Metaphorical mapping consistency via Dynamic Logic Programming

João Alexandre Leite*; Francisco Câmara Pereira†
Amílcar Cardoso†; Luís Moniz Pereira*
*Centro de Inteligência Artificial (CENTRIA), Departamento de Informática
Universidade Nova de Lisboa, 2825-114 Caparica, Portugal
†Centro de Informática e Sistemas (CISUC), Universidade de Coimbra
Polo II, Pinhal de Marrocos, 3030 Coimbra, Portugal
jleite@di.fct.unl.pt; camara@dei.uc.pt
amilcar@dei.uc.pt; lmp@di.fct.unl.pt

Abstract

In this paper we explore a symbiosis between *Dynamic Logic Programming* and *Metaphor* to solve the problem of inconsistency in metaphorical knowledge integration. The problem of solving inconsistencies that may arise when knowledge from two different domains is combined, given a metaphorical mapping, is crucial, be it at the stage where we want to evaluate the appropriateness of the mapping function, or at a subsequent stage when we want to reason with the combined knowledge. We propose, in a formal and rigorous manner, a transformation and its semantics that solves the problem of inter-domain inconsistencies by employing the principle of inertia to the rules of the source domain. This work is part of an ongoing larger project, *Dr. Divago*, whose final goal is to develop a system to perform automatic creative reasoning.

1 Introduction and Motivation

It is common to surround the word *Creativity* with a mystic aura of no less ethereal concepts like inspiration, gift and genius. Some, on the other hand, have been investigating towards our understanding of what is creativity, or more precisely, what makes us creative in comparison to other animals. Among these theories, e.g. Guilford (1967); De Bono (1970), lies the fact that the ability to associate apparently unrelated concepts, to see similarity where there seems to be difference, to break rules or ignore constraints where it is not expected to happen, is recurrently present in situations involving the word *Creativity*.

In modelling Computational Creativity, we believe it is relevant to understand that a creative system must be able to work in a heterogeneous knowledge base of different domains, in such a way that it can extract and interrelate concepts apparently distant and different. Recent Metaphor Theories (Gentner et al., 1989; Martin, 1990; Indurkhya, 1992; Veale and Keane, 1993; Barnden, 1997) seem to be quite promising in fulfilling this goal. These theories set forth processes to associate concepts according to an initial metaphor and yielding an interpretation. An interpretation consists in a coherent 1-to-1 mapping of elements between two domains (the *Source* and the *Target*, commonly called *Vehicle* and *Tenor*) starting from an initially chosen pair of mapped elements.

Apart from having been deeply studied along the ages, and notably being a very common process of association between domains, according to Eco (1984) and Veale and Keane (1993), Metaphor is deeply embodied in the core of our cognition, having a constant and vital role in communication (Lakoff and Johnson, 1980).

In what concerns modelling Computational Creativity, metaphor may play a determinant role, since it can be used as a device for cross-domain interrelation establishment. With such a device, a system could be able to search in domains different from the one directly related to the task at hand, since there would be counterparts for some concepts (e.g. to search for new *"musical"* ideas in Visual Arts, to search for new *"computational"* concepts in Social Environments, etc.). An example of work in progress towards such a system is *Dr. Divago* (Pereira, 1998). In this system, knowledge is represented at two distinct levels: instancial knowledge in the form of tree-like structures (e.g. musical structures); ontological knowledge in the form of concept maps mainly describing the concepts found in instancial knowledge (e.g. generic knowledge about music). The role of Metaphor in this system is to establish correspondences between concept maps of different domains, thus allowing for transfer of information contained in instancial knowledge. The usefulness of this cross-domain mapping obviously depends much on the quality of the concept maps involved, this being the motivation leading to the development of *Clouds* (Pereira, 2000), a module now integrated in *Dr. Divago*, aiming at helping the user in building her own concept maps by means of machine learning techniques.

In the present work, within *Dr. Divago*, we want to extend the generation of cross-domain links such as those based on the metaphor theory of Veale and Keane (1993), to cope with the transfer of operational/procedural knowledge from one domain to another. In such cases, particularly when negative information is bought into the arena, inconsistency is a reality that has to be dealt with, as shown by the following example:

Example 1 *Consider the rule* "objects aren't big when they are in the background"*, extracted from* Clouds *in the domain of Visual Arts, represented in the clausal form as:*

$$not\, big(X) \leftarrow object(X), background(X). \quad (1)$$

which, if we were to map it to the music domain according to the following translation:

$$big \rightarrow long \qquad object \rightarrow motif$$
$$background \rightarrow accompaniment$$

would correspond to:

$$not\, long(X) \leftarrow motif(X), accompaniment(X). \quad (2)$$

This can easily be proven inconsistent, at least for some instances (it is not difficult to find accompaniment motifs *that are* long*, eg. isometric motets from ars nova).* ⋄

Each metaphorical mapping will lead to the transfer of a set of rules from the *Vehicle* to the *Tenor* of which: some are redundant because equivalent rules already exist there; some, of particular interest to the creative process, yield new knowledge to the *Tenor*; some are contradictory with the knowledge already present in the *Tenor*. Although contradiction is not necessarily evil, this being more evident in creative processes, we nevertheless have to detect and deal with it.

The need to detect and deal with contradiction in knowledge transfer lead us to consider the recent paradigm of *Dynamic Logic Programming* (Alferes et al., 1998, 2000) as a formal, while at the same time intuitive, vehicle to a solution.

The paradigm of *Dynamic Logic Programming (DLP)*, supported by the notion of *Logic Program Updates* (Leite, 1997; Leite and Pereira, 1997, 1998; Alferes et al., 1998, 2000), is simple and quite fundamental. Suppose that we are given a set of theories (encoded as generalized logic programs) representing different states of the world. Different states may represent different time periods or different sets of priorities or, in our case, different domains. Consequently, the individual theories contain mutually contradictory as well as overlapping information. The role of *Logic Program Updates* is to use the mutual relationships existing between different states to precisely determine the *declarative* as well as the *procedural* semantics of the combined theory, composed of all individual theories. Cross-domain inconsistencies are detected and solved due to the application of the principle of inertia to the individual rules of the theories. This principle of inertia, applied to the individual rules of a theory, states that a rule from the initial knowledge should only persist in time, after an update, if it does not lead to a contradiction by means of a new rule. In our case, the rules from the *Vehicle*, after the metaphorical mapping, should only persist if they do not lead to a contradiction by means of a rule from the *Tenor*.

Going back to the example above, let us suppose that we learn, in the music domain, the following rule:

$$\begin{aligned} long(X) \leftarrow\; & motif(X), accompaniment(X), \\ & isometric\_motet\_element(X). \end{aligned} \quad (3)$$

stating that *"the accompaniment element of an isometric motet is long"*[1]. If we were simply to consider the union of both rules, we would obtain a contradiction for all *accompaniment elements of an isometric motet*. If we were to simply consider rule 3, we would loose valuable information from rule 2 concerning *accompaniment motifs not belonging to isometric motets*. Our goal is to be able to conclude that *"accompaniment elements of an isometric motet are long"*, and that *"accompaniment elements of non-isometric motets aren't long"*. That is, we would like to use rule 2 for those instances that do not generate a contradiction by means of rule 3. This is precisely the behavior of *Logic Program Updates*: to exert inertia on those rules (in this case obtained by the mapping) that are not contradicted by rules from the new domain. In Example 3, we will return to this problem and show how to obtain the desired result.

The non-monotonic behavior of DLP, together with its modular characteristic shows to be quite important when dealing with metaphors.

This paper presents a symbiosis of the two above mentioned theories (*DLP* and *Metaphor*) towards the goal of *Computational Creativity*. It is being implemented as part of the above mentioned *Dr. Divago* (Pereira, 1998), where interesting results are to be expected.

Enjoying the advantages of employing a theoretically sound formalism to the problem of *Metaphorical Reasoning*, DLP has also been implemented as a meta-interpreter (DLP System, 1998) running under the XSB System (1999), allowing for not only theoretical but also practical reasoning.

The paper is structured in the following way: in Sect.2 we elaborate on *Metaphor Theories*; in Sect.3 we introduce the reader to the notions of *Logic Program Updates* and DLP; in Sect.4 we set forth the notion of *Metaphorical Reasoning* by means of DLP; in Sect.5 we draw some properties and illustrate with examples; in Sect.6 we conclude and give hints about future developments.

[1] Also known as Talia

2 Metaphor Theory

Metaphor is a constantly used resource of communication, be it as an embellishment for discourse or as a necessary device to assess concepts unexplainable in other ways. Since in any Natural Language, lexicon is not static or exhaustive, Metaphor has a constructive role in the evolution of communication. Words or expressions like "software", "e-mail" or "search engine" are recent examples of metaphors that are becoming entries in our dictionaries. Furthermore, some researchers advocate that Metaphor is a central device for Learning (Winston, 1980) and is an irreducible and irreplaceable function at the basis of our creative faculties (Richards, 1936; Black, 1962).

There seems to be general agreement that metaphor involves two objects or situations and some kind of transference from an object or situation to the other. One object is referred to as the *tenor* or the target domain, and the other object as the *vehicle* or the source domain. For example, the expression "Your claims are *indefensible*" follows the metaphor "Argument is War"(Lakoff and Johnson, 1980), in which an argument (the tenor) is partially defined, understood, performed and talked about in terms of a war (the *vehicle*). Here, we can observe the *Systematicity* of Metaphor: it is not limited to a single, static, association between two concepts (Argument and War). Instead, other associations emerge dynamically as one explores further in both domains, i.e., "Argument is War" is the underlying metaphor for "He *attacked every weak point* in my argument", "His criticisms were *right on target*", "I *demolished* his argument", "If you use that *strategy*, he'll *wipe you out*", and others that appear in everyday speech.

An interesting and subtle property of a metaphorical interpretation is that it is directional, i.e., each domain or object has a different role and its interchange, although possibly yielding an equally valuable metaphor, will not lead to the same meaning. For example, if we have the metaphor "War is Argument", things turn quite different. It is somewhat unconventional to talk about a war as being an argument (then, it would be natural to say "the general *exposed his claims* quite aggressively" or "thousands of soldiers perished from that *discussion*") because this metaphor is not so deeply rooted in our common sense reasoning.

While it is natural of metaphor to be associated with creative thought and freedom of association, it is constrained by deep rules of coherency. Although it is not expected, in a metaphor, to find a mapping for every single concept in a domain, those that are mapped should be coherent among themselves (this phenomenon is called Local Coherency in Indurkhya (1992)). Some research on metaphor interpretation consist precisely in finding the largest mapping function that avoids inconsistencies. One example is Veale and Keane's metaphor interpretation framework, called *Sapper* (Veale and Keane, 1993). It uses a hybrid model of semantic memory that consists of a connectionist structure in which each unit (or node) or fixed cluster of units is assigned to a concept, and an activation-carrying inter-unit linkage is assigned to each inter-concept relation. Also known as a *localist* network, this organization receives all domain knowledge Sapper uses to interpret metaphor. It applies 3 operational principles:

1. Metaphor is a means of learning new conceptual structure by linking existing diverse schemata in novel ways. This linkage of domains is achieved by augmenting the network with *conceptual bridges* that link the *tenor* and *vehicle* schemata of the metaphor (Veale, 1995).
2. Metaphor comprehension involves *explicit* structural changes to the semantic memory network; these changes, essentially the conceptual bridges of (1) above, are explicit inasmuch as they are recognizably the trace residue of a novel metaphor, and as such may be built upon (elaborated) at a later time (Veale, 1995).
3. A metaphor is a dynamic conceptual *agency*, which may continue to grow as more conceptual structure is acquired regarding either the *tenor* and the *vehicle* domains.

In Sapper, metaphor interpretation takes two major cyclic steps:

1. In the symbolic mode of processing, it searches for potentially inter-concept relations between the two different domains (*tenor* and *vehicle*). These relations, to which Veale calls *dormant bridges*, are obtained from the application of the following two rules:
   - The triangulation rule: "Whenever two concepts share an association with a third concept, this association provides for a plausible dormant bridge".
   - The squaring rule (second order similarity): If two concepts share an association with other two concepts that are connected by an awaken bridge (as described below), this association provides for a plausible dormant bridge.
2. The conceptual bridge awakening phase is performed in the connectionist mode of processing, in which dormant bridges, as laid down in the symbolic mode, are recognized to represent *domain crossover points* between the *tenor* and *vehicle* schemata, and are thus *awakened* or *burnt in*.

We are adopting these ideas to develop our cross-domain links generator, which allows to obtain metaphorical program mappings, described later in this paper.

Other works related to metaphor are also meaningful, such as Barnden (1997); Gentner et al. (1989); Martin (1990).

3 Dynamic Logic Programming

The idea of *dynamic program updates*, inspired by Leite (1997), is simple and quite fundamental. Suppose that we are given a set of program modules P_s, indexed by different states of the world s. Each program P_s contains some knowledge that is supposed to be true at the state s. Different states may represent different time periods or different sets of priorities or, in our case, different domains. Consequently, the individual program modules may contain mutually contradictory as well as overlapping information. The role of the *dynamic program update* $\bigoplus \{P_s : s \in S\}$ is to use the mutual relationships existing between different states (and specified in the form of the ordering relation) to precisely determine, at any given state s, the *declarative* as well as the *procedural* semantics of the combined program, composed of all modules.

Consequently, the notion of a *dynamic program update* supports the important paradigm of *dynamic logic programming*. Given individual and largely *independent* program modules P_s describing our knowledge at different states of the world (for example, the knowledge acquired at different times), the *dynamic program update* $\bigoplus \{P_s : s \in S\}$ specifies the exact meaning of the union of these programs. Dynamic programming significantly facilitates modularization of logic programming and, thus, modularization of non-monotonic reasoning as a whole.

In this section we start by recalling the definition of the so called generalized logic programs and their stable semantics (Alferes et al., 1998) which extend the stable semantics of normal logic programs (Gelfond and Lifschitz, 1988). We then recall the definition and semantic characterization of the update of a generalized logic program by means of another such logic program $P_1 \oplus P_2$.

Since we have that the notion of *dynamic program update* $\bigoplus\{ P_s : s \in S\}$ over an ordered set $\mathcal{P} = \{ P_s : s \in S\}$ of logic programs is a generalization of the notion of single program updates $P_1 \oplus P_2$, throughout the remainder of the paper, we will restrict ourselves, without loss of generality, to the case of single program updates $P \oplus U$.

For a formal and detailed presentation of *dynamic program update* and *dynamic logic programming*, the reader is referred to Alferes et al. (1998, 2000).

3.1 Generalized Logic Programs and their Stable Models

In order to represent *negative* information in logic programs and in their updates, we need more general logic programs that allow default negation $not\, A$ not only in premises of their clauses but also in their heads. We call such programs *generalized logic programs.*

It will be convenient to *syntactically* represent generalized logic programs as *propositional Horn theories*. In particular, we will represent default negation $not\, A$ as a standard propositional variable (atom). Suppose that $\mathcal{K}$ is an arbitrary set of propositional variables whose names do not begin with a "not". By the propositional language $\mathcal{L}_{\mathcal{K}}$ *generated* by the set $\mathcal{K}$ we mean the language $\mathcal{L}$ whose set of propositional variables consists of:

$$\{A : A \in \mathcal{K}\} \cup \{not\, A : A \in \mathcal{K}\}.$$

Atoms $A \in \mathcal{K}$, are called *objective atoms* while the atoms $not\, A$ are called *default atoms*. From the definition it follows that the two sets are disjoint.

By a *generalized logic program P* in the language $\mathcal{L}_{\mathcal{K}}$ we mean a finite or infinite set of propositional Horn clauses of the form:

$$L \leftarrow L_1, \ldots , L_n$$

where L and L_i are atoms from $\mathcal{L}_{\mathcal{K}}$. If all the atoms L appearing in heads of clauses of P are objective atoms, then we say that the logic program P is *normal.* Consequently, from a syntactic standpoint, a logic program is simply viewed as a propositional Horn theory. However, its *semantics* significantly differs from the semantics of classical propositional theories and is determined by the class of stable models defined below.

By a (2-valued) *interpretation* M of $\mathcal{L}_{\mathcal{K}}$ we mean any set of atoms from $\mathcal{L}_{\mathcal{K}}$ that satisfies the condition that for any A in $\mathcal{K}$, precisely one of the atoms A or $not\, A$ belongs to M. Given an interpretation M we define:

$$\begin{aligned} M^{+} &= \{A \in \mathcal{K} : A \in M\} \\ M^{-} &= \{not\, A : not\, A \in M\} = \\ &= \{\, not\, A : A \notin M\}. \end{aligned}$$

Definition 1 (Stable models of generalized logic progs.) *We say that a (2-valued) interpretation M of $\mathcal{L}_{\mathcal{K}}$ is a stable model of a generalized logic program P if M is the least model of the Horn theory $P \cup M^{-}$:*

$$M = \mathit{Least}(P \cup M^{-}) \diamond$$

Following an established tradition, from now on we will often be omitting the default (negative) atoms when describing interpretations and models.

3.2 Program Updates

Suppose that $\mathcal{K}$ is an arbitrary set of propositional variables, and P and U are two generalized logic programs in the language $\mathcal{L} = \mathcal{L}_{\mathcal{K}}$. By $\widehat{\mathcal{K}}$ we denote the following superset of $\mathcal{K}$:

$$\widehat{\mathcal{K}} = \mathcal{K} \cup \{A^{-},\, A_P,\, A_P^{-},\, A_U,\, A_U^{-} : A \in \mathcal{K}\}.$$

This definition assumes that the original set $\mathcal{K}$ of propositional variables does not contain any of the newly added symbols of the form $A^{-}, A_P, A_P^{-}, A_U, A_U^{-}$ so that they are all disjoint sets of symbols. If $\mathcal{K}$ contains any such symbols then they have to be *renamed* before the extension of $\mathcal{K}$ takes place. We denote by $\widehat{\mathcal{L}} = \mathcal{L}_{\widehat{\mathcal{K}}}$ the extension of the language $\mathcal{L} = \mathcal{L}_{\mathcal{K}}$ generated by $\widehat{\mathcal{K}}$.

Definition 2 (Program Updates) *Let P and U be generalized programs in the language $\mathcal{L}$. We call P the original program and U the updating program. By the update of P by U we mean the generalized logic program $P \oplus U$, which consists of the following clauses in the extended language $\widehat{\mathcal{L}}$:*

(RP) Rewritten original program clauses:

$$A_P \leftarrow B_1, \dots, B_m, C_1^-, \dots, C_n^- \tag{4}$$
$$A_P^- \leftarrow B_1, \dots, B_m, C_1^-, \dots, C_n^- \tag{5}$$

for any clause

$$A \leftarrow B_1, \dots, B_m, \; not\, C_1, \dots, not\, C_n$$

respectively

$$not\, A \leftarrow B_1, \dots, B_m, \; not\, C_1, \dots, not\, C_n$$

in the original program P. The rewritten clauses are obtained from the original ones by replacing atoms A (resp. $not\, A$) occurring in their heads by the atoms A_P (resp. A_P^-) and by replacing negative premises $not\, C$ by C^-.

(RU) Rewritten updating program clauses:

$$A_U \leftarrow B_1, \dots, B_m, C_1^-, \dots, C_n^- \tag{6}$$
$$A_U^- \leftarrow B_1, \dots, B_m, C_1^-, \dots, C_n^- \tag{7}$$

for any clause

$$A \leftarrow B_1, \dots, B_m, \; not\, C_1, \dots, not\, C_n$$

respectively

$$not\, A \leftarrow B_1, \dots, B_m, \; not\, C_1, \dots, not\, C_n$$

in the updating program U. The rewritten clauses are obtained from the original ones by replacing atoms A (resp. $not\, A$) occurring in their heads by the atoms A_U (resp. A_U^-) and by replacing negative premises $not\, C$ by C^-.

(UR) Update rules:

$$A \leftarrow A_U \qquad A^- \leftarrow A_U^- \tag{8}$$

for all objective atoms $A \in \mathcal{K}$. The update rules state that an atom A must be true (resp. false) in $P \oplus U$ if it is true (resp. false) in the updating program U.

(IR) Inheritance rules:

$$A \leftarrow A_P, not\, A_U^- \qquad A^- \leftarrow A_P^-, not\, A_U \tag{9}$$

for all objective atoms $A \in \mathcal{K}$. The inheritance rules say that an atom A (resp. A^-) in $P \oplus U$ is inherited (by inertia) from the original program P provided it is not rejected (i.e., forced to be false) by the updating program U. More precisely, an atom A is true (resp. false) in $P \oplus U$ if it is true (resp. false) in the original program P, provided it is not made false (resp. true) by the updating program U.

(DR) Default rules:

$$A^- \leftarrow not\, A_P, not\, A_U \qquad not\, A \leftarrow A^- \tag{10}$$

for all objective atoms $A \in \mathcal{K}$. The first default rule states that an atom A in $P \oplus U$ is false if it is neither true in the original program P nor in the updating program U. The second says that if an atom is false then it can be assumed to be false by default. It ensures that A and A^- cannot both be true. ◇

3.3 Semantic Characterization of Program Updates

Follows a semantic characterization of update programs $P \oplus U$ by describing their stable models. This characterization shows precisely how the semantics of the update program $P \oplus U$ depends on the syntax and semantics of the programs P and U.

Let P and U be *fixed* generalized logic programs in the language $\mathcal{L}$. Since the update program $P \oplus U$ is defined in the extended language $\widehat{\mathcal{L}}$, we begin by first showing how interpretations of the language $\mathcal{L}$ can be extended to interpretations of the extended language $\widehat{\mathcal{L}}$.

Definition 3 (Extended Interpretation) *For any interpretation M of $\mathcal{L}$ we denote by $\widehat{M}$ its extension to an interpretation of the extended language $\widehat{\mathcal{L}}$ defined, for any atom $A \in \mathcal{K}$, by the following rules:*

$A^- \in \widehat{M}$ *iff* $not\, A \in M$
$A_P \in \widehat{M}$ *iff* $\exists A \leftarrow Body \in P \wedge M \models Body$
$A_P^- \in \widehat{M}$ *iff* $\exists not\, A \leftarrow Body \in P \wedge M \models Body$
$A_U \in \widehat{M}$ *iff* $\exists A \leftarrow Body \in U \wedge M \models Body$
$A_U^- \in \widehat{M}$ *iff* $\exists not\, A \leftarrow Body \in U \wedge M \models Body$.◇

We will also need the following definition:

Definition 4 *For any model M of the program U in the language $\mathcal{L}$ define:*

$Defaults[M] =$
$\{not\, A : M \models \neg Body, \forall (A \leftarrow Body) \in P \cup U\}$;
$Rejected[M] =$
$\{A \leftarrow Body \in P : \exists\, (not\, A \leftarrow Body') \in U$ *and* $M \models Body'\}$
$\cup$
$\{not\, A \leftarrow Body \in P : \exists\, (A \leftarrow Body') \in U$ *and* $M \models Body'\}$; ◇

The set $Defaults[M]$ contains default negations $not\, A$ of all *unsupported* atoms A, i.e., atoms that have the property that the body of every clause from $P \cup U$ with the head A is false in M. Consequently, negation $not\, A$ of these unsupported atoms A can be assumed by default. The set $Rejected[M] \subseteq P$ represents the set of clauses of the original program P that are *rejected* (or contradicted) by the update program U and the interpretation M.

Now we are able to describe the semantics of the update program $P \oplus U$ by providing a complete characterization of its stable models.

Theorem 1 (Stable models of update programs) *An interpretation N of the language $\widehat{\mathcal{L}} = \mathcal{L}_{\widehat{\mathcal{K}}}$ is a stable model of the update $P \oplus U$ if and only if N is the extension $N = \widehat{M}$ of a model M of U that satisfies the condition:*

$$M = Least(P \cup U - Rejected[M] \cup Defaults[M]) \diamond$$

4 Metaphorical Reasoning and DLP

In this section, we will show how the ideas behind *Dynamic Logic Programming* can be used in the context of *Metaphorical Reasoning*.

As we have seen before, a metaphorical framework can be seen as consisting of two theories (*tenor* and *vehicle*), defined in two different languages, together with a function mapping part of the language of the *vehicle* into the language of the *tenor*. For simplicity we will consider that the mapping function is defined for all elements of the *vehicle* language. Although this is usually not the case, we could, without loss of generality, extend the language of the *tenor* with those unmapped elements from the language of the *vehicle*, and extend the mapping function accordingly.

The two theories will be represented by generalized logic programs. The mapping function between the two languages will allow the construction of a function mapping theories of one language into theories of the other language. Formally we have:

Definition 5 (Metaphorical Program Mapping) *Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be two arbitrary set of propositional variables whose names do not begin with a " not". Let $\psi_{1,2}$:$\mathcal{K}_1 \rightarrow \mathcal{K}_2$ be a function, mapping elements from $\mathcal{K}_1$ into elements of $\mathcal{K}_2$. Let $\mathcal{L}_1$ (resp. $\mathcal{L}_2$) be the language obtained from $\mathcal{K}_1$ (resp. $\mathcal{K}_2$). Let $\mathcal{P}_1$ (resp. $\mathcal{P}_2$) be the set of generalized logic programs in the language $\mathcal{L}_1$ (resp. $\mathcal{L}_2$). We define the metaphorical program mapping as the function $\Psi_{1,2}$:$\mathcal{P}_1 \rightarrow \mathcal{P}_2$ such that for every $P_1 \in \mathcal{P}_1$, $\Psi_{1,2}(P_1)$ is obtained by replacing every objective atom A (resp. default atom not A) appearing in a rule of P_1 by $\psi_{1,2}(A)$ (resp. not $\psi_{1,2}(A)$).* $\diamond$

Example 2 *Let $\mathcal{K}_1$ be:*

$$\{big, object, background\}$$

and $\mathcal{K}_2$ be:

$$\left\{ \begin{array}{c} long, motif, accompaniment, \\ isometric\_motet\_element \end{array} \right\}$$

corresponding to the example from the introduction. Let P_1 be[2]*:*

$$not\, big(X) \leftarrow object(X), background(X).$$

and P_2 be

$$\begin{array}{c} long(X) \leftarrow motif(X), accompaniment(X), \\ isometric\_motet\_element(X). \end{array}$$

Let $\psi_{1,2}$:$\mathcal{K}_1 \rightarrow \mathcal{K}_2$ be defined by:

$$\begin{array}{c} \psi_{1,2}(big) = long \\ \psi_{1,2}(object) = motif \\ \psi_{1,2}(background) = accompaniment \end{array}$$

If we apply $\Psi_{1,2}$ to P_1 we obtain $\Psi_{1,2}(P_1)$:

$$not\, long(X) \leftarrow motif(X), accompaniment(X). \diamond$$

Now that we have a process to transform a theory in one language (*vehicle*) into a corresponding theory in the language of the *tenor* (possibly extended with new elements), we need a way to combine this transformed theory with the theory representing the *tenor* to obtain a final theory.

This final theory will consist of the *tenor* theory together with those rules from the transformed *vehicle* theory that are not contradicted by the rules from the *tenor*. This can be seen as a process of accommodation where the rules from the transformed *vehicle* are combined with those from the *tenor*, provided they do not generate contradictions. This process is essentially equivalent to the inertia exerted on the rules of the initial program during an update. With this in mind, the definition of Metaphorical Program Update is:

Definition 6 (Metaphorical Program Update) *Let $\mathcal{K}_1$ and $\mathcal{K}_2$ be two arbitrary set of propositional variables whose names do not begin with a " not". Let $\mathcal{L}_1$ and $\mathcal{L}_2$ be the languages obtained from $\mathcal{K}_1$ and $\mathcal{K}_2$ respectively. Let P_1 and P_2 be two generalized logic programs in the languages $\mathcal{L}_1$ and $\mathcal{L}_2$ respectively. Let $\psi_{1,2}$ be a function mapping elements from $\mathcal{K}_1$ into elements of $\mathcal{K}_2$. The metaphorical program update of P_1 by P_2 given $\psi_{1,2}$, denoted by $P_1 \odot P_2$ is given by $\Psi_{1,2}(P_1) \oplus P_2$.* $\diamond$

Example 3 *With P_1, P_2 and $\psi_{1,2}$ as in the previous example, the program $P_1 \odot P_2$ is:*

$$\begin{array}{c} long(X)^{-}_{P_1} \leftarrow motif(X), accompaniment(X). \\ long(X)_{P_2} \leftarrow motif(X), accompaniment(X), \\ isometric\_motet\_element(X). \end{array}$$

[2] Where, as usual, rules with variables stand for the set of their ground instantiations.

$$
\begin{array}{ll}
A^- \leftarrow A^-_{P_1}, not\, A_{P_2} & A^- \leftarrow A^-_{P_2} \\
A^- \leftarrow not\, A_{P_1}, not\, A_{P_2} & A \leftarrow A_{P_2} \\
A \leftarrow A_{P_1}, not\, A^-_{P_2} & not\, A \leftarrow A^-
\end{array}
$$

where A is a proposition from $\mathcal{K}_2$ and rules for A stand for their ground instances. Note that if we were to simply join the two programs, i.e. $\Psi_{1,2}(P_1) \cup P_2$, the two rules would produce a contradiction for X : $isometric\_motet\_element(X)$. If we, on the other hand, perform a metaphorical program update of P_1 by P_2, this contradiction no longer exists. From $P_1 \odot P_2$, we are able to conclude $long(X)$ for

$$
\left\{ \begin{array}{c} X : isometric\_motet\_element(X), motif(X), \\ accompaniment(X) \end{array} \right\}
$$

and $not\, long(X)$, otherwise, as intended. ⋄

We go on by describing the semantics of the metaphorical program update $P_1 \odot P_2$ by providing a complete characterization of its stable models.. This will be done by means of a fixed-point equation defining the set of rules from the transformed *vehicle* that are rejected by the *tenor*, yielding the set of rules from the transformed *vehicle* that carry over to the final theory due to inertia.

Proposition 2 (Stable models of $P_1 \odot P_2$) *An interpretation N of the language $\widehat{\mathcal{L}}_2$ is a stable model of the metaphorical program update $P_1 \odot P_2$ if and only if N is the extension $N = \widehat{M}$ of a model M of P_2 that satisfies the condition:*

$$
\begin{aligned}
M = Least(&\Psi_{1,2}(P_1) \cup P_2 - Rejected[M] \cup \\
&\cup\ Defaults[M])
\end{aligned}
$$

where $Rejected[M]$ and $Defaults[M]$ are as in Def.4, replacing P with $\Psi_{1,2}(P_1)$. ⋄

The set $Defaults[M]$ contains default negations $not\, A$ of all *unsupported* atoms A, i.e., atoms that have the property that the body of every clause from $\Psi_{1,2}(P_1) \cup P_2$ with the head A is false in M. Consequently, negation $not\, A$ of these unsupported atoms A can be assumed by default. The set $Rejected[M] \subseteq \Psi_{1,2}(P_1)$ represents the set of transformed clauses of the *vehicle* program P_1 that are *rejected* (or contradicted) by the *tenor* program P_2 and the interpretation M.

5 Examples and Properties

In this section we will present a more elaborate example, based on the running example, and discuss some important characteristics of metaphorical program updates. We end the section with some considerations about the possible sources of contradiction within the presented framework.

Example 4 *Consider the following generalized logic program, representing some knowledge about the domain of Visual Arts*[3], *P_1:*

$$
\begin{aligned}
not\, big(X) &\leftarrow object(X), background(X). && (\mathrm{r}_1) \\
tension(X) &\leftarrow unbalanced(X). && (\mathrm{r}_2) \\
contrast(X,Y) &\leftarrow colour(X), colour(Y), \\
&\quad high\_value\_difference(X,Y). && (\mathrm{r}_3)
\end{aligned}
$$

Now consider another generalized logic program, representing some knowledge about the domain of Music, P_2:

$$
\begin{aligned}
long(X) &\leftarrow motif(X), accompaniment(X), \\
&\quad isometric\_motet\_element(X). && (\mathrm{r}_4) \\
tension(X) &\leftarrow dissonant(X). && (\mathrm{r}_5)
\end{aligned}
$$

Let the metaphorical mapping be defined by the function $\psi_{1,2}$:$\mathcal{K}_1 \to \mathcal{K}_2$ such that:

$$
\begin{aligned}
\psi_{1,2}(big) &= long \\
\psi_{1,2}(object) &= motif \\
\psi_{1,2}(background) &= accompaniment \\
\psi_{1,2}(tension) &= tension \\
\psi_{1,2}(unbalanced) &= dissonant \\
\psi_{1,2}(colour) &= note \\
\psi_{1,2}(high\_value\_difference) &= large\_interval \\
\psi_{1,2}(contrast) &= contrast
\end{aligned}
$$

If we apply $\Psi_{1,2}$ to P_1 we obtain $\Psi_{1,2}(P_1)$:

$$
\begin{aligned}
not\, long(X) &\leftarrow motif(X), accompaniment(X). && (\mathrm{r}_6) \\
tension(X) &\leftarrow dissonant(X). && (\mathrm{r}_7) \\
contrast(X,Y) &\leftarrow note(X), note(Y), \\
&\quad large\_interval(X,Y). && (\mathrm{r}_8)
\end{aligned}
$$

Looking at the rules from $\Psi_{1,2}(P_1)$ and those from P_2, we can intuitively distinguish several paradigmatic cases:

- *rule r_6 will be a valid rule for those instances that are not covered by rule r_4 as explained during the running example;*
- *rule r_7 will not add anything to the metaphorical update for it is the same as rule r_5. This represents those cases where the translated rules from the* vehicle *are already present in the* tenor*;*
- *rule r_8 will bring not only new relations but also new concepts to the* tenor*. This represents the most interesting case with respect to creative reasoning.*

[3]The languages in which the programs are written are left implicit.

The program $P_1 \odot P_2$ is:

$$long(X)^{-}_{P_1} \leftarrow motive(X), accompaniment(X). \quad (11)$$

$$tension(X)_{P_1} \leftarrow dissonant(X). \quad (12)$$

$$contrast(X,Y)_{P_1} \leftarrow note(X), note(Y), large\_interval(X,Y). \quad (13)$$

$$long(X)_{P_2} \leftarrow motif(X), accompaniment(X), isometric\_motet\_element(X). \quad (14)$$

$$tension(X)_{P_2} \leftarrow dissonant(X). \quad (15)$$

*plus the corresponding **UR**, **IR** and **DR**. Note that rules (11) through (15), alone, are meaningless because they only have auxiliary literals ($L^{-}_{P_1}, L_{P_2}, ...$) as their conclusions and there are no rules for the literals in their premisses. It is through **UR**, **IR** and **DR** that we are able to determine the semantics of $P_1 \odot P_2$ with respect to the relevant literals. In the semantics of $P_1 \odot P_2$ we have:*

$long(X)$ *for*

$$\left\{ \begin{array}{c} X : isometric\_motet\_element(X), \\ motif(X), accompaniment(X) \end{array} \right\}$$

$not\, long(X)$ *for*

$$\left\{ \begin{array}{c} X : not\, isometric\_motet\_element(X), \\ motif(X), accompaniment(X) \end{array} \right\}$$

$contrast(X,Y)$ *for*

$$\left\{ \begin{array}{c} X,Y : note(X), note(Y), \\ large\_interval(X,Y) \end{array} \right\}$$

$tension(X)$ *for*

$$\{X : dissonant(X)\} \diamond$$

We believe it is interesting to draw the reader's attention to some properties emerging from the definition of metaphorical program updates, and their relation to known metaphor theory characteristics. The first one is related to the very basic intuition whereby metaphors bring new knowledge into the target domain. In fact, it is easy to see that in general we have that[4]:

$$SM(P_1 \odot P_2) \neq SM(P_2) \quad (16)$$

$$SM(P_1) \neq SM(P_2 \odot P_1) \quad (17)$$

The second and very important characteristic is that of directionality which, as explained before, means that each domain has a different role and its interchange, although possibly yielding an equally valuable metaphor, will not lead to the same meaning. If we have a bijective mapping function $\psi_{1,2}$ such that $\psi_{2,1} = \psi^{-1}_{1,2}$, then, in general, we have that

$$SM(P_1 \odot P_2) \neq SM(\Psi_{1,2}(P_2 \odot P_1)) \quad (18)$$

If we consider, for example, P_1 to represent a set of rules from the domain of Painting, and P_2 to represent a set of rules from the domain of Music, we could have an informal interpretation of the above properties reading as:

- a painter that becomes a musician would compose music different from that of a musician (16);
- a painter would paint differently from a musician that became a painter (17);
- a painter that becomes a musician would compose music different from that of a musician that becomes a painter (if he was to map his painting rules to music composition rules) (18).

It would be easy to check all these properties in the previous example.

In what contradiction is concerned, it is important to mention that the metaphorical program update $(P_1 \odot P_2)$ only detects and deals with inconsistencies arising from rules from different domains. Other sources of inconsistencies can exist: P_1 can be contradictory, and so can be P_2; even in cases where P_1 and P_2 are not contradictory, $P_1 \odot P_2$ can be so, this happening when the contradiction is *'latent'* in one of the domains, and is *'brought alive'* by the metaphorical update, such as in the following example:

Example 5 *Consider the following program P_1:*

$$not\, hot(X) \leftarrow blue(X)$$
$$hot(X) \leftarrow red(X)$$

the metaphorical mapping $\psi_{1,2}$:

$$\psi_{1,2}(blue) = blues$$
$$\psi_{1,2}(red) = jazz$$
$$\psi_{1,2}(hot) = hot$$

and the program P_2

$$jazz(Miles) \leftarrow \qquad blues(Miles) \leftarrow$$

the metaphorical program update $P_1 \odot P_2$ is contradictory because both $hot(Miles)$ and $not\, hot(Miles)$ are derivable. $\diamond$

Nevertheless, be they important or not for our purposes, all contradictions can be detected by inspecting the truth value of the literals A^-, A_P, A^-_P, A_U, A^-_U, etc. of the program $P_1 \odot P_2$, and dealt with either by the semantics of $P_1 \odot P_2$ itself or by other known contradiction removal techniques, e.g. Alferes et al. (1995).

[4] Where by $SM(P)$ we mean the set of stable models of the program P, restricted to the relevant language ($\mathcal{L}_1$ or $\mathcal{L}_2$ depending on the case).

6 Conclusions and Future Work

Being *Metaphor* a common device for communication that uses interrelationships between different domains to assess new enriched mixed concepts, it is, as we believe, a powerful source for modelling *Creativity*. The ability to search for solutions in distant domains, apparently unrelated to the actual problem, is determinant for our creative abilities (Guilford, 1967; De Bono, 1970). Metaphor theories, such as Veale and Keane (1993), can be used as cross-domain bridge establishment methods, fundamental for knowledge integration within different domains.

In this paper we have explored the application of *Dynamic Logic Programming* to the problem of knowledge integration in metaphorical reasoning. The problem of resolving inconsistencies that may arise when knowledge from two different domains is combined, given a metaphorical mapping, is crucial, be it at the stage where we want to evaluate the appropriateness of the mapping function, or at a subsequent stage when we want to reason with the combined knowledge.

Quite interestingly, this combined knowledge becomes a new third domain which is not a crude sum of the original ones, but a blend of concepts and relationships among them which, in some cases, can yield potentially creative outcomes, much in the line of Turner and Fauconnier (1995).

We have proposed, in a formal and rigorous manner, a transformation that, by employing the principle of inertia on the rules of the *vehicle*, solves the problem of interdomain inconsistencies. We have also characterized the models of the combined theory, all of this by means of the notions of *Dynamic Logic Programming*. Since DLP has also been implemented as a meta-interpreter (DLP System, 1998) running under XSB System (1999), this allows for not only theoretical but also practical reasoning.

This work is part of an ongoing larger project, *Dr. Divago*, whose final goal is to develop a system to perform automatic creative metaphorical reasoning.

In what future work is concerned, besides the integration of this framework within *Dr. Divago*, we are exploring some changes in the inertia rules to allow more flexibility and expressiveness, namely by permitting predominance of the *vehicle* over the *tenor*, among other possibilities. We are also exploring the development of this framework to enable for several distinct simultaneous metaphors, possibly with some preference relations amongst themselves. The mappings associated with these metaphors could be represented by generalized logic programs and the preferences could be refined by a combination of *Updates* with *Preferences* such as in Alferes and Pereira (2000).

Acknowledgements

We would like to thank Miguel Ferrand for his suggestions. The work of J. A. Leite was partially supported by PRAXIS XXI scholarship no. BD/13514/97. The work of J. A. Leite and L. M. Pereira was partially supported by PRAXIS XXI project MENTAL.

References

J. J. Alferes and C. V. Damásio and L. M. Pereira, *A Logic Programming System for Non-monotonic Reasoning*, Journal of Automated Reasoning (14):93-147, 1995

J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinski and T. C. Przymusinski. *Dynamic Logic Programming*. In A. Cohn, L. Schubert and S. Shapiro (eds.), Procs. of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, pages 98-109. Morgan Kaufmann, June 1998.

J. J. Alferes, J. A. Leite, L. M. Pereira, H. Przymusinski and T. C. Przymusinski. *Dynamic Updates of Non-Monotonic Knowledge Bases*. To appear in The Journal of Logic Programming, 2000.

J. J. Alferes and L. M. Pereira. *Updates plus Preferences*. Technical Report, Dept. de Informática, New University of Lisbon, Portugal, 2000

Barnden, J. A. *An AI system for metaphorical reasoning about mental states in discourse*. In J-P. Koenig (Ed.), Conceptual Structure, Discourse and Language II, Stanford, Ca. 1997.

M. Black. *Models and Metaphor: studies in language and philosophy*. Ithaca, NY: Cornell University Press, 1962.

Edward De Bono. *Lateral Thinking: Creativity Step by Step*. Harper & Row, New York. 1970.

The DLP System, available from `centria.di.fct.unl.pt/~jja/updates`.

Umberto Eco. *Semiotics and the Philosophy of Language*. London: Macmillan Press. 1984.

Turner, M. and G. Fauconnier. (1995). *Conceptual Integration and Formal Expression*, Journal of Metaphor and Symbolic Activity 10(3).

Gentner, D., Falkenheiner, B. and Skorstad, J. Metaphor: *The Good, The Bad and The Ugly*. In Y. Wiks(Ed.), Theoretical Issues in Natural Language Processing. Hillsdale, NJ: Lawrence Erlbaum Associates. 1989.

J. P. Guilford. *The Nature of Human Intelligence*. New York: McGraw-Hill.1967.

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

M. Gelfond and V. Lifschitz. *Classsical negation in logic programs and disjunctive databases*. New Generation Computing, 9:365-385, 1991.

B. Indurkhya. *Metaphor and Cognition*, Kluwer Academic Publishers, Dordrecht, 1992.

G. Lakoff and M. Johnson. *Metaphors We Live By*. Chigaco, Illinois: University of Chicago Press. 1980.

João A. Leite. *Logic Program Updates*. M.Sc. Dissertation, Universidade Nova de Lisboa, Portugal, 1997.

J. A. Leite and L. M. Pereira. *Generalizing updates: from models to programs*. In J.Dix, L.M. Pereira and T.C.Przymusinski (eds), Selected extended papers from the LPKR'97: ILPS'97 workshop on Logic Programming and Knowledge Representation, pages 224-246, Springer-Verlag, LNAI 1471, 1998

J. A. Leite and L. M. Pereira. *Iterated Logic Program Updates*. In J. Jaffar (ed.), Procs. of the 1998 Joint International Conference and Symposium on Logic Programming, Manchester, England, pages 265-278. MIT Press, June 1998.

Martin, J.H. *A computational model of metaphor interpretation*. Academic Press, 1990.

Francisco C. Pereira, *Modelling Divergent Production: a multi domain approach*. In Procs. of the Thirteenth European Conference of Artificial Intelligence, ECAI'98, Brighton, UK, 1998.

Francisco C. Pereira, *Construção Interactiva de Mapas Conceptuais*, M.Sc. Dissertation, University of Coimbra, Portugal, 2000

Richards, I. A. (1936). *The Philosophy of Rhetoric*. NY: Oxford University Press, 1936.

Tony Veale (1995). *Metaphor, Memory and Meaning: Symbolic and Connectionist Issues in Metaphor Comprehension.* PhD thesis submitted to Trinity College Dublin, 1995

Tony Veale. *'Just in Time' Analogical Mapping, An Iterative-Deepening Approach to Structure-Mapping* In Procs. of the Thirteenth European Conference of Artificial Intelligence, ECAI'98, Brighton, UK, 1998.

T. Veale and M. Keane. *A Connectionist Model of Semantic Memory for Metaphor Interpretation*, presented at the 1993 Workshop on Neural Architectures and Distributed AI.1993.

P. H. Winston. *Learning and Reasoning by Analogy*. Communications of the Association for Computing Machinery, 23(12), 1980.

The XSB Group. *The XSB logic programming system, version 2.0*, 1999. Available from `http://www.cs.sunysb.edu/~sbprolog`.