jozef Šiška for their questions and ideas.

Abstract

Over the years, logic programming has proved to be a good and natural tool for expressing, querying and manipulating explicit knowledge in many areas of computer science. However, it is not so easy to use in dynamic environments. Evolving logic programs (**EVOLP**) are an elegant and powerful extension of logic programming suitable for multiagent systems, planning and other uses where information tends to change dynamically.

This work characterizes **EVOLP** by transforming it into an equivalent normal logic program. The proposed transformation is further examined and used to write a freely available, extensible and reusable implementation of **EVOLP** under the stable model semantics.

Keywords: Logic Programming, Stable Model Semantics, Evolving Logic Programs, Transformational Semantics, Implementation

Abstrakt

Preklad anglického keď bude jeho finálna verzia …

Kľúčové slová: preklad anglických kľúčových slov

Preface

In the last years, a lot of effort has been invested in finding a language suitable for specifying and programming the evolution of knowledge bases represented as logic programs. Such a language could be used to declaratively program intelligent agents and multiagent systems. Evolving logic programs (**EVOLP**) is one of the languages developed for this purpose and unlike its predecessors, it is just a simple, yet powerful extension of traditional logic programming.

The aim of this work is to examine the possibilities of implementing **EVOLP** under the stable model semantics. First, we focused on defining a sound and complete transformation that would produce an equivalent normal logic program for any given evolving logic program (a so-called transformational semantics). Then we used the transformation to implement propositional **EVOLP** and tried to face the problems with introducing variables into the language.

The result is an implementation of **EVOLP** with variable support. It has been designed with maintainablilty, extensibility, and reusability in mind. However, it is still not practically usable because it misses some important features, e.g. support for arithmetic predicates, strong negation and rule-type variables. In the last chapter we sketch some possibilities of implementing these features.

Bratislava, April 2007 Martin Slota

Contents

Chapter 1

Introduction and Motivation

1.1 Logic Programming and Intelligent Agents

Construction of intelligent agents is one of the main matters of artificial intelligence. Such agents should be capable of operating independently in a partially observable environment that may change unexpectedly. Therefore they need to be able to update their model of the world according to the changes that take place in them and around them.

Logic programming showed as a good tool for both symbolic knowledge representation and hypothetical reasoning. Much research in the last decade has been devoted to finding a good way of updating knowledge represented by logic programs. A sequence of logic programs where each program is an update of the preceding programs was called a Dynamic Logic Program (**DLP**) and finding a suitable semantics for **DLP**s became the first step to using logic programming for programming intelligent agents. Quite a number of semantics with different properties were introduced. We will only mention the Dynamic Stable Model semantics [1, 2, 3, 4] that was later improved and called Refined Dynamic Stable Models [5, 6, 7]. For a more comprehensive overview of semantics for **DLP**s see [8, 9].

Although dynamic logic programming provides a way of updating a logic program by another logic program, it still doesn't tell us how we should construct these programs. Update languages like LUPS [10, 11], EPI [12], KUL and KABUL [8] were developed for the purpose of incre-

mentally constructing a sequence of logic programs from a set of rules expressing when a rule should be added or deleted from the next program in the sequence. Evolving Logic Programs (**EVOLP**) [13] also comes from this line of work but while its predecessors were getting more and more complicated as more constructs were being added, **EVOLP** is a simple, yet powerful extension of traditional logic programming. Syntactically, evolving logic programs are just generalized logic programs. Semantically they allow for arbitrary updates of the program by adding new rules to it, both by self-updates and by updates from the environment (through events).

1.2 The Roadmap

We believe **EVOLP** is an interesting language with a neat idea behind it and as such it is worth implementing. An implementation running under the well-founded semantics has been available for quite some time [14]. But an implementation of the evolution stable model semantics from the papers about **EVOLP** appears only in [15, 16] and only for propositional programs.

The aim of this work is to examine the problems with implementing **EVOLP** and providing a freely available and reusable implementation of **EVOLP** under the evolution stable model semantics. The first steps led to defining the transformational semantics for **EVOLP**, i.e. a sound and complete transformation that turns an arbitrary evolving logic program into an equivalent normal logic program. The proposed transformation is then used to implement propositional **EVOLP** and practical problems with introducing variables are examined and partially resolved.

The remainder of this work is structured as follows:

Chapter 2 – Preliminaries: This Chapter presents the syntax and semantics of logic programs, dynamic logic programs and evolving logic programs. It only contains the definitions and theorems needed later in the work. We also take a look at the transformational semantics for **DLP**s [17] because our transformational semantics for **EVOLP** is based on it.

Chapter 3 – Transformational Semantics for EVOLP: Here we define the transormational semantics for **EVOLP** and prove that it is sound and complete w.r.t. the evolution stable model semantics. We also make a worst case approximation of the size of the transformed program.

Chapter 4 – Implementation of EVOLP: TODO

Chapter 5 – Conclusion and Future Work: In the last Chapter we sum up this work, describe the unresolved problems and sketch some ideas of dealing with them.

Chapter 2

Preliminaries

This Chapter contains a collection of definitions and theorems that will be used subsequently. In Sect. 2.1 we briefly introduce the syntax and semantics of generalized logic programs and formulate some related propositions and theorems.

Dynamic logic programs (**DLP**s) are presented in Sect. 2.2. Special attention is paid to the transformational semantics for **DLP**s because the transformational semantics for **EVOLP** proposed in Chap. 3 is based on it.

Section 2.3 contains the definition of evolving logic programs.

2.1 Logic Programs

This section contains some definitions and theorems from the wide area of logic programming. For a more thorough overview see [18, 8] or Wikipedia. We will start off by defining the syntax of logic programs.

2.1.1 Syntax

Definition 2.1 (Atoms and literals)**.** Let $\mathcal{U}$ be an arbitrary denumerable set of propositional atoms (a language). An *atom of* $\mathcal{U}$ is any $A \in \mathcal{U}$. A *default literal over* $\mathcal{U}$ is an atom of $\mathcal{U}$ preceded by a "**not**" representing default negation. A *literal over* $\mathcal{U}$ is either an atom of $\mathcal{U}$ or a default literal over $\mathcal{U}$.

The set of all literals[1] is denoted by $\mathcal{U}^*$, i.e. $\mathcal{U}^* = \mathcal{U} \cup \{\textbf{not}\ A \mid A \in \mathcal{U}\}$.

Let L be a literal. If L is a default literal $\textbf{not}\ A$, then $\textbf{not}\ L$ denotes the atom A. Similarily, if L is an atom A, then $\textbf{not}\ L$ denotes the default literal $\textbf{not}\ A$.

Definition 2.2 (Rules)**.** A *rule r over $\mathcal{U}$* is an ordered pair $(H(r), B(r))$ where $H(r)$ (dubbed the head of the rule) is a literal over $\mathcal{U}$ and $B(r)$ (dubbed the body of the rule) is a finite set of literals over $\mathcal{U}$. A rule $r = (L_0, \{L_1, L_2, \ldots, L_n\})$ is usually written as

$$L_0 \leftarrow L_1, L_2, \ldots, L_n. \tag{2.1}$$

We say a literal L appears in a rule (2.1) iff the set

$$\{L, \textbf{not}\ L\} \cap \{L_0, L_1, \ldots, L_n\}$$

is non-empty. Two rules r, r' are conflicting, denoted by $r \bowtie r'$, iff $H(r) = \textbf{not}\ H(r')$.

A *definite rule over $\mathcal{U}$* is a rule containing only atoms of $\mathcal{U}$. A *normal rule over $\mathcal{U}$* is a rule with an atom in its head.

Definition 2.3 (Generalized logic program)**.** A *generalized logic program over $\mathcal{U}$* is a set of rules over $\mathcal{U}$. We say a literal L appears in a generalized logic program P iff L appears in some rule of P.

Definition 2.4 (Normal logic program)**.** A *normal logic program over $\mathcal{U}$* is a set of normal rules over $\mathcal{U}$.

Definition 2.5 (Definite logic program)**.** A *definite logic program over $\mathcal{U}$* is a set of definite rules over $\mathcal{U}$.

Remark. Every definite logic program is also a normal logic program. Every normal logic program is also a generalized logic program.

[1] in the rest of the text, the words "over $\mathcal{U}$" will be dropped for the sake of readability where it is clear from the context which $\mathcal{U}$ we are talking about (just like here)

2.1.2 Semantics

First we will define a model-theoretic semantics of definite logic programs. Subsequently we will use this semantics to define the stable model semantics of generalized logic programs.

Definition 2.6 (Interpretation)**.** By an *interpretation of* $\mathcal{U}$ we mean any set of atoms $I \subseteq \mathcal{U}$. Given an interpretation I we define

$$I^- = \{\textbf{not}\ A \mid A \notin I\} \quad ,$$
$$I^* = I \cup I^- \quad .$$

An atom A is true in an interpretation I, denoted by $I \models A$, if $A \in I$, and false otherwise. A default literal **not** A is true in I, denoted by $I \models \textbf{not}\ A$, if $A \notin I$, and false otherwise. A set of literals B is true in I, denoted by $I \models B$, iff each literal in B is true in I.

Definition 2.7 (Model)**.** Interpretation M is a model of a generalized logic program P iff for every rule $r \in P$ the following condition holds: if $M \models B(r)$, then $M \models H(r)$.

Definition 2.8 (Minimal model)**.** A *minimal model of a generalized logic program* P is every model M of P such that no $I \subsetneq M$ is a model of P.

Theorem 2.9 (Least model of a definite logic program)**.** Let P be a definite logic program. Then P has a unique minimal model. This model is called the *least model of* P.

Remark. The least model is generaly considered to be a good semantics for definite logic programs because it minimizes the set of atoms inferred by the program and all the other models are supersets of the least model.

Now let's take a look at how we can compute the least model.

Definition 2.10 (Immediate consequence operator)**.** The *immediate consequence operator* is for every definite logic program P and every interpretation I defined as

$$T_P(I) = \{A \mid (\exists r \in P)(H(r) = A \wedge B(r) \subseteq I)\}$$

Proposition 2.11 (Monotonicity of the immediate consequence operator)**.** The immediate consequence operator is monotone, i.e. for every definite logic program P and all interpretations I_1, I_2 such that $I_1 \subseteq I_2$ it holds that $T_P(I_1) \subseteq T_P(I_2)$

Proof. Follows easily from the definition. □

Theorem 2.12. Let P be a definite logic program, $M_0 = \emptyset$ and $M_{i+1} = T_P(M_i)$ for every $i \geq 0$. Then

$$\bigcup_{i<\omega} M_i$$

is the least model of P[2].

Example 2.13. Let's take the set of atoms $\mathcal{U} = \{\text{tired}, \text{sleepy}, \text{hungry}, \text{happy}\}$ and construct the definite logic program P over $\mathcal{U}$:

$$P: \quad \text{sleepy} \leftarrow \text{tired}. \tag{2.2}$$

$$\text{tired} \leftarrow . \tag{2.3}$$

$$\text{happy} \leftarrow \text{sleepy}, \text{hungry}. \tag{2.4}$$

These rules can be interpreted as follows: Rule (2.2) says that if I'm tired, then I'm also sleepy. Rule (2.3) says I'm tired. Rule (2.4) says that if I'm sleepy and hungry, I'm happy (because usually I eat too much before going to sleep and then I don't sleep very well).

We can construct the least model of P according to Theorem 2.12:

$$\begin{aligned}
M_0 &= \emptyset \\
M_1 &= T_P(M_0) = \{A \mid (\exists r \in P)(H(r) = A \wedge B(r) \subseteq \emptyset\} = \\
&= \{\text{tired}\} \\
M_2 &= T_P(M_1) = \{A \mid (\exists r \in P)(H(r) = A \wedge B(r) \subseteq \{\text{tired}\}\} = \\
&= \{\text{tired}, \text{sleepy}\} \\
M_3 &= T_P(M_2) = \cdots = M_2
\end{aligned}$$

[2] ω is the first limit ordinal, for more details see [18]

and the least model of P is

$$M = \bigcup_{i<\omega} M_i = \emptyset \cup \{\text{tired}\} \cup \{\text{tired}, \text{sleepy}\} \cup \{\text{tired}, \text{sleepy}\} \cup \ldots =$$
$$= \{\text{tired}, \text{sleepy}\}$$

Definition 2.14. Let P be a generalized logic program over $\mathcal{U}$. By $least\,(P)$ we'll denote the least model of the definite logic program P over $\mathcal{U}^*$ (i.e. all default literals in P are treated as new atoms when computing the model).

Remark. The previous definition doesn't define the least model of a generalized logic program. Generalized logic programs do not always have a least model. They can have multiple minimal models and sometimes not all of them are intuitive. Stable models are those minimal models that also pass an extra condition which makes them "constructive". Their definition and an example follow.

Definition 2.15 (Stable model)**.** We say that an interpretation M is a *stable model* of a generalized logic program P iff

$$M^* = least\left(P \cup M^-\right)^{3}$$

Example 2.16. Let $\mathcal{U} = \{\text{write\_thesis}, \text{tired}\}$. We can construct the following generalized logic program over $\mathcal{U}$:

$$P: \quad \text{write\_thesis} \leftarrow \textbf{not}\ \text{tired}. \tag{2.5}$$

The interpretation of rule (2.5) is: If I have no evidence that I am tired, then I will continue writing the thesis. This program has 3 models, in particular

$$M_1 = \{\text{write\_thesis}\}$$
$$M_2 = \{\text{tired}\}$$
$$M_3 = \{\text{write\_thesis}, \text{tired}\}$$

[3] the default literals in M^- are interpreted as facts

Only M_1 and M_2 are minimal and the only stable model of P is M_1. It is also the most natural consequence of the program – we cannot infer tired and therefore we can use the rule (2.5) and infer $\mathrm{write\_thesis}$.

Proposition 2.17. Let P be a generalized logic program, M its stable model and A an atom. Then:

$$A \in M \iff (\exists r \in P)(H(r) = A \wedge M \models B(r))$$

Proof. From Definition 2.15 and Theorem 2.12 we have that

$$M^* = \bigcup_{i<\omega} M_i$$

where $M_0 = \emptyset$ and $M_{i+1} = T_{P \cup M^-}(M_i)$ for every $i \geq 0$.

Now let $A \in M$. Then $A \in M^*$ and therefore some $i \in \mathbb{N}$ exists such that $A \in M_{i+1}$. This means that a rule $r \in P$ exists such that $H(r) = A$ and $B(r) \subseteq M_i \subseteq M^*$. So $M \models B(r)$ and r is the rule we search for.

For the converse implication let's take some rule $r \in P$ such that $H(r) = A$ and $M \models B(r)$. This means that $B(r) \subseteq M^*$ and from the monotonicity of the immediate consequence operator we have that for some $i \in \mathbb{N}$ it must hold that $B(r) \subseteq M_i$. Therefore $A \in M_{i+1} \subseteq M^*$ and thus $A \in M$. □

2.2 Dynamic Logic Programming

Syntactically, a dynamic logic program is simply a sequence of generalized logic programs. Semantically, the rules in each program of the sequence are preferred over rules from preceding programs.

Now we will introduce the syntax and the refined dynamic stable model semantics for **DLP**s (as it is defined in [7]). It is an improved version of the dynamic stable model semantics that can be found in [8].

Definition 2.18 (Dynamic logic program)**.** A *dynamic logic program over* $\mathcal{U}$ (**DLP**) is a sequence of generalized logic programs over $\mathcal{U}$. Let $\mathcal{P} = (P_1, P_2, \ldots, P_n)$ be a **DLP**. We use $\rho(\mathcal{P})$ to denote the multiset of

all rules appearing in the programs $P_1, P_2, \ldots, P_n$ and $\mathcal{P}^i$ $(1 \leq i \leq n)$ to denote the i-th component of $\mathcal{P}$, i.e. P_i.

Definition 2.19 (Default assumptions)**.** Let $\mathcal{P}$ be a dynamic logic program and I an interpretation[4]. Then:

$$Def(\mathcal{P}, I) = \{\textbf{not}\, A \mid (\nexists r \in \rho(\mathcal{P}))(H(r) = A \wedge I \models B(r))\}$$

Definition 2.20 (Rejected rules)**.** Let $\mathcal{P}$ be a **DLP** of length n, I an interpretation and $j \in \{1, 2, \ldots, n\}$. Then:

$$Rej^j(\mathcal{P}, I) = \left\{ r \in \mathcal{P}^j \;\middle|\; (\exists k, r') \left(k \geq j \wedge r' \in \mathcal{P}^k \wedge r \bowtie r' \wedge I \models B(r') \right) \right\} \ ,$$
$$Rej(\mathcal{P}, I) = \bigcup_{i=1}^{n} Rej^i(\mathcal{P}, I) \ .$$

Definition 2.21 (Refined dynamic stable model, [7])**.** Let $\mathcal{P}$ be a **DLP** and M an interpretation. M is a *(refined) dynamic stable model*[5] $\mathcal{P}$ iff

$$M^* = least\left([\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M)] \cup Def(\mathcal{P}, M)\right) \ .$$

Remark. The defined semantics is a generalization of the stable model semantics, i.e. the stable models of a generalized logic program P are the same as the refined dynamic stable models of the **DLP** $\mathcal{P} = (P)$. We can also see an analogy with the definition of stable models: $\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M)$ plays the role of P and $Def(\mathcal{P}, M)$ the role of M^-.

Example 2.22. Consider the following two generalized logic programs:

$$P_1: \quad \text{tired} \leftarrow . \tag{2.6}$$
$$\text{drink\_coffee} \leftarrow \text{tired}. \tag{2.7}$$
$$\text{write\_thesis} \leftarrow \textbf{not}\, \text{tired}. \tag{2.8}$$
$$P_2: \quad \textbf{not}\, \text{tired} \leftarrow . \tag{2.9}$$

[4] here the words "over $\mathcal{U}$" and "of $\mathcal{U}$" are dropped again ...

[5] in the remainder of this work we will always work with the refined dynamic stable model semantics but for the sake of readability we will drop the word "refined" where possible

Rule (2.6) states that I'm tired. The other two rules in P_1 define what I will do depending on whether I'm tired or not. The rule (2.7) in P_2 states I'm not tired (any more).

P_1 has exactly one stable model: $M_1 = \{\text{tired}, \text{drink\_coffee}\}$. The dynamic logic program $\mathcal{P} = (P_1, P_2)$ has exactly one dynamic stable model: $M_2 = \{\text{write\_thesis}\}$. To verify that M_2 is really a dynamic stable model of $\mathcal{P}$ we need to check the following (according to Definitions 2.19, 2.20 and 2.21):

$$
\begin{aligned}
Def(\mathcal{P}, M_2) &= \{\textbf{not}\ \text{drink\_coffee}\} \\
Rej(\mathcal{P}, M_2) &= \{tired \leftarrow .\} \\
M_2^* &= \{\text{write\_thesis}, \textbf{not}\ \text{tired}, \textbf{not}\ \text{drink\_coffee}\} = \\
&= least\left(\left\{\begin{array}{l} \text{drink\_coffee} \leftarrow \text{tired}. \\ \text{write\_thesis} \leftarrow \textbf{not}\ \text{tired}. \\ \quad \textbf{not}\ \text{tired} \leftarrow . \end{array}\right\} \cup \{\textbf{not}\ \text{drink\_coffee}\}\right)
\end{aligned}
$$

2.2.1 Transformational Semantics

The transformational semantics for **EVOLP** that we will define in Chap. 3 is based on the transformational semantics for **DLP**s defined in [17]. On an example we will show how the transformation works.

Example 2.23. Let's take a **DLP** $\mathcal{P} = (P_1, P_2)$ where P_1 and P_2 are defined in Example 2.22. If we use the transformation from [17], we will get the following normal logic program:

$$
\begin{aligned}
P: \quad & \text{tired}^- \leftarrow \textbf{not}\ rej(0, \text{tired}^-). && (2.10)\\
& \text{drink\_coffee}^- \leftarrow \textbf{not}\ rej(0, \text{drink\_coffee}^-). && (2.11)\\
& \text{write\_thesis}^- \leftarrow \textbf{not}\ rej(0, \text{write\_thesis}^-). && (2.12)\\
& \text{tired} \leftarrow \textbf{not}\ rej(1, \text{tired}). && (2.13)\\
& \text{drink\_coffee} \leftarrow \text{tired}, \textbf{not}\ rej(1, \text{drink\_coffee}). && (2.14)\\
& \text{write\_thesis} \leftarrow \text{tired}^-, \textbf{not}\ rej(1, \text{write\_thesis}). && (2.15)\\
& \text{tired}^- \leftarrow \textbf{not}\ rej(2, \text{tired}^-). && (2.16)
\end{aligned}
$$

$$rej(0, \text{tired}^-) \leftarrow . \tag{2.17}$$

$$rej(0, \text{drink\_coffee}^-) \leftarrow \text{tired}. \tag{2.18}$$

$$rej(0, \text{write\_thesis}^-) \leftarrow \text{tired}^-. \tag{2.19}$$

$$rej(1, \text{tired}) \leftarrow . \tag{2.20}$$

$$rej(0, \text{tired}^-) \leftarrow rej(2, \text{tired}^-). \tag{2.21}$$

So what happened with the input program? The first big change is that all default literals were turned into new atoms. The reason is that in dynamic logic programming the set $Def(\mathcal{P}, M)$ may be smaller than M^- (its counterpart in the stable model semantics) so we have to treat the default literals differently. So in order to simulate the set of defaults, we add the rules (2.10) to (2.12). The next four rules (2.13) to (2.16) are just rewritten rules of the original programs. Now you must be asking: What are those "**not** $rej(\dots)$" literals good for? They are employed as guards that either allow or disallow the use of each rule. And the remaining five rules (2.17) to (2.21) infer the correct $rej(\dots)$ literals, also based on the original rules.

The program P has a unique stable model:

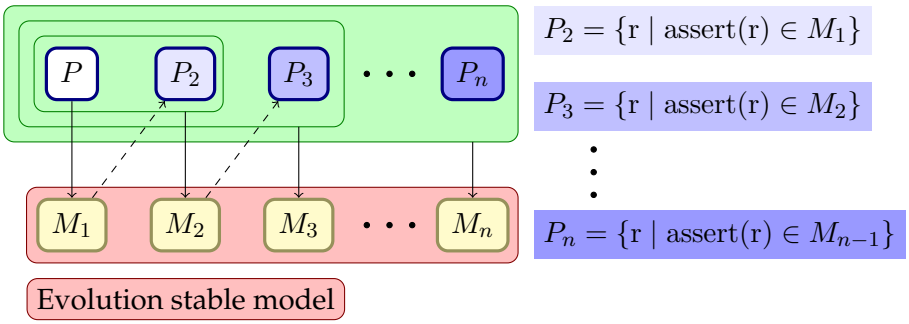
$$M = \{\text{write\_thesis}, \text{tired}^-, \text{drink\_coffee}^-, \\ rej(1, \text{tired}), rej(1, \text{tired}^-), rej(1, \text{write\_thesis}^-)\}$$

M directly corresponds to the dynamic stable model $M' = \{\text{write\_thesis}\}$ of the dynamic logic program $\mathcal{P}$.

2.3 Evolving Logic Programs

Evolving logic programs (**EVOLP**) can be seen as an extension of generalized logic programs. They can contain rules that add a new rule to the program in case they are fired. The newly added rule can also be of this type and so some rules can be added at the beginning and some only after the first set has been added. We can say the program evolves in steps. These steps are called evolution steps. In the semantics, the new rules after each evolution step are collected in a separate program and dynamic logic

Figure 2.1: Semantics of **EVOLP** (without events)



programming is used to prefer the rules that were added later.

In order to be useable in dynamic environments, there must also be some way of adding information coming from outside of the program. Otherwise an agent written in **EVOLP** would be unable to receive information from its sensors or to communicate with other agents. Therefore, after each evolution step an evolving logic program called event is added to the set of newly added rules. The event can contain an arbitrary set rules that may even change or completely reprogram the behaviour of the agent.

Now we will formally define the syntax and semantics of **EVOLP** as it is defined in [13]. The semantics (without events excluded) is also illustrated in Fig. 2.1.

Definition 2.24 (Extended language)**.** Let $\mathcal{U}$ be a set of propositional atoms (not containing the predicate $assert/1$). The *extended language* $\mathcal{U}_{assert}$ is a minimal set of propositional atoms such that:

- $\mathcal{U} \subseteq \mathcal{U}_{assert}$
- If r is a rule over $\mathcal{U}_{assert}$, then $assert(r) \in \mathcal{U}_{assert}$.

Definition 2.25 (Evolving logic program)**.** An *evolving logic program over* $\mathcal{U}$ is a (possibly infinite) set of rules over $\mathcal{U}_{assert}$.

Definition 2.26 (Event sequence)**.** An *event sequence over* $\mathcal{U}$ is a sequence of evolving programs over $\mathcal{U}$.

Definition 2.27 (Evolution interpretation and its evolution trace)**.** An *evolution interpretation of length* n of an evolving program P over $\mathcal{U}$ is a finite sequence $\mathcal{I} = (I_1, I_2, \ldots, I_n)$ of interpretations of $\mathcal{U}_{assert}$. The *evolution trace* associated with an evolution interpretation $\mathcal{I}$ of P is the sequence of programs $(P_1, P_2, \ldots, P_n)$ where

$$P_1 = P \text{ and } P_i = \{r \mid assert(r) \in I_{i-1}\} \text{ for all } i \in \{2, 3, \ldots, n\}$$

Definition 2.28 (Evolution stable model given an event sequence)**.** An evolution interpretation $\mathcal{M} = (M_1, M_2, \ldots, M_n)$ with evolution trace $(P_1, P_2, \ldots, P_n)$ is an *evolution stable model of an evolving program P given an event sequence* $(E_1, E_2, \ldots, E_n)$ iff for every $i \in \{1, 2, \ldots, n\}$ M_i is a dynamic stable model of $(P_1, P_2, \ldots, P_{i-1}, P_i \cup E_i)$.

Example 2.29. Consider the following evolving logic program:

$$P: \quad \text{write\_thesis} \leftarrow \textbf{not}\ \text{tired}. \tag{2.22}$$

$$\text{drink\_coffee} \leftarrow \text{tired}, \textbf{not}\ \text{no\_coffee}. \tag{2.23}$$

$$\text{make\_coffee} \leftarrow \text{tired}, \text{no\_coffee}. \tag{2.24}$$

$$assert(\text{tired} \leftarrow .) \leftarrow \text{write\_thesis}. \tag{2.25}$$

$$assert(\textbf{not}\ \text{tired} \leftarrow .) \leftarrow \text{drink\_coffee}. \tag{2.26}$$

P could be a program of a simple agent who is trying to write a thesis. The agent can do 3 things: write the thesis, drink coffee or make coffee. She also relies on a sensor that sends the fact

$$\text{no\_coffee} \leftarrow .$$

as an event in case the agent runs out of coffee.

The meaning of the rules is as follows: Rule (2.22) says the agent is writing the thesis as long as it's not tired. Rules (2.23) and (2.24) say what she will do when she's tired – depending on whether she has coffee she either drinks it or makes some more. Rules (2.25) and (2.26) specify whether the agent will be tired in the next evolution step. If she was writing the

thesis, she will get tired. In case she was drinking coffee, the tiredness will wear off. If she was making coffee, no change will take place.

The following table shows the evolution of the program P (given the sequence of events that are also present in the table):

Time	Program	Event	Model
1	P	$\emptyset$	$\{\text{write\_thesis},$ $\text{assert}(\text{tired} \leftarrow .)\}$
2	$\{\text{tired} \leftarrow .\}$	$\emptyset$	$\{\text{tired}, \text{drink\_coffee},$ $\text{assert}(\textbf{not}\ \text{tired} \leftarrow .)\}$
3	$\{\textbf{not}\ \text{tired} \leftarrow .\}$	$\{\text{no\_coffee} \leftarrow .\}$	$\{\text{no\_coffee}, \text{write\_thesis},$ $\text{assert}(\text{tired} \leftarrow .)\}$
4	$\{\text{tired} \leftarrow .\}$	$\{\text{no\_coffee} \leftarrow .\}$	$\{\text{tired}, \text{no\_coffee},$ $\text{make\_coffee}\}$
5	$\emptyset$	$\emptyset$	$\{\text{tired}, \text{drink\_coffee},$ $\text{assert}(\textbf{not}\ \text{tired} \leftarrow .)\}$
6	$\{\textbf{not}\ \text{tired} \leftarrow .\}$	$\emptyset$	$\{\text{write\_thesis},$ $\text{assert}(\text{tired} \leftarrow .)\}$

We start off with P and an empty event and compute the the first model. It says the agent is writing the thesis and in the next step she should get tired. We infer the second program from the model, add another empty event and compute the second model. Now the agent is tired and drinks coffee. In the next step the sensor starts complaining that there's no more coffee but the agent doesn't really care. She's not tired so she's writing the thesis. In the fourth step she makes coffee and which makes the sensor stop complaining. In the fifth step the agent drinks coffee again in order to continue writing the thesis in the sixth step ...

Chapter 3

Transformational Semantics for EVOLP

3.1 Definition of the Transformation

We will define a transformation which turns an evolving logic program P together with an event sequence $\mathcal{E}$ of length n into a normal logic program $P_{\mathcal{E}}$ over an extended language. We will prove later that the stable models of this program are in one-to-one correspondence with evolution stable models of P given $\mathcal{E}$.

The transformation is essentially a multiple parallel usage of a similar transformation for **DLP**s introduced in [17]. The extended language of the transformed program will be

$$\begin{aligned}\mathcal{U}_{trans} \quad = \quad & \{A^j, A^j_{neg} \mid A \in \mathcal{U}_{assert} \wedge 1 \leq j \leq n\} \cup \\ & \cup \{rej(A^j, i), rej(A^j_{neg}, i) \mid A \in \mathcal{U}_{assert} \wedge 1 \leq j \leq n \wedge 0 \leq i \leq j\} \cup \{u\}\end{aligned}$$

Atoms of the form A^j and A^j_{neg} in the extended language allow us to compress the whole evolution interpretation (consisting of n interpretations of $\mathcal{U}_{assert}$) into just one interpretation of $\mathcal{U}_{trans}$. Atoms of the form $rej(A^j, i)$ and $rej(A^j_{neg}, i)$ are needed for rule rejection simulation. The atom u will serve to formulate constraints needed to eliminate some unwanted models of $P_{\mathcal{E}}$. More explanation will come after the transformation is defined.

To simplify notation in the transformation and proofs of its soundness and completeness, we'll use the following conventions: Let L be a literal over $\mathcal{U}_{assert}$, B a set of literals over $\mathcal{U}_{assert}$ and j a natural number. Then:

- if L is an atom A, then L^j is A^j,
- if L is a default literal **not** A, then L^j is A^j_{neg},
- $B^j = \{L^j \mid L \in B\}$
- $(L^j_{neg})_{neg} = L^j$

Definition 3.1. Let P be an evolving logic program and $\mathcal{E} = (E_1, E_2, \ldots, E_n)$ an event sequence. By a transformational equivalent of P given $\mathcal{E}$ we mean the normal logic program $P_{\mathcal{E}} = P^1_{\mathcal{E}} \cup P^2_{\mathcal{E}} \cup \ldots \cup P^n_{\mathcal{E}}$ over $\mathcal{U}_{trans}$, where each $P^j_{\mathcal{E}}$ consists of these six groups of rules:

1. **Rewritten program rules.** For every rule $(L \leftarrow body.) \in P$ it contains the rule
$$L^j \leftarrow body^j, \mathbf{not}\, rej(L^j, 1).$$

2. **Rewritten event rules.** For every rule $(L \leftarrow body.) \in E_j$ it contains the rule
$$L^j \leftarrow body^j, \mathbf{not}\, rej(L^j, j).$$

3. **Assertion rules.** For every rule $r = (L \leftarrow body.)$ over $\mathcal{U}_{assert}$ and all i, $1 < i \leq j$, such that $(assert(r))^{i-1}$ is in the head of some rule of $P^{i-1}_{\mathcal{E}}$ it contains the rule
$$L^j \leftarrow body^j, (assert(r))^{i-1}, \mathbf{not}\, rej(L^j, i).$$

4. **Default assumptions.** For every atom $A \in \mathcal{U}_{assert}$ such that A^j or A^j_{neg} appears in some rule of $P^j_{\mathcal{E}}$ (from the previous groups of rules) it also contains the rule
$$A^j_{neg} \leftarrow \mathbf{not}\, rej(A^j_{neg}, 0).$$

5. **Rejection rules.** For every rule of $P_{\mathcal{E}}^j$ of the form

$$L^j \leftarrow body, \mathbf{not}\, rej(L^j, i).^{1} \tag{3.1}$$

it also contains the rules

$$rej(L^j_{neg}, p) \quad \leftarrow body. \tag{3.2}$$
$$rej(L^j, q) \quad \leftarrow rej(L^j, i). \tag{3.3}$$

where:

(a) $p \leq i$ is the largest index such that $P_{\mathcal{E}}^j$ contains a rule with L^j_{neg} in its head and $\mathbf{not}\, rej(L^j_{neg}, p)$ in its body. If no such p exists, then the rule (3.2) is not in $P_{\mathcal{E}}^j$.

(b) $q < i$ is the largest index such that $P_{\mathcal{E}}^j$ contains a rule with L^j in its head and $\mathbf{not}\, rej(L^j, q)$ in its body. If no such q exists, then the rule (3.3) is not in $P_{\mathcal{E}}^j$.

6. **Totality constraints.** For all $i \in \{1, 2, \ldots, j\}$ and every atom $A \in \mathcal{U}_{assert}$ such that $P_{\mathcal{E}}^j$ contains rules of the form

$$A^j \quad \leftarrow body_+, \mathbf{not}\, rej(A^j, i).$$
$$A^j_{neg} \quad \leftarrow body_-, \mathbf{not}\, rej(A^j_{neg}, i).$$

it also contains the constraint

$$u \leftarrow \mathbf{not}\, u, \mathbf{not}\, A^j, \mathbf{not}\, A^j_{neg}.$$

Each $P_{\mathcal{E}}^j$ contains rules for simulating the computation of dynamic stable models of the "j-th **DLP**" from the definition of evolution stable model, i.e. the **DLP** $(P_1, P_2, \ldots, P_{j-1}, P_j \cup E_j)$. The first two groups of rules contain rules originating from P and E_j, assertion rules consist of all rules that could possibly be asserted until the j-th evolution step and default

[1] The set *body* contains literals from the original body of the rule in translated form and in case it is an assertion rule it also contains a literal of the form $(assert(r))^{i-1}$ – later we will call this literal the assertion guard of the rule.

assumptions contain the rules for simulating the necessary part of the set of defaults. Heads of rules from the first 4 groups and also their former bodies are j-indexed – this ensures that rules of $P_{\mathcal{E}}^j$ work with a new set of atoms, untouched by rules in $P_{\mathcal{E}}^1 \cup P_{\mathcal{E}}^2 \cup \ldots \cup P_{\mathcal{E}}^{j-1}$. Each of these rules also contains a default literal of the form **not** $rej(L^j, i)$ in its body – we will call this literal the *rejection guard* of the rule and i the level of the rule. It provides a means of rejecting the rule by a higher level rule, similarily as in the set of rejected rules. Furthermore, each assertion rule contains an atom of the form $(assert(r))^{i-1}$ in its body. This is its *assertion guard* and it assures the rule is only used in case it was actually asserted before. Assertion guards are also the only connection between rules of $P_{\mathcal{E}}^j$ and rules in $P_{\mathcal{E}}^1 \cup P_{\mathcal{E}}^2 \cup \ldots \cup P_{\mathcal{E}}^{j-1}$. A body of a rewritten program rule, rewritten event rule or an assertion rule without the assertion and rejection guards is called its *guardless body*.

Rejection rules are responsible for inferring the correct $rej(L^j, i)$ atoms. The first kind of rules introduces the rejection of the next less or equally preferred rule with a conflicting literal in its head. The second kind of rules takes care of propagating the rejection to even less preferred rules with the same head.

Totality constraints are important in the case that equally preferred rules reject each other and no rule with higher priority resolves their conflict. An interpretation causing such situation is not a dynamic stable model (more details can be found in [7]) and totality constraints are needed to eliminate the superfluous stable models of $P_{\mathcal{E}}$ originating from such situations.

The following two sections prove that the defined transformation is sound and complete. Now we will introduce one more convention which will be used throughout these sections to make some formulations nicer:

Definition 3.2. We will say a literal L over $\mathcal{U}_{assert}$ *trans-appears* in $P_{\mathcal{E}}$ iff L^j or L_{neg}^j appears in $P_{\mathcal{E}}$.

3.2 Soundness of the Transformation

In this section we will prove that the defined transformation is sound, i.e. every stable model of the transformed program has a corresponding evolution stable model. Therefore we will assume P is an evolving logic program, $\mathcal{E} = (E_1, E_2, \ldots, E_n)$ is an event sequence, N is a stable model of $P_{\mathcal{E}}$,

$$M_i = \{A \in \mathcal{U}_{assert} \mid A^i \in N\} \text{ for all } i \in \{1, 2, \ldots, n\} \tag{3.4}$$

and $(P_1, P_2, \ldots, P_n)$ is the evolution trace associated to the evolution interpretation $(M_1, M_2, \ldots, M_n)$, i.e.

$$P_1 = P \text{ and } P_i = \{r \mid assert(r) \in M_{i-1}\} \text{ for all } i \in \{2, 3, \ldots, n\} \tag{3.5}$$

Using this notation, formulation of the soundness property boils down to: $(M_1, M_2, \ldots, M_n)$ is an evolution stable model of P given $\mathcal{E}$. According to the definition of evolution stable model this holds iff each M_i is a dynamic stable model of $(P_1, P_2, \ldots, P_{i-1}, P_i \cup E_i)$. Hence we choose one arbitrary but fixed $j \in \{1, 2, \ldots, n\}$ and prove that M_j is a dynamic stable model of $\mathcal{P} = (P_1, P_2, \ldots, P_{j-1}, P_j \cup E_j)$ and the property will follow.

We will need a number of auxiliary propositions to prove this. To make their formulation (and also the formulation of their proofs) simpler and more comprehensible, we will use the notation introduced before and also these common assumptions:

- $j \in \{1, 2, \ldots, n\}$
- *Rewritten rules* are all rewritten program rules, all rewritten event rules and assertion rules such that their assertion guard is true in N.
- A rewritten rule is *unrejected* iff its rejection guard is true in N.
- *Level of a rule* $r \in \rho(\mathcal{P})$ is i iff $r \in \mathcal{P}^i$.
- As N is a stable model of $P_{\mathcal{E}}$ we can use Definition 2.15 and Theorem 2.12 to obtain:

$$N^* = \bigcup_{i<\omega} N_i$$

where $N_0 = \emptyset$ and $N_{i+1} = T_{P_{\mathcal{E}} \cup N^-}(N_i)$ for all $i \geq 0$.

- We can use Theorem 2.12 once again:

$$least\,(\mathcal{U}^*_{assert})\,[\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M_j)] \cup Def(\mathcal{P}, M_j) = R = \bigcup_{i<\omega} R_i$$

where $R_0 = \emptyset$ and $R_{i+1} = T_{[\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M_j)] \cup Def(\mathcal{P}, M_j)}(R_i)$ for all $i \geq 0$.

According to the definition of dynamic stable model, M_j is a dynamic stable model of $\mathcal{P}$ iff $M^*_j = R$. All the rest of the proof aims at prooving this equality. First we show some properties of the stable model N and introduce some relations between N and M_j.

Lemma 3.3. Let L be a literal over $\mathcal{U}_{assert}$ and $k \in \{0, 1, \ldots, j\}$. Then $rej(L^j, k) \in N$ holds iff some $i \in \{k, k+1, \ldots, j\}$ exists such that $P_{\mathcal{E}}$ contains rules of the form

$$\begin{aligned} L^j &\leftarrow body_+, \mathbf{not}\, rej(L^j, k). \\ L^j_{neg} &\leftarrow body_-, \mathbf{not}\, rej(L^j_{neg}, i).^{2} \end{aligned}$$

and $N \models body_-$.

Proof. First let's assume $rej(L^j, k) \in N$. Then from Proposition 2.17 it follows that $P_{\mathcal{E}}$ contains some rule

$$r = (rej(L^j, k) \leftarrow body.)$$

such that $N \models body$. This must be one of the two kinds of rejection rules:

1. If r is of the form (3.2), then a look at the definition of rejection rules tells us that the rules we search for must also be in $P_{\mathcal{E}}$.

2. If r is of the form (3.3), then $body$ is of the form $rej(L^j, k_1)$ where $k < k_1 \leq j$. As $body$ is true in N, we can use Proposition 2.17 again to find a rule of $P_{\mathcal{E}}$ of the form

$$rej(L^j, k_1) \leftarrow body_1.$$

such that $N \models body_1$. Two cases are possible again, in the first the proof ends and in the second we get an index k_2 such that $k_1 < k_2 \leq j$ and $rej(L^j, k_2) \in N$. If the second case would occur forever, we would get an infinite increasing bounded sequence of natural numbers $k < k_1 < k_2 < \ldots \leq j$, which is not possible. Hence after a finite number of iterations the first case must occur and the proof ends.

Now to the converse implication. Let $k = k_1 < k_2 < \ldots < k_s \leq i$ be all indices such that $P_{\mathcal{E}}$ contains a rule of the form

$$L^j \leftarrow body_t, \mathbf{not}\, rej(L^j, k_t).$$

where $t \in \{1, 2, \ldots, s\}$. From the definition of rejection rules (Definition 3.1) we have that $P_{\mathcal{E}}$ contains the rule

$$rej(L^j, k_s) \leftarrow body_-.$$

and also the rules

$$rej(L_j, k_t) \leftarrow rej(L_j, k_{t+1}).$$

for all $t \in \{1, 2, \ldots, s-1\}$. The claim now follows by s times applying Proposition 2.17. □

Lemma 3.4. For every atom $A \in \mathcal{U}_{assert}$ such that it trans-appears in $P_{\mathcal{E}}$ the following holds:

$$A^j \in N \iff A^j_{neg} \notin N$$

Proof. First let's assume $A^j \in N$. Proposition 2.17 implies that $P_{\mathcal{E}}$ contains some rule of the form

$$A^j \leftarrow body_+, \mathbf{not}\, rej(A^j, i).$$

such that $N \models body_+$ and $rej(A^j, i) \notin N$. If r is any rule of $P_{\mathcal{E}}$ with A^j_{neg} in its head, then it must be of the form

$$A^j_{neg} \leftarrow body_-, \mathbf{not}\, rej(A^j_{neg}, k).$$

We will consider two cases:

a) If $k \leq i$, then from Lemma 3.3 we have that $rej(A^j_{neg}, k) \in N$.

b) If $k > i$, then, as $rej(A^j, i) \notin N$, we can use Lemma 3.3 to get $N \not\models body\_$.

Both cases imply that the body of r is false in N. Hence there is no rule of $P_{\mathcal{E}}$ with A^j_{neg} in its head and a body true in N and from Proposition 2.17 it follows that $A^j_{neg} \notin N$.

We will prove the converse implication by contradiction. Suppose $A^j \notin N$ and at the same time $A^j_{neg} \notin N$. Let s be the highest index such that $P_{\mathcal{E}}$ contains a rule

$$A^j \leftarrow body_s, \mathbf{not}\, rej(A^j, s).$$

such that $N \models body_s$. If no such rule is in $P_{\mathcal{E}}$, let $s = -1$. Similarily let t be the highest index such that $P_{\mathcal{E}}$ contains a rule

$$A^j_{neg} \leftarrow body_t, \mathbf{not}\, rej(A^j_{neg}, t).$$

such that $N \models body_t$. As A trans-appears in $P_{\mathcal{E}}$, there is always at least one such rule between the default assumptions so t is defined in all cases and $t \geq 0$. Let's consider two situations again:

a) $s \geq t$: If $rej(A^j, s) \notin N$, then A^j would be (by Proposition 2.17) in N. So $rej(A^j, s) \in N$ and Lemma 3.3 implies that $P_{\mathcal{E}}$ contains a rule

$$A^j_{neg} \leftarrow body_u, \mathbf{not}\, rej(A^j_{neg}, u).$$

such that $N \models body_u$ and $u \geq s$. But $u \leq t \leq s$ must also hold (according to the way t was constructed), so $u = s$. But then it can't be that $A^j, A^j_{neg} \notin N$, because $P_{\mathcal{E}}$ contains a totality constraint forbidding this – a contradiction with the assumptions.

b) $s < t$: If $rej(A^j_{neg}, t) \notin N$, then A^j_{neg} would be (by Proposition 2.17) in N. So $rej(A^j_{neg}, t) \in N$ and from Lemma 3.3 we get that $P_{\mathcal{E}}$ contains a rule

$$A^j \leftarrow body_v, \mathbf{not}\, rej(A^j, v).$$

such that $N \models body_v$ and $v \geq t$. But $v \leq s < t$ must also hold (according to the way s was constructed), which is a contradiction. □

Lemma 3.5. Let B be a set of literals over $\mathcal{U}_{assert}$ which trans-appear in $P_{\mathcal{E}}$. Then:

$$M_j \models B \iff N \models B^j$$

Proof. Let $L \in B$. If L is an atom A, then

$$M_j \models L \iff A \in M_j \overset{(3.4)}{\iff} A^j \in N \iff N \models L^j$$

If L is a default literal **not** A, then

$$M_j \models L \iff A \notin M_j \overset{(3.4)}{\iff} A^j \notin N \overset{Lemma\ 3.4}{\iff} A^j_{neg} \in N \iff N \models L^j \quad \square$$

Lemma 3.6. $P_{\mathcal{E}}$ contains a rewritten rule of level i with L^j in its head and a guardless body $body^j$ iff

$$(L \leftarrow body.) \in \mathcal{P}^i$$

Proof. Let $P_{\mathcal{E}}$ contain a rewritten rule r of level i with L^j in its head and a guardless body $body^j$. Let $r' = (L \leftarrow body.)$. We will consider 2 cases:

1. If r is a rewritten program rule or a rewritten event rule, then $i \in \{1, j\}$ and $\mathcal{P}^i$ contains the rule r' by the definition of $P_{\mathcal{E}}$.

2. If r is an assertion rule, then $i \in \{2, 3, \ldots, j\}$ and its assertion guard $(assert(r'))^{i-1}$ is true in N. Therefore $assert(r') \in M_{i-1}$ and hence $r' \in \mathcal{P}^i$.

For the converse implication let $r = (L \leftarrow body.) \in \mathcal{P}^i$ for some $i \in \{1, 2, \ldots, j\}$. We will consider two cases again:

1. If $r \in P$ or $r \in E_j$, then $P_{\mathcal{E}}$ contains a rewritten rule of level i with L^j in its head and the guardless body $body^j$ by the definition.

2. If $r \in P_i$ and $i > 1$, then $assert(r) \in M_{i-1}$ and therefore $(assert(r))^{i-1} \in N$. Hence $P_{\mathcal{E}}$ must contain a rule with $(assert(r'))^{i-1}$ in its head

(Proposition 2.17) and therefore it must also contain the assertion rule

$$L^j \leftarrow body^j, (assert(r))^{i-1}, \mathbf{not}\, rej(L^j, i). \qquad \square$$

Lemma 3.7. $P_{\mathcal{E}}$ contains an unrejected rewritten rule with L^j in its head and a guardless body $body^j$ iff

$$(L \leftarrow body.) \in \rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M_j)$$

Proof. Let $P_{\mathcal{E}}$ contain an unrejected rewritten rule r of level i with L^j in its head and a guardless body $body^j$. Then by Lemma 3.6 we have

$$r' = (L \leftarrow body.) \in \mathcal{P}^i$$

We will continue by contradiction – let's assume $r' \in Rej^i(\mathcal{P}, M_j)$. Then some $k \in \{i, i+1, \ldots, j\}$ exists such that $\mathcal{P}^k$ contains a rule $r_- = (\mathbf{not}\, L \leftarrow body_-.)$ such that $M_j \models body_-$. Then by Lemma 3.5 we have $N \models body_-^j$ and from Lemma 3.3[3] it follows that $rej(L^j, i) \in N$, which is a contradiction with the assumption that r is unrejected. Hence

$$r' \in \mathcal{P}^i \setminus Rej^i(\mathcal{P}, M_j)$$

For the converse implication let $r' = (L \leftarrow body.) \in \mathcal{P}^i \setminus Rej^i(\mathcal{P}, M_j)$ for some $i \in \{1, 2, \ldots, j\}$. By Lemma 3.12 we have that $P_{\mathcal{E}}$ contains a rewritten rule r of level i with L^j in its head and the guardless body $body^j$. We need to prove that r is unrejected. Assume it is rejected, i.e. $\mathbf{not}\, rej(L^j, i) \notin N$. Then $rej(L^j, i) \in N$ and from Lemma 3.3 and Lemma 3.5 it follows that $r' \in Rej^i(\mathcal{P}, M_j)$ which is a contradiction with the assumption. Therefore r is unrejected. $\square$

Lemma 3.8. Let $A \in \mathcal{U}_{assert}$ be such that it trans-appears in $P_{\mathcal{E}}$. Then

$$\mathbf{not}\, A \in Def(\mathcal{P}, M_j) \iff \mathbf{not}\, rej(A^j_{neg}, 0) \in N^-$$

[3] assumptions are satisfied by the existence of r and the rewritten rule of $P_{\mathcal{E}}$ corresponding to r_-

Proof. According to the definition of default assumptions $P_{\mathcal{E}}$ must contain a default assumption of the form

$$A^j_{neg} \leftarrow rej(A^j_{neg}, 0).$$

because A trans-appears in $P_{\mathcal{E}}$. We will prove both implications indirectly.

First let's assume **not** $A \notin Def(\mathcal{P}, M_j)$. Then some rule $r = (A \leftarrow body.) \in \rho(\mathcal{P})$ exists such that $M_j \models body$. $P_{\mathcal{E}}$ contains a corresponding rewritten rule r' (Lemma 3.6) and we also have $N \models body^j$ (Lemma 3.5). Now we can use Lemma 3.3 and the existence of r' and the default assumption mentioned earlier to conclude that $rej(A^j_{neg}, 0) \in N$. Hence **not** $rej(A^j_{neg}, 0) \notin N^-$.

For the converse implication consider that **not** $rej(A^j_{neg}, 0) \notin N^-$. Then $rej(A^j_{neg}, 0) \in N$ and we can use Lemma 3.3 to find a rule r of $P_{\mathcal{E}}$ of the form

$$A^j \leftarrow body, rej(A^j, i).$$

such that $N \models body$. $\mathcal{P}^i$ will contain the corresponding rule with A in its head and its body will be true in M_j (Lemma 3.6 and Lemma 3.5). Hence **not** $A \notin Def(\mathcal{P}, M_j)$. □

Lemma 3.9. For every literal L over $\mathcal{U}_{assert}$ such that it trans-appears in $P_{\mathcal{E}}$ the following holds:

$$L^j \in N \iff L \in R$$

Proof. In order to prove the first implication we will prove

$$(\forall i \in \mathbb{N})(L^j \in N_i \implies L \in R_i)$$

by induction on i:

1° $N_0 = \emptyset$, so the claim holds.

2° We assume that

$$L^j \in N_i \implies L \in R_i$$

holds for all L and prove

$$L^j \in N_{i+1} \Longrightarrow L \in R_{i+1}$$

If $L^j \in N_{i+1}$, then $P_{\mathcal{E}}$ must contain some rule $r = (L^j \leftarrow body.)$ such that $body \subseteq N_i$. This means that all guards in the rule's body are true in N (because $N_i \subseteq N$). If r is a default assumption, then by Lemma 3.8 we get $L \in Def(\mathcal{P}, M_j) \subseteq R_{i+1}$. Otherwise it must be an unrejected rewritten rule and according to Lemma 3.7 the corresponding rule r' has L in its head and $r' \in \rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M_j)$. Moreover by the induction hypothesis we have that the body of r' is a subset of R_i, so $L \in R_{i+1}$.

By another induction on i we will prove:

$$(\forall i \in \mathbb{N})(L \in R_i \Longrightarrow (\exists k \in \mathbb{N})(L^j \in N_k))$$

and the converse implication will follow.

1° $R_0 = \emptyset$, so the claim holds.

2° We assume that

$$L \in R_i \Longrightarrow (\exists k \in \mathbb{N})(L^j \in N_k)$$

holds for all L and prove

$$L \in R_{i+1} \Longrightarrow (\exists k \in \mathbb{N})(L^j \in N_k)$$

If $L \in R_{i+1}$, then

$$[\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M_j)] \cup Def(\mathcal{P}, M_j)$$

must contain some rule $r = (L \leftarrow body.)$ such that $body \subseteq R_i$. Two situations are possible:

(a) If $L \in Def(\mathcal{P}, M_j)$, then by Lemma 3.8 we get that $rej(L^j, 0) \in N^- \subseteq N_1$. Moreover L trans-appears in $P_{\mathcal{E}}$ because the rule

corresponding to r has L^j in its head. So it contains a default assumption for L^j and hence $L^j \in N_2$.

(b) If $r \in \mathcal{P}^i \setminus Rej^i(\mathcal{P}, M_j)$ for some $i \in \{1, 2, \ldots, n\}$, then $P_\mathcal{E}$ contains an unrejected rewritten rule r' with L^j in its head and the guardless body $body^j$ (Lemma 3.7). For each literal $X \in body$ we can use the inductive assumption and get a natural number k_X such that $X^j \in N_{k_X}$. Let $k_1 = \max\{k_X \mid X \in body\}$.

Moreover if the rule r' has an assertion guard $(assert(r))^{i-1}$ in its body, then it must be true in N, because r' is a rewritten rule. So some $k_2 \in \mathbb{N}$ must exist such that $(assert(r))^{i-1} \in N_{k_2}$. If it has no such guard, let's set $k_2 = 1$.

Now let $k = \max\{k_1, k_2\} \geq 1$. As r' is unrejected, we have **not** $rej(L^j, i) \in N^- \subseteq N_k$. Hence the whole body of r' is true in N_k and $L^j \in N_{k+1}$. □

Lemma 3.10. M_j is a dynamic stable model of $\mathcal{P}$.

Proof. M_j is a dynamic stable model of the dynamic logic program $\mathcal{P}$ iff

$$M_j^* = R$$

Now let L be a literal over $\mathcal{U}_{assert}$. Two situations are possible:

1. If L doesn't trans-appear in $P_\mathcal{E}$, then it cannot appear in $\rho(\mathcal{P})$ (if it appeared in a rule $r \in \rho(\mathcal{P})$, then it would trans-appear in the rule of $P_\mathcal{E}$ corresponding to r). So $L \in \{A, \textbf{not}\ A\}$ and **not** $A \in Def(\mathcal{P}, M_j)$ and hence **not** $A \in R$. We also have $A \notin R$ as the program $[\rho(\mathcal{P}) \setminus Rej(\mathcal{P}, M_j)] \cup Def(\mathcal{P}, M_j)$ doesn't contain any rule with A in its head. Furthermore A^j doesn't appear in $P_\mathcal{E}$, so by Proposition 2.17 we have $A^j \notin N$. Hence $A \notin M_j^*$ and also **not** $A \in M_j^*$. So taken all together, we proved

$$L \in M_j^* \iff L \in R$$

2. If L trans-appears in $P_{\mathcal{E}}$, then:

$$L \in M_j^* \iff M_j \models L \overset{Lem.3.5}{\iff} N \models L^j \iff L^j \in N \overset{Lem.3.9}{\iff} L \in R$$

□

Theorem 3.11 (Soundness). $\mathcal{M} = (M_1, M_2, \ldots, M_n)$ is an evolution stable model of P given $\mathcal{E}$.

Proof. Follows by applying Lemma 3.10 n times for $j = 1, j = 2, \ldots, j = n$. □

3.3 Completeness of the Transformation

For the rest of this subsection we will assume P is an evolving logic program, $\mathcal{E} = (E_1, E_2, \ldots, E_n)$ is an event sequence, $\mathcal{M} = (M_1, M_2, \ldots, M_n)$ is an evolution stable model of P given $\mathcal{E}$, $(P_1, P_2, \ldots, P_n)$ is the evolution trace associated to $\mathcal{M}$ and

$$\mathcal{P}_i = (P_1, P_2, \ldots, P_{i-1}, P_i \cup E_i) \text{ for all } i \in \{1, 2, \ldots, n\}$$

In order to prove that the transformation is complete, we have to find a stable model of $P_{\mathcal{E}}$ corresponding to $\mathcal{M}$. First let's define an interpretation N corresponding to $\mathcal{M}$:

$$\begin{aligned} N \quad = \quad & \{L^i \mid i \in \{1, 2, \ldots, n\} \wedge M_i \models L \wedge L \text{ trans-appears in } P_{\mathcal{E}}\} \cup \\ & \cup \{rej(L^i, k) \mid i \in \{1, 2, \ldots, n\} \wedge k \in \{1, 2, \ldots, i\} \wedge (\exists r \in Rej^k(\mathcal{P}_i, M_i))(H(r) = L)\} \cup \\ & \cup \{rej(A^i_{neg}, 0) \mid i \in \{1, 2, \ldots, n\} \wedge \textbf{not } A \notin Def(\mathcal{P}_i, M_i)\} \end{aligned}$$

The rest of this section is devoted to proving that N is a stable model of $P_{\mathcal{E}}$. According to Definition 2.15 and Theorem 2.12 this property can be rewritten as

$$N^* = R = \bigcup_{i<\omega} R_i$$

where $R_0 = \emptyset$ and $R_{i+1} = T_{P_{\mathcal{E}} \cup N^-}(R_i)$ for all $i \geq 0$. We will prove this equality in three steps:

1. The first step is also the most difficult. For every literal L over $\mathcal{U}_{assert}$ and every $j \in \{1, 2, \ldots, n\}$ we will prove

$$L^j \in N \Longleftrightarrow L^j \in R$$

by complete induction on j. The inductive hypothesis will be used in many places so it is formulated here once and for all: Let $j \in \{1, 2, \ldots, n\}$. Then for every literal L over $\mathcal{U}_{assert}$ the following holds:

$$(\forall i \in \{1, 2, \ldots, j-1\})(L^i \in N \Longleftrightarrow L^i \in R) \tag{3.6}$$

2. Next we will prove that for every literal L over $\mathcal{U}_{assert}$, every $j \in \{1, 2, \ldots, n\}$ and every $i \in \{0, 1, \ldots, j\}$ it holds that

$$rej(L^j, i) \in N \Longleftrightarrow rej(L^j, i) \in R$$

3. The last thing we have to take care of is that none of the totality constraints are broken, i.e. we have to prove

$$u \notin R$$

Now we will introduce notation used in the proofs:

- A rewritten rule is every rewritten program rule, rewritten event rule and every assertion rule of $P_{\mathcal{E}}$ with its assertion guard true in R.
- A rewritten rule is unrejected iff its rejection guard is true in R.
- Let $j \in \{1, 2, \ldots, n\}$. As M_j is a dynamic stable model of $\mathcal{P}_j$, we can use Definition 2.21 and Theorem 2.12 to get

$$M_j^* = \bigcup_{i<\omega} M_{j,i}$$

where $M_{j,0} = \emptyset$ and $M_{j,i+1} = T_{[\rho(\mathcal{P}_j) \setminus Rej(\mathcal{P}_j, M_j)] \cup Def(\mathcal{P}_j, M_j)}(M_{j,i})$.

The preparation phase is now over, we can step forward to the proofs. The first 5 lemmas are a prelude to the proof of the first step.

Lemma 3.12. Assume (3.6) holds and let $j \in \{1, 2, \ldots, n\}$. $P_{\mathcal{E}}$ contains a rewritten rule of level i with L^j in its head and a guardless body $body^j$ iff

$$(L \leftarrow body.) \in \mathcal{P}_j^i$$

Proof. Let $P_{\mathcal{E}}$ contain a rewritten rule r of level i with L^j in its head and a guardless body $body^j$. Let $r' = (L \leftarrow body.)$. We will consider 2 cases:

1. If r is a rewritten program rule or a rewritten event rule, then $i \in \{1, j\}$ and $\mathcal{P}_j^i$ contains the rule r' by the definition of $P_{\mathcal{E}}$.

2. If r is an assertion rule, then $i \in \{2, 3, \ldots, j\}$ and its assertion guard $(assert(r'))^{i-1}$ is true in R. Then by (3.6) we have $(assert(r'))^{i-1} \in N$ and by the definition of N we have $assert(r') \in M_{i-1}$. Hence $r' \in \mathcal{P}_j^i$.

For the converse implication let $r' = (L \leftarrow body.) \in \mathcal{P}_j^i$ for some $i \in \{1, 2, \ldots, j\}$. We will consider two cases:

1. If $r' \in P$ or $r' \in E_j$, then$P_{\mathcal{E}}$ contains a rewritten rule of level i with L^j in its head and the guardless body $body^j$ by the definition.

2. If $r' \in P_i$ and $i > 1$, then $assert(r') \in M_{i-1}$ and therefore $(assert(r'))^{i-1} \in N$. By (3.6) we get $(assert(r'))^{i-1} \in R$. Hence $P_{\mathcal{E}}$ must contain a rule with $(assert(r'))^{i-1}$ in its head and by the definition of $P_{\mathcal{E}}$ this implies it also contains the assertion rule

$$L^j \leftarrow body^j, (assert(r))^{i-1}, \mathbf{not}\, rej(L^j, i). \qquad \square$$

Lemma 3.13. Assume (3.6) holds and let $j \in \{1, 2, \ldots, n\}$. $P_{\mathcal{E}}$ contains an unrejected rewritten rule with L^j in its head and a guardless body $body^j$ iff

$$(L \leftarrow body.) \in \rho(\mathcal{P}_j) \setminus Rej(\mathcal{P}_j, M_j)$$

Proof. Let $P_{\mathcal{E}}$ contain an unrejected rewritten rule r of level i with L^j in its head and a guardless body $body^j$. Then by Lemma 3.12 $r' = (L \leftarrow body.) \in \mathcal{P}_j^i$. It is left to prove that r' is not a member of the set $Rej^i(\mathcal{P}_j, M_j)$. As r is unrejected, we know that $\mathbf{not}\, rej(L^j, i) \in R$. The only source of default

literals in R is N^-, so **not** $rej(L^j, i) \in N^-$. Therefore $rej(L^j, i) \notin N$ and by the definition of N we have that no rule of $Rej^i(\mathcal{P}_j, M_j)$ has L in its head. Thus $r' \notin Rej^i(\mathcal{P}_j, M_j)$.

For the converse implication let $r_1 = (L \leftarrow body.) \in \mathcal{P}_j^i \setminus Rej^i(\mathcal{P}_j, M_j)$ for some $i \in \{1, 2, \ldots, j\}$. By Lemma 3.12 we have that $P_{\mathcal{E}}$ contains a rewritten rule r_1' with L^j in its head and the guardless body $body^j$. Now we need to prove that r is unrejected. Assume it is rejected, i.e. **not** $rej(L^j, i) \notin R$. Then **not** $rej(L^j, i) \notin N^-$ and thus $rej(L^j, i) \in N$. By the definition of N we have that some rule $r_2 \in Rej^i(\mathcal{P}_j, M_j)$ exists such that its head is L. Also, according to the definition of $Rej^i(\mathcal{P}_j, M_j)$, some $k \in \{i, i+1, \ldots, j\}$ and some rule $r_3 \in \mathcal{P}_j^k$ with **not** L in its head and its body satisfied in M_j must exist. But according to the very same definition, the existence of r_3 implies $r_1 \in Rej^i(\mathcal{P}_j, M_j)$, which is a contradiction with the assumption. □

Lemma 3.14. Let A be an atom of $\mathcal{U}_{assert}$ that trans-appears in $P_{\mathcal{E}}$. Then

$$\textbf{not}\ A \in Def(\mathcal{P}_j, M_j) \iff \textbf{not}\ rej(A^j_{neg}, 0) \in R_1$$

Proof. Assume **not** $A \in Def(\mathcal{P}_j, M_j)$. Then according to the definition of N we have $rej(A^j_{neg}, 0) \notin N$. This implies **not** $rej(A^j_{neg}, 0) \in N^- \subseteq R_1$.

On the other hand if **not** $rej(A^j_{neg}, 0) \in R_1$, then it must be the case that **not** $rej(A^j_{neg}, 0) \in N^-$. But this implies $rej(A^j_{neg}, 0) \notin N$ and by the definition of N we get **not** $A \in Def(\mathcal{P}_j, M_j)$. □

Lemma 3.15. Let L be a literal over $\mathcal{U}_{assert}$. If (3.6) holds, then

$$(\forall i \in \mathbb{N})(L \in M_{j,i} \wedge L \text{ trans-appears in } P_{\mathcal{E}} \Longrightarrow (\exists l \in \mathbb{N})(L^j \in R_l))$$

Proof. We will prove by induction on i:

1° For $i = 0$ the claim trivially follows from $M_{j,0} = \emptyset$.

2° We assume the claim holds for i, i.e. if L is a literal over $\mathcal{U}_{assert}$, then

$$L \in M_{j,i} \wedge L \text{ trans-appears in } P_{\mathcal{E}} \Longrightarrow (\exists l \in \mathbb{N})(L^j \in R_l)$$

We will prove the claim for $i+1$. So let's assume $L \in M_{j,i+1}$ and L trans-appears in $P_{\mathcal{E}}$. Then there is some rule $r = (L \leftarrow body.) \in [\rho(\mathcal{P}_j) \setminus Rej(\mathcal{P}_j, M_j)] \cup Def(\mathcal{P}_j, M_j)$ such that $body \subseteq M_{j,i}$. Let's consider 2 cases:

(a) If r is a default assumption, then $body$ is empty and L is a default literal **not** A. $P_{\mathcal{E}}$ must also contain a default assumption of the form

$$A^j_{neg} \leftarrow \textbf{not}\, rej(A^j_{neg}, 0).$$

because L trans-appears in $P_{\mathcal{E}}$. Moreover by Lemma 3.14 we have **not** $rej(A^j_{neg}, 0) \in R_1$. Hence $A^j_{neg} \in R_2$.

(b) If $r \in \rho(\mathcal{P}_j) \setminus Rej(\mathcal{P}_j, M_j)$, then by Lemma 3.13 there must be some unrejected rewritten rule r' of $P_{\mathcal{E}}$ with L^j in its head and the guardless body $body^j$. We can apply the inductive assumption for each literal in $body$ and take the maximum p of all the indices we get. From the monotonicity of the immediate consequence operator $T_P(\cdot)$ we have $body^j \subseteq R_p$. If r' has an assertion guard, then let $q \in \mathbb{N}$ be such that the assertion guard belongs to R_q and $s = \max\{p, q\}$. If it has no assertion guard, let $s = p$. Now we have $L^j \in R_{s+1}$. □

Lemma 3.16. Let L be a literal over $\mathcal{U}_{assert}$. If (3.6) holds, then

$$(\forall i \in \mathbb{N})(L^j \in R_i \Longrightarrow L \text{ trans-appears in } P_{\mathcal{E}} \wedge (\exists l \in \mathbb{N})(L \in M_{j,l}))$$

Proof. We will prove by induction on i:

1° For $i = 0$ the claim trivially follows from $R_0 = \emptyset$.

2° We assume the claim holds for i, i.e. if L is a literal over $\mathcal{U}_{assert}$, then

$$L^j \in R_i \Longrightarrow L \text{ trans-appears in } P_{\mathcal{E}} \wedge (\exists l \in \mathbb{N})(L \in M_{j,l})$$

We will prove the claim for $i+1$. So let's assume $L^j \in R_{i+1}$. Then there is some rule $r \in P_{\mathcal{E}}$ with L^j in its head and its body satisfied in

R_i. Hence the first part of the proof is done – L trans-appears in $P_{\mathcal{E}}$ because $P_{\mathcal{E}}$ contains a rule with L^j in its head. To prove the second proposition, let's consider 2 cases:

(a) If r is a default assumption, then L is a default literal **not** A, r is of the form

$$A^j_{neg} \leftarrow \mathbf{not}\, rej(A^j_{neg}, 0).$$

and $\mathbf{not}\, rej(A^j_{neg}, 0) \in R_1$. So by Lemma 3.14 we have $\mathbf{not}\, A \in Def(\mathcal{P}_j, M_j)$ and thus $\mathbf{not}\, A \in M_{j,1}$.

(b) If r is not a default assumption, then it must be an unrejected rewritten rule of $P_{\mathcal{E}}$. Let its guardless body be $body^j$. Then by Lemma 3.13 we have

$$(L \leftarrow body.) \in \rho(\mathcal{P}_j) \setminus Rej(\mathcal{P}_j, M_j)$$

We can apply the inductive assumption for each literal in $body^j$ and take the maximum p of all the indices we get. From the monotonicity of the immediate consequence operator $T_P(\cdot)$ we have $body \subseteq M_{j,p}$. Hence $L \in M_{j,p+1}$. □

Lemma 3.17. Let L be a literal over $\mathcal{U}_{assert}$ and $j \in \{1, 2, \ldots, n\}$. Then

$$L^j \in N \iff L^j \in R$$

Proof. We will prove by complete induction on j.

1° The basis can be inferred from the inductive step with $j = 1$.

2° We assume (3.6) holds and prove

$$L^j \in N \iff L^j \in R$$

First let $L^j \in N$. Then $M_j \models L$ and L trans-appears in $P_{\mathcal{E}}$. So some $i \in \mathbb{N}$ exists such that $L \in M_{j,i}$ and by Lemma 3.15 we have $L^j \in R_l$ for some $l \in \mathbb{N}$. Hence $L^j \in R$.

On the other hand if $L^j \in R$, then some $i \in \mathbb{N}$ exists such that $L^j \in R_i$. Thus by Lemma 3.16 L trans-appears in $P_{\mathcal{E}}$ and $L \in M_{j,l}$ for some $l \in \mathbb{N}$. This implies $L^j \in N$. □

Lemma 3.18. Let L be a literal over $\mathcal{U}_{assert}$, $j \in \{1, 2, \ldots, n\}$ and $i \in \{0, 1, \ldots, j\}$. Then

$$rej(L^j, i) \in N \iff rej(L^j, i) \in R$$

Proof. We know that $rej(L^j, i) \in N$ holds iff $Rej^i(\mathcal{P}_j, M_j)$ contains a rule r_1 with L in its head. This in turn holds iff some $k \in \{i, i+1, \ldots, j\}$ exists such that $\mathcal{P}_j^k$ contains a rule $r_2 = (\textbf{not}\, L \leftarrow body.)$ such that $M_j \models body$. Furthermore by Lemma 3.12 and Lemma 3.17 this holds iff

$$P_{\mathcal{E}} \text{ contains a rewritten rule } r_2' \text{ with } L_{neg}^j \text{ in its head and its guardless body } body^j \text{ is a subset of } R \tag{3.7}$$

Now let's assume $rej(L^j, i) \in N$. Then (3.7) holds and according to the definition of rejection rules, $P_{\mathcal{E}}$ must also contain the rules

$$\begin{aligned}
rej(L^j, i_1) &\leftarrow body'.\\
rej(L^j, i_2) &\leftarrow rej(L^j, i_1).\\
rej(L^j, i_3) &\leftarrow rej(L^j, i_2).\\
&\vdots\\
rej(L^j, i_s) &\leftarrow rej(L^j, i_{s-1}).
\end{aligned}$$

where $k \geq i_1 > i_2 > \ldots > i_s = i$ and $body'$ contains all literals in $body^j$ and the assertion guard of r_2' if it has one. As $body^j \subseteq R$ and r_2' is a rewritten rule, there must be some $p \in \mathbb{N}$ such that $body' \subseteq R_p$. Hence we have $rej(L^j, i_1) \in R_{p+1}, rej(L^j, i_2) \in R_{p+2}, \ldots, rej(L^j, i) \in R_{p+s} \subseteq R$.

For the converse implication let's assume $rej(L^j, i) \in R$. Then $rej(L^j, i) \in R_p$ for some $p \in \mathbb{N}$, so $P_{\mathcal{E}}$ contains some rejection rule r_3' with $rej(L^j, i)$ in its head and its body satisfied in R_{p-1}. We will consider two cases:

1. If r_3' is of the form (3.2), then a look at the definition of rejection rules

tells us that (3.7) is satisfied. Thus $rej(L^j, i) \in N$.

2. If r'_3 is of the form (3.3), then its body is of the form $rej(L^j, i_1)$ where $i < i_1 \leq j$. As $rej(L^j, i_1) \in R_{p-1}$, $P_{\mathcal{E}}$ must contain a rejection rule with $rej(L^j, i_1)$ in its head and its body satisfied in R_{p-2}. Two cases are possible again, in the first the proof ends and in the second we get an index i_2 such that $i_1 < i_2 \leq j$ and $rej(L^j, i_2) \in R_{p-2}$. If the second case would occur forever, we would get an infinite increasing bounded sequence of natural numbers $i < i_1 < i_2 < \ldots \leq j$, which is not possible. Hence after a finite number of iterations the first case must occur. □

Lemma 3.19. It holds that

$$u \notin R$$

Proof. We will prove by contradiction. Assume $u \in R$. Then for some atom A of $\mathcal{U}_{assert}$ we have

$$\textbf{not}\ A^j, \textbf{not}\ A^j_{neg} \in R$$

This implies

$$\textbf{not}\ A^j, \textbf{not}\ A^j_{neg} \in N^-$$

so we have

$$A^j, A^j_{neg} \notin N$$

But then by the definition of N we have both $M_j \not\models A$ and $M_j \not\models \textbf{not}\ A$, which is not possible. □

Theorem 3.20 (Completeness)**.** N is a stable model of $P_{\mathcal{E}}$.

Proof. Follows from Lemma 3.17, Lemma 3.18 and Lemma 3.19. □

3.4 Size of the Transformed Program

Chapter 4

Implementation of EVOLP

maybe some sections here…

Chapter 5

Conclusion and Future Work

Future work:

- arithmetic predicates
- strong negation
- rule variables

References

[1] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic logic programming. In A. Cohn, L. Schubert, and S. Shapiro, editors, *Procs. of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 98–111. Morgan-Kaufmann, June 1998.

[2] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Updates of logic programs by logic programs. In *IIS'98: Seventh International Symposium on Intelligent Information Systems (Former WIS Series)*, pages 98–111, June 1998.

[3] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic logic programming. In M. Falaschi J. L. Freire and M. Vialres-Ferro, editors, *Procs. of the 1998 Joint Conference on Declarative Programming (AGP'98)*, pages 393–408, July 1998.

[4] José Júlio Alferes, João Alexandre Leite, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. Dynamic updates of non-monotonic knowledge bases. *The Journal of Logic Programming*, 45(1-3):43–70, September/October 2000.

[5] José Júlio Alferes, Federico Banti, and Antonio Brogi. A principled semantics for logic program updates. In Brewka and Peppas, editors, *Nonmonotonic Reasoning, Action, and Change (NRAC'03)*, 2003.

[6] José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. Semantics for dynamic logic programming: a principle-based approach. In V. Lifschitz and I. Niemelä, editors, *Procs. of the Seventh International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7)*, pages 8–20. Springer-Verlag, 2004.

[7] José Júlio Alferes, Federico Banti, Antonio Brogi, and João Alexandre Leite. The refined extension principle for semantics of dynamic logic programming. *Studia Logica*, 79(1):7–32, 2005.

[8] João Alexandre Leite. *Evolving Knowledge Bases – Specification and Semantics*. PhD thesis, Universidade Nova de Lisboa, July 2002.

[9] Martin Homola. Various semantics are equal on acyclic programs. In J. A. Leite and P. Torroni, editors, *Procs. of the 5th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA V)*, pages 78–95. Springer, 2004.

[10] José Júlio Alferes, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. LUPS – a language for updating logic programs. In M. Gelfond, N. Leone, and G. Pfeifer, editors, *Procs. of the 5th International Conference on Logic Programming and Nonmonotonic Reasoning*, pages 162–176. Springer, 1999.

[11] José Júlio Alferes, Luís Moniz Pereira, Halina Przymusinska, and Teodor C. Przymusinski. LUPS – a language for updating logic programs. *Artificial Intelligence*, 138(1&2), June 2002.

[12] Thomas Eiter, Michael Fink, Giuliana Sabbatini, and Hans Tompits. A framework for declarative update specifications in logic programs. In *IJCAI'01*, pages 649–654. Morgan-Kaufmann, 2001.

[13] José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira. Evolving logic programs. In S. Flesca, S. Greco, N. Leone, and G. Ianni, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (JELIA'02)*, pages 50–61. Springer-Verlag, 2002.

[14] Implementations of logic programs updates, http://centria.di.fct.unl.pt/~jja/updates/.

[15] João Alexandre Leite and Luís Soares. Enhancing a multi-agent system with evolving logic programs. In K. Inoue, K. Satoh, and F. Toni, editors, *Pre-Procs. of the 7th International Workshop on Computational Logic in Multi-Agent Systems, (CLIMA VII)*, pages 207–222, 2006.

[16] João Alexandre Leite and Luís Soares. Adding evolving abilities to a multi-agent system. In K. Satoh K. Inoue and F. Toni, editors, *Procs. of the 7th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VII)*. Springer-Verlag, 2007. To appear.

[17] Federico Banti, José Júlio Alferes, and Antonio Brogi. Operational semantics for DyLPs. In C. Bento A. Cardoso and G. Dias, editors, *Progress in Artificial Intelligence, Procs. 12th Portuguese Int. Conf. on Artificial Intelligence (EPIA'05)*, pages 43–54. Springer, 2005.

[18] Chitta Baral. *Knowledge Representation, Reasoning, and Declarative Problem Solving*. Cambridge University Press, New York, NY, USA, 2003.

[19] José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira. An evolving agent with evolp. In P. Rullo N. Leone, editor, *Procs. of APPIA-GULP-PRODE'03 Joint Conf. on Declarative Programming (AGP'03)*, pages 205–216, September 2003.

[20] José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira. An evolvable rule-based e-mail agent. In S. Abreu, editor, *Progress in Artificial Intelligence, Procs. 11th Portuguese Int. Conf. on Artificial Intelligence (EPIA'03)*. Springer, December 2003.

[21] José Júlio Alferes, Antonio Brogi, João Alexandre Leite, and Luís Moniz Pereira. An evolving agent with evolp. In M. Klusch, S. Ossowski, A. Omicini, and H. Laamanen, editors, *Procs. of the 7th International Workshop on Cooperative Information Agents (CIA'03)*, pages 281–297. Springer-Verlag, 2003.

Definition Index