Descriptive dynamic logic and its application
to reflective architectures

Carles Sierra\textsuperscript{a,\ast}, Lluis Godo\textsuperscript{b,1},
Ramon López de Mántaras\textsuperscript{a,2}, Mara Manzano\textsuperscript{b,3}

\textsuperscript{a} Artificial Intelligence Research Institute, IIIA, Spanish Council for Scientific Research, CSIC, 08193, Bellaterra, Spain
\textsuperscript{b} Departament de Lògica, Facultat de Filosofia, Universitat de Barcelona, 08028, Barcelona, Spain

Abstract

The aim of this paper is to propose dynamic logic as a common logical framework to describe and identify the most relevant formal characteristics of multi-language logical architectures (MLA) in order to investigate the expressive power of the knowledge bases that can be built upon them. In general, a MLA allows to build knowledge bases as a set of units with initial local theories written in possible different logical languages. Each unit is also usually allowed to have its own intra-unit deductive system. Moreover, the whole knowledge base is equipped with an additional set of deductive rules, called \textit{bridge rules}, to control the information flow among the different units of the knowledge base. The set of bridge rules act as an inter-unit deductive system. The reasoning dynamics of a knowledge base on top of a MLA can therefore be described by how the local theories of the units evolve during execution.

Keywords: Dynamic logic; Multi language architectures; Reflection

1. Introduction

In Artificial Intelligence, multi-language logical architectures (MLA) (MC \cite{7-9}, BMS \cite{18}, ML\textsuperscript{2} \cite{5}, DESIRE \cite{19}, MILORD II \cite{1,2,16}, OMEGA \cite{4}, FOL \cite{21}) are particular types of architectures, used to build knowledge-based systems, that play a major role in dealing with complex reasoning patterns, such as those involved in non-monotonic reasoning, scheduling or planning. Despite the fact that many commonalities can be intuitively found \cite{12}, there was a lack of formal frameworks to compare and describe them. These architectures are based on the use of several logical languages to define local theories (or meta-theories acting upon theories) that influence/modify each other. These influences are modelled by complex control patterns of the reasoning flow between system units (also called modules or contexts) containing different knowledge theories evolving in time. Furthermore, the control patterns are often dynamically changed at run time. Therefore, when trying to define a formal framework to describe multi-language architectures it is mandatory that such a framework be able to model this dynamic behaviour.
Dynamic logic \[10, 11, 15\] has been traditionally used to describe and compare dynamic systems. Particularly, it has been used to deal with computational systems, understanding computations of programs as dynamic state changes. The aim of this kind of logic is the study of the mathematical properties of programs, and their behaviour. The construction of tools to reason about programs and the discovery of the key concepts involved in this reasoning process are the long term research in the field.

In this paper we present an extension of our previous work \[17\] proposing a particular theory in propositional dynamic logic (PDL) called descriptive dynamic logic (DDL) in such a way that a knowledge base built within a concrete MLA will be mapped into a theory in DDL. The atomic DDL-formulas will be taken as the quotations of the formulas of the set of languages of the architecture upon which the knowledge base is built. The key point is what a state and a program are meant to be in this DDL. In the dynamic logic terminology, a state is a complete description of which atoms are true and which atoms are false. Thus, a state for DDL will be a complete description of which formulas belong to the local theory of each unit in the knowledge base in a given moment of its execution. A change in a particular local theory of a unit will occur by the action of its local deductive system. Since programs in dynamic logic are understood in terms of transitions between states, programs in DDL will represent deductions. In particular, the set of atomic programs will correspond to the set of possible elementary deductive steps in the deduction systems (intra- or inter-unit) of the architecture, while compound programs will represent any possible control reasoning flow combining as many atomic programs as necessary. Finally, what kind of changes of local theories in units are allowed is a matter of the particular computational characteristics of each architecture. Our approach will allow to describe such characteristics in terms of compound DDL-formulas, and, on the whole, to check computational properties of a knowledge base built upon a MLA by making proofs in suitable theories in DDL.

Summarizing, the basic idea of our approach is to map multi-language knowledge-based systems (MKB) into DDL by representing:

1. elementary computational steps of knowledge bases as atomic programs in DDL,
2. local theories of the knowledge bases as formulas of DDL, and
3. operational semantics of knowledge bases as (non-logical) axioms in DDL.

Our goal is to show that execution of a knowledge base can be made equivalent to deduction in DDL. In this context DDL can be understood as a formal basis to describe multi-language architectures, as well as a specification language for knowledge bases.

The paper is structured as follows. Section 2 contains a brief reminder of PDL. Section 3 presents a formalization of MLA and related notions. In Sections 4–6 a particular theory in dynamic logic called DDL is proposed to model MLAs. A complete axiomatization of it is also provided. Finally, in Section 7 the mapping of multi-language knowledge-bases into DDL is exemplified with the description of two different reflective architectures which are a particular case of multi-language architectures.

2. Background on propositional dynamic logic

PDL is a powerful program logic used as a meta-language for talking about computer programs. A program can be seen as a dynamic object, that is, an object able to make the computer pass from one state (the content of the memory registers used by the program) to another. Due to the state change, the truth values of the formulas describing the state also change. The objective of the logic of programs is to create a logical basis to be able to express our reasoning about computer programs. The correctness of the programs, the property of halting, etc. are among the properties of programs that we may be interested to express in a logic of programs. It is also interesting to formalize general common features of programs such as WHILE, REPEAT, or IF, THEN constructs. Furthermore, it is also useful to have a deductive calculus in which to verify our reasoning about programs. PDL provides all that and uses modal logic as its basis because modal logics allow
to express changes in truth values due to changes of states. Modal models and in particular Kripke models suit PDL perfectly. The universe of the Kripke structure is now a universe of states. To each program we associate an accessibility relation in such a way that a pair of states \((s, t)\) is in that relation if and only if there is a computation of the program transforming the state \(s\) into the state \(t\). Finally, as in modal logic, each formula is interpreted as a set of states. Since we conceive a program as a binary relation between initial and final states, we associate an accessibility relation to every program, thus having a multi-modal language. PDL is not only multi-modal, but there are also operations that can be performed on programs. In fact, this is the most important feature of PDL and its axioms are not just describing the accessibility relations in itself, as it is the case with modal logics, but the operations that our program operators perform in them.

Next we briefly review the basic concepts of PDL. For a detailed description the reader is referred to [10, 11, 13].

**General syntax for PDL.** Given a set of propositional atomic variables \(\Phi_0\) and atomic programs \(\Pi_0\), the set \(\Phi\) of compound formulas and the set \(\Pi\) of compound programs of PDL are defined as:

1. \(true \in \Phi, false \in \Phi, \Phi_0 \subseteq \Phi\),
2. if \(A, B \in \Phi\) then \(\neg A \in \Phi \) and \((A \lor B) \in \Phi\),
3. if \(A \in \Phi\) and \(\alpha \in \Pi\) then \(\langle \alpha \rangle A \in \Phi\),
4. \(\Pi_0 \subseteq \Pi\),
5. if \(\alpha \in \Pi\) and \(\beta \in \Pi\) then \((\alpha \lor \beta) \in \Pi\) and \(\alpha^* \in \Pi\),
6. if \(A \in \Phi\) then \(A^? \in \Pi\).

\([\alpha]A\) is the usual modal abbreviation for \(\neg \langle \alpha \rangle \neg A\). Also \(\land, \rightarrow\) and \(\leftrightarrow\) are abbreviations with the standard meaning.

**General semantics of PDL.** The semantics of PDL is defined relative to a structure \(M\) of the form \(M = (W, \tau, \rho)\), where \(W\) is a set of states, \(\tau\) a mapping \(\tau: \Phi \rightarrow 2^W\) assigning to each formula \(A\) the set of states in which \(A\) is true, and a mapping \(\rho: \Pi \rightarrow 2^W \times W\) which assigns to each program a set of pairs \((s, t)\) representing transitions between states. More concretely, the mappings \(\tau\) and \(\rho\) are defined as follows:

- \(\tau(true) = W\)
- \(\tau(false) = \emptyset\)
- \(\tau(\neg A) = W - \tau(A)\)
- \(\tau(A \lor B) = \tau(A) \cup \tau(B)\)
- \(\tau(\langle \alpha \rangle A) = \{s \in W | \exists t, (s, t) \in \rho(\alpha)\text{ and } t \in \tau(A)\}\)
- \(\rho(\alpha; \beta) = \{(s, t) | \exists u, (s, u) \in \rho(\alpha)\text{ and } (u, t) \in \rho(\beta)\}\)
- \(\rho(\alpha \lor \beta) = \rho(\alpha) \cup \rho(\beta)\)
- \(\rho(\alpha^*) = \{(s, t) | \exists s_0, ..., s_k \text{ such that } s = s_0, s_k = t\text{ and } (s_{i-1}, s_i) \in \rho(\alpha)\text{ for all } 1 \leq i \leq k\}\)
- \(\rho(A^?) = \{(s, s) | s \in \tau(A)\}\)

Henceforth, we shall denote by \(\mathcal{G}_{\phi^\pi^\Pi}\) the class of the above standard structures for PDL. As for notions of satisfiability and validity we write:

- \((M, s) \models A\), saying the \(A\) is true in \(s\), iff \(s \in \tau(A)\),
- \((M, s) \models \Gamma\), iff \((M, s) \models A\) for all \(A \in \Gamma\),
- \(M \models A\), saying that \(A\) is \(M\)-valid, iff \((M, s) \models A\) for every \(s\) in \(M\),
- \(\Gamma \models A\), saying that \(A\) is valid, iff \(A\) is \(M\)-valid for every \(M \in \mathcal{G}_{\phi^\pi^\Pi}\).

Furthermore, a formula \(A\) is said to be a **global logical consequence** of a set of formulas \(\Gamma\) if for any structure \(M \in \mathcal{G}_{\phi^\pi^\Pi}\) we have that \(M \models A\) whenever \(M \models \Gamma\), in which case we write \(\Gamma \models_{\mathcal{G}_{\phi^\pi^\Pi}} A\). A formula \(A\) is said to be a **local logical consequence** of a set of formulas \(\Gamma\) if for any structure \(M\) and state \(s\), we have that \((M, s) \models A\) whenever \((M, s) \models \Gamma\); in which case we simply write \(\Gamma \vdash_{\mathcal{G}_{\phi^\pi^\Pi}} A\).

**Axiomatic of PDL.** The next set of axioms define the PDL [11]:

- (A1) All instances of tautologies of the propositional calculus
- (A2) \(\langle \alpha \rangle \beta \rightarrow \langle \alpha \rangle \langle \beta \rangle A\)
- (A3) \(\langle \alpha \lor \beta \rangle A \rightarrow (\langle \alpha \rangle A \lor \langle \beta \rangle A)\)
- (A4) \(\langle \alpha^* \rangle A \rightarrow (A \lor \langle \alpha \rangle A)\)
- (A5) \(\langle A^? \rangle B \rightarrow A \land B\)
- (A6) \(\langle \alpha^* \rangle (A \rightarrow [\alpha] B) \rightarrow (A \rightarrow [\alpha^*] B)\)
- (A7) \([\alpha] (A \rightarrow B) \rightarrow ([\alpha] A \rightarrow [\alpha] B)\)
The set of theorems of PDL, denoted by \( \vdash_{\text{PDL}} \), is defined as the set of axioms above plus the theorems that can be obtained from the following inference rules applied to other theorems:

\[(\text{MP}) \text{ from } \vdash_{\text{PDL}} A \text{ and } \vdash_{\text{PDL}} \neg A \implies B \text{ infer } \vdash_{\text{PDL}} B \text{ (Modus Ponens)}\]

\[(\text{G}) \text{ from } \vdash_{\text{PDL}} A \text{ infer } \vdash_{\text{PDL}} [\forall x]A \text{ (Generalization)}\]

It has been shown (see for instance [10]) that \( \vdash_{\text{PDL}} \) completely axiomatizes the set of valid formulas of the class of models \( V \), i.e. it holds that \( \vdash_{\text{PDL}} A \) if \( \vdash A \).

3. Multi-language logical architectures

The basic notion we want to express by a "MLA" is that of a logical system that provides as basic tools:

(1) a set of possibly different logical languages,
(2) a set of inference rules within each language,
(3) a set of inference rules between some of these languages,
(4) the possibility of defining a set of units containing theories written in a particular language together with a particular set of inference rules, and
(5) a set of possible interconnections of units through bridge rules, respecting some topological criteria, e.g. a binary tree structure, an acyclic graph, etc.

In order to make easier the formalization, in the next definitions we introduce, at different levels of abstraction, three concepts related to the above notion of MLA. Namely, what we call MLA (in the highest level of abstraction) represents the most general characteristics of our target computational systems. Then, a multi-language knowledge-based structure (MKB-ST) specifies a subclass of those systems by fixing its components, that is the set of units, their languages, the inference rules among them and the topology. Finally, a MKB is the specification of a concrete system obtained by filling the units of a multi-language knowledge-based structure (MKB-ST) with particular domain theories.

**Definition 1.** A multi-language logical architecture is a 4-tuple \( \text{MLA} = (L, \Delta, S, T) \), where:

1. \( L = \{L_j\}_{j \in J} \) is a set of finite logical languages.
2. \( \Delta = \bigcup_{j_1,j_2 \in J} \Delta_{j_1,j_2} \) is a set of (instances of) inference rules between pairs of languages, where \( \Delta_{j_1,j_2} \subseteq 2^{L_{j_1}} \times L_{j_2} \). In particular, when \( j_1 = j_2 \), \( \Delta_{j_1,j_2} \) denotes a set of inference rules of the corresponding language; otherwise it denotes a set of bridge rules between two different languages.
3. \( S \) is a finite set of symbols to identify units.
4. \( T \) is the set of possible topologies. Each topology is determined by a set of directed links between symbols from \( S \), i.e. \( T \) is a subset of \( 2^{S \times S} \).

Notice that we focus only on finite languages as it is the usual case in knowledge bases where some limitative rules are imposed on the generation of formulas. This fact will be essential in our approach because it will allow to express (finite) big conjunctions in DDL involving all the formulas of a language.

Notice also that it would be possible to further generalize the above definition of architecture by allowing inference rules with premises in different languages, that is, having \( \Delta = \{\Delta_{j_1,j_2,...,j_m}, \alpha_{j_1,j_2,...,j_m}^i\}_{j_1,j_2,...,j_m \in J} \), where

\[ \Delta_{j_1,j_2,...,j_m} \subseteq \bigcup_{i \in I_j} 2^{L_{j_m}} \times L_{j_1} \times ... \times L_{j_{m-1}} \times L_{j_m} \]

**Definition 2.** A multi-language knowledge-based structure \( \text{MKB-ST} \) for a given MLA is a 5-tuple \( \text{MKB-ST} = (\text{MLA}, U, \text{ML}, \text{MA}, B) \) where:

1. \( \text{MLA} = (L, \Delta, S, T) \) is a multi-language logical architecture.
2. \( U = \{u_x\}_{x \in \mathbb{X}} \) is a set of unit identifiers, i.e. \( U \) is a subset of \( S \).
3. \( \text{ML} : U \rightarrow L \) assigns a language to each unit identifier, i.e. \( \text{ML}(u) = L_j \) for some \( j \in J \), then \( \text{ML}(u) \subseteq \Delta_{j,j} \).
4. \( \text{MA} : U \rightarrow L \) assigns a set of inference rules to each unit identifier, i.e. \( \text{MA}_j : U \rightarrow \bigcup_{i \in I_j} 2^{L_{j}} \) such that if \( \text{MA}_j(u) = L_p \) for some \( j \in J \), then \( \text{MA}_j(u) \subseteq \Delta_{j,j} \).
5. \( B : U \times U \rightarrow \bigcup_{i \in I_{j_1, j_2}} 2^{L_{j_1, j_2}} \), such that:

(i) if \( u_1 \neq u_2 \), \( M_L(u_1) = L_i \) and \( M_L(u_2) = L_j \) then 
\[ B(u_1, u_2) \subseteq \Delta_{ij} \]
(ii) \( B(u, u) = \emptyset \), for any \( u \in U \),
(iii) \( \{(u_1, u_2) | B(u_1, u_2) \neq \emptyset \} \in T \), that is, the topology of MKB-ST, is in accordance with the allowed topologies in MLA.

Notice that in this definition, even in the case where \( u_1 \neq u_2 \), \( B(u_1, u_2) \) can be empty, denoting that unit \( u_1 \) has no (directed) link with the unit \( u_2 \). In this way, a unit \( u_1 \) is connected to a unit \( u_2 \) whenever \( B(u_1, u_2) \neq \emptyset \).

**Definition 3.** A multi-language knowledge-based system MKB for a given structure MKB-ST is a 3-tuple MKB = (MKB-ST, \( M_{\Sigma}, M_{\Delta} \)) where:

1. MKB-ST = (MLA, \( U, M_L, M_A, B \)) is a Multi-language knowledge-based structure, and
2. \( M_{\Sigma} \) assigns a concrete signature \( M_{\Sigma}(u) = (\text{Oper}, \text{Sort}, \text{Func}) \) for the language \( M_L(u) \) of each unit identifier \( u \), such that \( \text{Func}:\text{Oper} \rightarrow \text{Sort} \), gives a type in \( \text{Sort} \), for each element in the alphabet \( \text{Oper} \).
3. \( M_A \) assigns a set of formulas (initial local theory) built upon \( M_{\Sigma} \) to each unit identifier, i.e. \( M_A: U \rightarrow \bigcup_{\text{sort}} L_i \) such that if \( M_L(u) = L_i \) then \( M_A(u) \subseteq L_i \).

As a consequence of the execution of a MKB, the local theory associated to \( u \) will change due to the intra- and inter-unit deductions. A bridge rule \( r \in B(u_1, u_2) \) is an inference rule between unit \( u_1 \) and unit \( u_2 \) whose premises are in language \( L_i = M_L(u_1) \) and the consequent is in a (possibly different) language \( L_j = M_L(u_2) \), that is \( r \subseteq 2^{L_i} \times L_j \). Notice that the sets of bridge rules can be empty. An empty set \( B(u_1, u_2) = \emptyset \) means that no communication between units \( u_i \) and \( u_j \) is defined. An empty set \( M_A(u) = \emptyset \) means that the local theory attached to unit \( u_i \) will evolve only if inter-unit bridge rules leading to that unit are defined, otherwise the unit will remain static along the execution process of the MKB.

The distinction between architecture and knowledge base structure is in some cases not as sharp as implied by the above definitions. Often some inference rules are already fixed in the architecture (e.g. DESIRE, MILORD II), and the number and/or structure and/or role of units that can be built is also fixed (e.g. DESIRE, MILORD II, ML^2). So, these definitions have to be considered as a very general approach to MLA and MKB descriptions.

The next example illustrates the way this notation can be used to formalize the BMS architecture [18] as a MKB-ST.

**Example.** Let \( \text{MLA}_B = (L, \Delta, S, T) \), where each component is described as follows:

- \( L = \{\text{PL}^*, \text{FOL}^*\} \), where \( \text{PL}^* \) is a propositional language with negation (\( \neg \)), conjunction (\( \& \)) and implication (\( \rightarrow \)) and sentences are pairs of classical propositional formulas and truth values of the set \( \{1, 0, u\} \), meaning true, false and unknown, respectively. \( \text{FOL}^* \) is a first-order language with predicate signature \( (T, PA) \) where \( T(s) \) means that \( s \) is true and \( PA(s) \) means that \( s \) is a plausible assumption. Constants of \( \text{FOL}^* \) are quoting of propositions in \( \text{PL}^* \).
- \( \Delta = \{\Delta_{\text{PL}^*, \text{PL}^*}, \Delta_{\text{PL}^*, \text{FOL}^*}, \Delta_{\text{FOL}^*, \text{FOL}^*}, \Delta_{\text{FOL}^*, \text{PL}^*}\} \), where \( \Delta_{\text{PL}^*, \text{PL}^*} \) and \( \Delta_{\text{FOL}^*, \text{FOL}^*} \) contain the instances of *modus ponens, and-introduction*, and possibly other inference rules. The bridge rules are

\[
\Delta_{\text{PL}^*, \text{FOL}^*} = \left\{ \left( \phi, 1 \right), \left( \phi, u \right), \left( \phi, 0 \right) \right\},
\Delta_{\text{FOL}^*, \text{PL}^*} = \left\{ PA(\phi) \right\}.
\]

- \( S \) is the set of symbol identifiers used in BMS and \( T \) the possible topologies, consisting of all possible pairs of unit interconnections.

The corresponding MKB-ST of the BMS system, which is a particular structure over the above architecture, contains just a couple of units. That is, \( \text{MKB-ST}_B = (\text{MLA}, U, M_L, M_A, B) \), where

- \( U = \{a, b\} \),
- \( M_L = \{a \rightarrow \text{PL}^*, b \rightarrow \text{FOL}^*\} \),
- \( M_A = \{a \rightarrow \Delta_{\text{PL}^*, \text{PL}^*}, b \rightarrow \Delta_{\text{FOL}^*, \text{FOL}^*}\} \), and
- \( B = \{(a, b) \rightarrow \Delta_{\text{PL}^*, \text{FOL}^*}, (b, a) \rightarrow \Delta_{\text{FOL}^*, \text{PL}^*}\} \).

^5This MLA is a generalization of the BMS architecture [18].
4. DDL = Quoting MKB-STs in PDL

In this section our aim is to present our logical tools to represent and reason about the computational dynamics of multi-language knowledge-bases. According to the hierarchy of concepts introduced in Section 3, the modelling of such knowledge bases is done at two different levels.

(1) First we define an extension of PDL, called DDL to describe knowledge-base structures. This extension consists of:
   - defining a language to represent the basic components of the structure of a particular MKB-ST, that is units, languages, inference rules and topology, and
   - fixing a set of axioms, and the corresponding class of models, to describe the common behaviour of multi-language architectures.

(2) The second step to model a particular knowledge base is then to build a particular theory, in the DDL describing its structure, containing formulas representing both the formulas of its units and the inference rules used in each unit and between different units.

Our final goal is to faithfully describe the computational behaviour of the multi-language knowledge base by performing logical deduction in DDL theories, or in other words, to be able to check some properties of the multi-language system by means of proofs in DDL.

To define DDL we need first of all to fix the set of atomic formulas and the set of programs. Given a MKB-ST = (MLA, U, M_L, M_A, B), the set of atomic formulas of DDL will be defined as the set of "quoted" formulas built upon the languages L in MLA together with concrete signatures for each unit, and indexed by the unit identifiers in U. The set of DDL atomic programs will be restricted to represent deduction steps in those languages. For notation simplicity, in the following definitions the set of formulas of a unit u will be denoted by ML(u), that is without explicit reference to the corresponding signature.

Definition 4. Given a MKB-ST, the set of atomic formulas of DDL is defined as the following finite set:

$$\Phi_0 = \{ u : \varphi \mid u \in U, \varphi \in M_L(u) \}$$

that is, formulas are indexed by unit names, using the notation unit-identifier: [formula].

In the following definition, we build up the set of atomic programs $\Pi_0$ from both intra-unit inference rules and inter-unit inference rules, or bridge rules.

Definition 5. Given a MKB-ST, the set $\Pi_0$ of atomic programs of DDL is defined as the union of the intra-unit inference rules $\Pi_{0 \text{ intr}}$ and the MKB-ST bridge rules $\Pi_{0 \text{ inter}}$, i.e. $\Pi_0 = \Pi_{0 \text{ intr}} \cup \Pi_{0 \text{ inter}}$, where:

$$\Pi_{0 \text{ intr}} = \bigcup_{k \in U} \Pi_{0 \text{ intr}}^k$$

$$\Pi_{0 \text{ inter}} = \bigcup_{k \in U} \Pi_{0 \text{ inter}}^k$$

being

$$\Pi_{0 \text{ intr}}^k = \{ [ \Gamma \vdash_M \varphi ] \mid \Gamma \cup \{ \varphi \} \subseteq M_L(u_k), (\Gamma, \varphi) \in M_A(u_k) \}$$

and

$$\Pi_{0 \text{ inter}} = \{ [ \Gamma \vdash_M \varphi ] \mid \Gamma \subseteq M_L(u), \varphi \in M_L(u), (\Gamma, \varphi) \in B(u, u_1) \}$$

where $[ \Gamma \vdash_M \varphi ]$ is an abbreviation for the quoting function applied to a deduction step.

Notice that atomic programs contain not only the inference rules applied but also the formulas involved in the deduction steps. Notice also that the quoting function of DDL is similar to that of OMEGA [4]. So, having defined the quoting function for formulas, we extend it to sets of formulas and deduction as follows, where $\Gamma = \{ y_1, \ldots, y_n \}$:

$$[ \Gamma ] = \text{set}(\{ y_1, \ldots, y_n \})$$

$$[ \Gamma \vdash_M \varphi ] = \text{proof}(\{ \Gamma \}, [ \varphi ], [ k ], [ I ])$$

It is clear then that the access to components of quoted proofs is possible by means of the appropriate accessor functions.

Compound programs representing arbitrary applications of intra- and inter-unit inference rules can be built by means of indeterministic unions of atomic programs. Therefore, no commitments are done about any particular control strategy. For
instance, we define below the compound programs corresponding to the intra- and inter-unit deduction.

**Definition 6.** Given the set $\Pi_0$ of atomic programs, we define the next two sets of indeterministic compound programs standing for inter- and intra-unit deductions:

- intra-unit deduction is $\Gamma \vdash_{\text{intra}} = \bigcup \{ \alpha | \alpha \in \Pi_{0_{\text{intra}}} \}$,
- inter-unit deduction is $\Gamma \vdash_{\text{inter}} = \bigcup \{ \alpha | \alpha \in \Pi_{0_{\text{inter}}} \}$.

Notice again that given finiteness of the languages in the architectures, the set $\Pi_0$ of atomic programs is finite, and thus the above compound programs representing intra and inter deductions are well defined.

5. Syntax and semantics of DDL

5.1. The intended models of DDL

The semantics of DDL is defined, as in PDL, relative to a structure $M$ of the form $M = (W, \tau, \rho)$, as a particularization of the general semantics for PDL by adding a particular interpretation of both the DDL atomic formulas and the DDL atomic programs. The computational systems one could model in DDL is very wide, showing very different behaviours from a logical point of view. Some of these systems are non-monotonic, some are conservative, etc.

The class of models we define for DDL restrict the possible transitions between states to those satisfying a set of requirements that capture the basic properties of the deductive steps performed in multi-language architectures. The idea in mind is that performing a deductive step $\Gamma \vdash \varphi$ in any unit means two things: first, the formulas in $\Gamma$ of unit $u_k$ were previously proved, and second, the formula $\varphi$ of unit $u_1$ becomes proved. Then, the intuition behind the semantics of DDL is that states provide which formulas are proved in a particular moment of the execution of the system and therefore transitions corresponding to atomic programs, which represent deductive steps of the kind $\Gamma \vdash \varphi$, have to satisfy the following natural requirements:

1. an atomic program $\Gamma \vdash \varphi$ is executable in all states that satisfy the quoted formulas from $\Gamma$,
2. an atomic program $\Gamma \vdash \varphi$ leads to states that satisfy $\vdash \varphi$,
3. an atomic formula is true in the target state whenever it is true in the source state (that is, we have the property of monotonicity),
4. an atomic formula different from the conclusion in the program and true in the target state must also be true in the source state (that is, there are no side-effects),
5. if an atomic program is executable in a source state of another atomic program, it is also executable in the target state of that program (that is, we have a persistence property), and finally
6. atomic programs are partial functions, that is, if an atomic program is executable in a given state, there exists a unique target state (atomic programs are meant to represent a single execution of a rule of inference).

Notice that condition (5) can be extended to compound programs that do not contain test programs as components, since a test program could be applicable before the execution of an atomic program but not after. This is so because the meaning of the falsity of the quoted formula $\rho$ in a given state $s$ is that the formula has not been proved in the logic of the unit. Therefore, $\lnot \rho \text{true}$ is true in $s$. But nothing prevents $\lnot \rho \text{true}$ to become false after the application of an inference rule proving $\rho$. Henceforth we shall denote by $\Pi_{\text{free}}$ the set of programs that do not contain test programs as components.

The above requirements impose a set of constraints in the possible transitions allowed for a $\rho$ in the class of models. In what follows, for any state $s$, $f(s)$ will stand for the atomic truth set of $s$, composed of the atomic DDL-formulas which are true in $s$, i.e. $f(s) = \{ p | s \in \tau(p), p \in \Phi_0 \}$.

**Definition 7 (Class of models $\mathcal{F}^{\text{free}}$.** The class of structures $\mathcal{F}^{\text{free}}$ consists of structures $(W, \tau, \rho)$ satisfying the following property: given a set of sentences $\Gamma = \{ \psi_i | i \in I \}$ from a unit $u_k$ and a sentence $\varphi$ from a unit $u_1$ the following conditions hold for any program $\Gamma \vdash_{\text{intra}} \varphi$:
(C1) if \( s \in \bigcap_{l \in \ell} \tau(k;[\psi_i]) \) then there exists \( t \) such that
\( (s,t) \in \rho([\Gamma \vdash_{kl} \varphi]) \).

(C2) if \( (s,t) \in \rho([\Gamma \vdash_{kl} \varphi]) \) then \( s \in \bigcap_{l \in \ell} \tau(k;[\psi_i]) \) and
\( t \in \tau(l;[\varphi]) \).

(C3) if \( (s,t) \in \rho([\Gamma \vdash_{kl} \varphi]) \) then if \( s \in \tau(m;[\psi]) \) then
\( t \in \tau(m;[\psi]) \) for all \( \psi \in M_L(u_{mk}) \).

(C4) if \( (s,t) \in \rho([\Gamma \vdash_{kl} \varphi]) \) then \( t \in \tau(p) \) then \( s \in \tau(p) \) for all \( p \in \Phi_0 \) and \( p \neq l;[\varphi] \).

(C5) if \( (s,t) \in \rho([\Gamma \vdash_{kl} \varphi]) \) then if \( (s,t') \in \rho(\beta) \) then there exists \( r \) such that \( (t,r) \in \rho(\beta) \) for all
\( \beta \in \Pi_{\text{free}} \).

(C6) if \( (s,t) \in \rho([\Gamma \vdash_{kl} \varphi]) \) then if \( (s,t') \in \rho(\beta) \) then \( t = t' \).

We will use the notations \( \models_{\mathcal{G}^{\mathcal{P}}} A \) and \( \Delta \models_{\mathcal{G}^{\mathcal{P}}} A \) to say that \( A \) is a valid formula in the class of models \( \mathcal{G}^{\mathcal{P}} \), and that \( A \) is a logical consequence of a set of formulas \( \Delta \) in \( \mathcal{G}^{\mathcal{P}} \), respectively.

To have an idea about what kind of formulas can be proved to be valid in \( \mathcal{G}^{\mathcal{P}} \), let us consider the case where a unit \( u_k \) of a multi-language logical system is endowed with the modus ponens inference rule. The behaviour of this inference rule can be described by the following DDL formula:

\[ MP: \langle \beta \rangle (k;[\varphi] \land k;[\varphi \rightarrow \psi]) \rightarrow \langle \beta \rangle \models_{\mathcal{k}} k;[\psi] \]

where the modus ponens program \( \models_{\mathcal{k}} \) is defined as the following indeterministic finite union:

\[ \models_{\mathcal{k}} = \bigcup_{\gamma,\delta \in \tau(M(u_k))} \{ \gamma, \delta \rightarrow \delta \} \models_{\mathcal{k}} \delta \].

The above formula \( MP \) says that if it is possible to prove \( \varphi \) and \( \varphi \rightarrow \psi \), after some previous deduction (represented by the program \( \beta \)), then it is also possible to prove \( \psi \) after \( \beta \) followed by an application of the modus ponens inference rule. Notice that, since we can only have atomic programs corresponding to instances of deduction steps, the program standing for the modus ponens inference rule is actually defined as the finite union of those atomic programs corresponding to all possible particular instantiations of the rule in the unit \( u_k \). Then, it is easy to show that the formula \( MP \) is valid in \( \mathcal{G}^{\mathcal{P}} \).

The following formula schemes will become the axioms of DDL.

(1) DED-1: \( \langle [\Gamma \vdash_{ij} \varphi] \rangle \models true \iff \bigwedge_{\varphi \in \Gamma} \langle \varphi \rangle \]

(2) DED-2: \( \langle [\Gamma \vdash_{ij} \varphi] \rangle \models true \iff \bigwedge_{\varphi \in \Gamma} \langle \varphi \rangle \]

(3) MON: \( p \models [\alpha]p \) for \( p \in \Phi_0 \)

(4) SEA: \( \langle [\Gamma \vdash_{ij} \varphi] \rangle \models p \) for \( p \in \Phi_0 \) and \( p \neq j;[\varphi] \)

(5) PER: \( \langle \alpha \rangle \models true \land \langle \beta \rangle \models true \iff \langle \alpha;\beta \rangle \models true \) for \( \beta \in \Pi_{\text{free}} \)

(6) PFUN: \( \langle x \rangle A \models [\alpha]A \) for \( x \in \Pi_0 \).

We will see now, in Theorem 1, that they exactly correspond to the previous conditions (C1)-(C6) imposed over the class of models \( \mathcal{G}^{\mathcal{P}} \), except for the PFUN schema for which we can only prove its validity.

**Theorem 1.** Let \( M = (W, \tau, \rho) \) belong to the class \( \mathcal{G}^{\mathcal{P}} \) of standard PDL models. Then the following conditions are verified:

(1) \( M \) satisfies (C1) and (C2) iff DED-1 and DED-2 are valid in \( M \);

(2) \( M \) satisfies (C3) iff MON is valid in \( M \);

(3) \( M \) satisfies (C4) iff SEA is valid in \( M \);

(4) if \( \rho(\alpha) \) is partially functional for any atomic program \( \alpha \), then \( M \) satisfies (C5) iff PER is valid in \( M \);

(5) PFUN is valid in \( M \) if \( M \) satisfies (C6).

**Proof.**

(1) \( M \) satisfies (C1) and (C2) iff DED-1 and DED-2 are valid in \( M \): Let \( \beta \equiv [\Gamma \vdash_{ij} \varphi] \), DED-1 is valid in \( M \) iff for all \( s \) in \( M \), \( (M,s) \models [\beta]true \iff \bigwedge_{\varphi \in \Gamma} \langle \varphi \rangle \]

It is clear that condition (C1) corresponds to the right to left implication in relation (1). Now DED-2 is valid in \( M \) iff for all \( s \) in \( M \), \( (M,s) \models [\beta]true \iff \bigwedge_{\varphi \in \Gamma} \langle \varphi \rangle \]

It is easy to check that condition (C2) corresponds to both relation (2) and the left to right implication in relation (1). Altogether we finally get that both conditions (C1) and (C2) hold iff both relations (1) and (2) hold.

(2) \( M \) satisfies (C3) iff MON is valid in \( M \): MON is valid in \( M \) iff for every \( s \) in \( M \), \( (M,s) \models p \rightarrow [\alpha]p \)
(for $p \in \Phi_0$) if $\forall s(\sigma(t(p) = \forall t((s, t) \in \rho(x)) \Rightarrow t \in \tau(p)))$ iff C3.

(3) $M$ satisfies (C4) iff $SEA$ is valid in $M$: Let $\beta \equiv \{\Gamma \vdash \varphi\}$. $SEA$ is valid in $M$ iff for every $s$ in $M$, $(M, s) \vdash \langle \beta \rangle p \rightarrow p$ (where $p \neq j(\varphi)$) iff $\forall s(\exists t((s, t) \in \rho(\beta) \land t \in \tau(p)))$ if $s \in \tau(p)$) iff C4.

(4) Suppose $\rho(\alpha)$ is partially functional, $x \in \Pi_0$, i.e., if $(s, t)$ and $(s, t')$ belong to $\rho(\alpha)$ then $t = t'$ and $\beta \in \Pi^\alpha_{\text{free}}$. In this case, we have:

$\text{PER}$ is valid in $M$ iff $(M, s) \vdash \langle x \rangle \text{true} \land \langle \beta \rangle \text{true}$

iff $\exists t((s, t) \in \rho(x))$ and $\exists t'(s, t') \in \rho(\beta)$ implies $\exists t_1, t_2((s, t_1) \in \rho(x) \land (t_1, t_2) \in \rho(\beta))$

but, since $\rho(\alpha)$ is partially functional, $t_1 = t$ and therefore $M$ satisfies (C5). On the other hand, if $M$ satisfies (C5) it is easy to show that $\text{PER}$ is valid.

(5) Straightforward.

This completes the proof. □

Theorem 1 gives the validity of the above schemes in the class $\mathcal{G}_{\rho^\alpha\varphi}$ since any model $M = (W, \tau, \rho) \in \mathcal{G}_{\rho^\alpha\varphi}$ satisfies (C6), and therefore, $\rho(\alpha)$ is partially functional for all atomic programs $x$, and thus $\text{PER}$ is valid in $M$.

**Corollary 1.** Let $M = (W, \tau, \rho)$ be a model in $\mathcal{G}_{\rho^\alpha\varphi}$ for which $\rho(\alpha)$ is partially functional for any atomic program $x$. Then, it follows that $\text{DED-1, DED-2, MON, SEA and PER}$ are valid in $M$ iff $M \in \mathcal{G}_{\rho^\alpha\varphi}$. As a consequence, the schemes $\text{DED-1, DED-2, MON, SEA, PER and PFUN}$ are valid in the class of models $\mathcal{G}_{\rho^\alpha\varphi}$.

### 5.2. Axiomatic of DDL

In this section we provide an axiom system for DDL, and prove its soundness and completeness with respect to the previously defined class of models $\mathcal{G}_{\rho^\alpha\varphi}$.

**Definition 8.** Given a multi-language knowledge-base Structure, the corresponding DDL logic is defined as the following extension of PDL:

- The language of DDL is defined upon the sets of atomic formulas $\Phi_0$ and atomic programs $\Pi_0$ as given in Definitions 4 and 5, respectively.
- The additional axioms of DDL are the schemes $\text{DED-1, DED-2, MON, SEA, PER and PFUN}$.

**Theorem 2 (Soundness and completeness).** DDL is sound and complete with respect to the class of models $\mathcal{G}_{\rho^\alpha\varphi}$.

**Proof.**

(1) **Soundness.** Let $\vdash_{\text{DDL}} \varphi$ (i.e. $\varphi$ is a theorem of the logic DDL). Thus, DDL $\vdash_{\text{PDL}} \varphi$ (where DDL is the set of instances of the scheme axioms of the logic DDL). Therefore $\vdash_{\text{PDL}} \delta_1 \land \ldots \land \delta_n \rightarrow \varphi$, where the formulas $\delta_1, \ldots, \delta_n$ are the axioms in DDL used in the deduction. The number is finite because of the finiteness of the deduction, and we use the deduction theorem: $\psi \vdash \pi \vdash \psi \rightarrow \pi$. Then $\vdash_{\text{PDL}} \delta_1 \land \ldots \land \delta_n \rightarrow \varphi$, because PDL is sound (see [10]). Then, $\vdash_{\text{DDL}} \delta_1 \land \ldots \land \delta_n \rightarrow \varphi$, because $\mathcal{G}_{\rho^\alpha\varphi} \subseteq \mathcal{G}_{\rho^\alpha\varphi}$. Finally, $\vdash_{\text{DDL}} \varphi$, because $\delta_1, \ldots, \delta_n$ are valid in DDL by Corollary 1.

(2) **Completeness.** Let $\vdash_{\text{PDL}} \varphi$. Thus, DDL $\vdash_{\text{PDL}} \varphi$ (by Corollary 1). Therefore, $\vdash_{\text{PDL}} \delta_1 \land \ldots \land \delta_n \rightarrow \varphi$, where DDL' is a finite set of axioms obtained from the scheme axioms in DDL. We put in DDL' all the instances of DED-1, DED-2 and SEA. To reduce the instances of MON, PER and PFUN to a finite number we use the Fischer-Ladner closure. Then, $\vdash_{\text{DDL}} \delta \rightarrow \varphi$, by completeness of PDL (see [10]). Thus, DDL' $\vdash_{\text{PDL}} \delta_1 \land \ldots \land \delta_n \rightarrow \varphi$, using the propositional rules of the calculus. Then, DDL' $\vdash_{\text{PDL}} \varphi$, by monotonicity (DDL' $\subseteq$ DDL). Finally, $\vdash_{\text{DDL}} \varphi$, since the logic DDL is obtained when adding to PDL the axioms in DDL.

This completes the proof. □

### 5.3. Representing deductive closures in DDL

Since, in our framework, programs represent deductions, it is interesting to see how to define, for any program $\alpha$, the program representing the deductive closure of $\alpha$. These programs are easily defined in PDL as

$$\alpha^c = \bigwedge_{\varphi \in \alpha^*} \left( \langle \alpha^* \rangle \varphi \rightarrow \varphi \right)$$

where $\alpha^c$ represents the closure of program $\alpha$, meaning that this program will lead to a state in which no different state is reachable by another application of program $\alpha$. Some intuitive properties that $\alpha^c$ should
verify can be actually proved. But before doing that, we need some previous results.

**Proposition 1.** Let \((W, \tau, \rho) \in \mathcal{E}'_{\text{free}}\). Then the following condition holds for any \(a, b \in \Pi_{\text{free}}\): if \((s, t) \in \rho(a)\) and \((s, t) \in \rho(b)\) then there exists \(r\) such that \((t_1, r) \in \rho(b)\) and \((t_2, r) \in \rho(a)\).

**Proof.** Let \(a, b \in \Pi_{\text{free}}\). The proof will follow by induction on the structure of the programs \(a\) and \(b\).

1. **Atomic case:** Let \(a = \langle \cdot \rangle p\) and \(b = \langle \cdot \rangle q\). Then by condition (C5), there exists \(r_1\) such that \((t_1, r_1) \in \rho(\beta)\) and \(f(r_1) = f(s) \cup \{t: \exists t_1 : t_1 = \rho(p)\}\). Analogously, there exists \(r_2\) such that \((t_2, r_2) \in \rho(\alpha)\) and \(f(r_2) = f(s) \cup \{t: \exists t_1 : t_1 = \rho(q)\}\). So, \(r_1 \) and \(r_2 \) denote the same state.

2. **Let \(a = \delta_1 \cup \delta_2\), being \(\delta_1, \delta_2 \in \Pi_{\text{free}}\), and suppose that \(\delta_1\) and \(\delta_2\) satisfy the condition. Since \((s, t) \in \rho(\delta_1)\) and \((s, t) \in \rho(\delta_2)\), there exists \(u_1\) such that \((s, u_1) \in \rho(\delta_1)\) and \((s, t) \in \rho(\delta_2)\). By induction hypothesis, it is easy to show that there exist \(u_2\) and \(r\) such that \((s, u_2) \in \rho(\beta)\) and \((t_1, r) \in \rho(\delta_1)\) and \((t_2, r) \in \rho(\delta_2)\). Therefore, we have proved that \((t_1, r) \in \rho(\beta)\) and \((t_2, r) \in \rho(\beta)\).

3. **Let \(a = \delta_1 \cup \delta_2\), being \(\delta_1, \delta_2 \in \Pi_{\text{free}}\), and suppose that \(\delta_1\) and \(\delta_2\) satisfy the condition. Let \((s, t) \in \rho(\delta_1)\) and \((s, t) \in \rho(\delta_2)\). Assume \((s, t) \in \rho(\delta_1)\) then by the induction hypothesis there exists \(r\) such that \((t_1, r) \in \rho(\beta)\) and \((t_2, r) \in \rho(\delta_1)\) and therefore \((t_2, r) \in \rho(\delta_2)\). The case \((s, t) \in \rho(\delta_2)\) is analogous.

4. **Let \(a = \delta_1 \cup \delta_2\), being \(\delta_1, \delta_2 \in \Pi_{\text{free}}\), and suppose that \(\delta_1\) and \(\delta_2\) satisfy the condition. Let \((s, t) \in \rho(\alpha)\) and \((s, t) \in \rho(\beta)\). There is a finite sequence of applications of \(\alpha\) leading from \(s\) to \(t_1\). In each intermediate step we can apply the induction hypothesis getting a new connecting chain by \(\alpha\) of states leading from \(t_2\) to \(r\), and a step by \(\beta\) from \(t_1\) to \(r\).

The rest of the cases are proved in a similar way.

As a particular case of this proposition, by making \(\alpha = \beta\) we get that the transition relations of test-free programs are weakly directed. A binary relation \(R\) is weakly directed if when \((s, t_1), (s, t_2) \in R\) there exists \(r\) such that \((t_1, r), (t_2, r) \in R\). It is known that in modal logics the axiom corresponding to this kind of relations is \(\Box \square \varphi \rightarrow \square \Box \varphi\), and is called axiom \(G\). The validity of this axiom in DDL is expressed also in Corollary 2.

**Corollary 2.** If \((W, \tau, \rho) \in \mathcal{E}'_{\text{free}}\) then \(\rho(\alpha)\) is weakly directed for all \(\alpha \in \Pi_{\text{free}}\); therefore, the formula

\[ G : \langle \alpha \rangle [a]A \rightarrow [a] \langle \alpha \rangle A \]

is valid in \(\mathcal{E}'_{\text{free}}\) for all \(\alpha \in \Pi_{\text{free}}\).

**Proof.** The first part is straightforward. Let \((W, \tau, \rho) \in \mathcal{E}'_{\text{free}}\). We want to prove that \(\tau(\langle \alpha \rangle [a]A) \leq \tau(\langle [a] \langle \alpha \rangle A)\). Let \(s \in \tau(\langle [a] \langle \alpha \rangle A)\), then there exists \(t\) such that \((s, t) \in \rho(\alpha)\) and \(t \in \tau(\langle [a] A)\). Let \((s, t) \in \rho(\alpha)\). We want to find \(r\) such that \((t, r) \in \rho(\alpha)\) and \(r \in \tau(A)\). By the first part of the corollary \(\rho(\alpha)\) is weakly directed so, there exists \(r\) such that \((t, r), (t', r) \in \rho(\alpha)\). But \(t \in \tau([a] A)\), so \(r \in \tau(A)\).

**Theorem 3.** The following formulas are valid in \(\mathcal{E}'_{\text{free}}\):

1. \(\langle \alpha^* \rangle p \rightarrow [a^*]p, \quad \forall \alpha \in \Pi_{\text{free}}, \forall p \in \Phi_0\)
2. \(\langle \alpha \rangle p \rightarrow [a^*]p, \quad \forall \alpha \in \Pi_{\text{free}}, \forall p \in \Phi_0\)

**Proof.**

1. Let \(s \models \langle \alpha \rangle p\), i.e. there exists \(t_0\) such that \((s, t_0) \in \rho(\alpha^*)\) and \(t_0 \models p\). We have to prove that \(s \models [a^*]p\), that is, for all \(t, s \in \rho(\alpha^*)\) it holds \(t \models p\). Notice that \((s, t) \in \rho(\alpha^*)\) if \((s, t) \in \rho(\alpha^*)\) and \(t \models \psi\). By hypothesis, \(t \models \psi\). Hence it is the case that \(t \models p\).

2. Trivial by noticing that \(\langle \alpha \rangle A \rightarrow [a^*]A\) is a valid formula in PDL.

**Lemma 1.** The following are provable formulas in PDL:

1. \(\langle \alpha \rangle \text{true} \leftrightarrow ([a]A \rightarrow \langle \alpha \rangle A)\)
2. \(\langle a^* \rangle A \rightarrow \langle \alpha^* \rangle A\).
Proof. (1) It is a standard result in modal logic. For (2) we prove the following successive implications, where $B$ is a shorthand for $\langle a^*; B ? \rangle p$:

\[ \langle a^* \rangle B ? p \rightarrow \langle a^*; B ? \rangle p, \]
by definition,

\[ \langle a^*; B ? \rangle p \rightarrow \langle a^* \rangle (B \wedge p), \]
by axiom (A2),

\[ \langle a^* \rangle (B \wedge p) \rightarrow \langle a^* \rangle B \wedge \langle a^* \rangle p, \]
by standard modal calculus,

\[ \langle a^* \rangle B \wedge \langle a^* \rangle p \rightarrow \langle a^* \rangle p, \]
by propositional calculus.

This completes the proof. \(\square\)

Theorem 4. DDL proves

\[ \langle a \rangle \text{true} \rightarrow \langle a^* \rangle p \rightarrow [a^*]p \]
for $a \in \Pi_{\text{free}}$ and for $p \in \Phi_0$.

Proof. Easy from Theorem 3(1), and using Lemma 1. \(\square\)

6. Describing reflective knowledge-based systems with DDL.

In this section we describe, by means of DDL, the reasoning dynamics of the reflective architecture BMS [19], and a reflective component of the GET language [6]. In the BMS system the reflection mechanism is declarative, in the sense that formulas in a unit are reified by bridge rules into another unit, which plays the role of a meta-level reasoner. The formulas deduced at this meta-level unit are then reflected back. In the case of GET we describe its program tactics mechanization, which is based on a set of primitive tactics that GET represents as names denoting code, plus a set of tacticals that combine the primitive ones to generate complex proof strategies. The reflection consists in the semantic attachment between names (and terms) of tactics and the code associated to them.

6.1. Example 1: BMS reasoning dynamics

The inference system of the structure of the BMS system, presented in the example of Section 3, can be described in DDL as a set of formulas. Some of them, that will be needed later, are the following ones (remember that $a$ is the unit identifier for the object unit and $b$ for the meta unit):

\[ \langle a \rangle \langle b : [\psi \land b : [\psi] \rightarrow \langle a ; \Leftrightarrow b : [\psi] \rangle \]

\[ \langle a \rangle \langle b : [\psi \rightarrow \psi] \land b : [\psi] \rightarrow \langle a ; \Leftrightarrow b : [\psi] \rangle \]

\[ \langle a \rangle a : [(\varphi, 1)] \rightarrow a : \Leftrightarrow b : [T(\varphi)] \]

\[ \langle a \rangle a : [(\varphi, u)] \rightarrow a : \Leftrightarrow b : [\sim T(\varphi)] \]

\[ \langle a \rangle a : [(\varphi, 0)] \rightarrow a : \Leftrightarrow b : [\sim T(\varphi)] \]

\[ \langle a \rangle a : [(\varphi, 1)] \rightarrow a : \Leftrightarrow b : [\sim T(\varphi)] \]

Now consider the well-known tweety example as the particular knowledge base MKB_{TWEETY}, over the above structure, defined as the 3-tuple $(MKB_{BTCMS}, M_\Sigma, M_\Omega)$, where:

(1) $M_{\Sigma}(a) = \{\{\text{penguin}, \text{bird}, \text{flies}\}, \{\text{Prop}\}, \{\text{penguin} \rightarrow \text{Prop}, \text{bird} \rightarrow \text{Prop}, \text{flies} \rightarrow \text{Prop}\}$,
(2) $M_\Omega(b) = \{\{\text{penguin}, \text{bird}, \text{flies}\}, \{\text{Constant}\}, \{\text{penguin} \rightarrow \text{Constant}, \text{bird} \rightarrow \text{Constant}, \text{flies} \rightarrow \text{Constant}\}$, where $\{\text{penguin}, \text{bird}, \text{flies}\}$ are a set of constants representing the propositions in $M_{\Sigma}(a)$,
(3) $M_\Omega(a) = \{\{\text{penguin} \Rightarrow \sim \text{flies}, 1\}, \{\text{penguin} \Rightarrow \text{bird}, 1\}\}$, and
(4) $M_\Omega(b) = \{\sim T(\text{penguin})\&T(\text{bird}) \Rightarrow PA(\text{flies})\}$.

This knowledge base is mapped into the following DDL theory which we call it $R_{TWEETY}$:

\[ a : [(\text{penguin} \Rightarrow \sim \text{flies}, 1)] \]
\[ a : [(\text{penguin} \Rightarrow \text{bird}, 1)] \]
\[ b : [\sim T(\text{penguin})\&T(\text{bird}) \Rightarrow PA(\text{flies})] \]

To prove in $R_{TWEETY}$ that from knowing bird and not knowing penguin we get flies is equivalent to prove the next formula in DDL:

\[ R_{TWEETY} \cup \{a : [(\text{bird}, 1)], a : [(\text{penguin}, u)]\} \]
\[ \vdash_{\text{DDL}} \text{BMS}_{\text{Control}} a : [(\text{flies}, 1)] \]

which means that after the execution of the BMS system we get to a state in which the formula flies is true. The implicit control in BMS system, represented by the compound program $BMS_{\text{Control}} = (\sim \sim \Leftrightarrow c) \Leftrightarrow \sim \Leftrightarrow c) \Leftrightarrow \sim \Leftrightarrow c)$ means that the control of BMS makes all possible deductions at unit $a$, reflects up to unit $b$ all formulas in $a$, makes all
possible deductions in unit $b$ and finally reflects down to unit $a$, which can be proved, as the next proof tree shows. In this proof we repeatedly use Theorem 3(2). In the proof, only the modus ponens in DDL is used.

$$a:[(bird, 1)] \quad a:[(penguin, u)]$$

$$(1) \quad [\neg_{-ab} \land \neg_{-ba}] b:[T(bird) \land \neg T(penguin)]$$

$$(2) \quad [\neg_{-ab} \land \neg_{-ba}] b:[T(bird) \land \neg T(penguin)]$$

$$(3) \quad [\neg_{-ab} \land \neg_{-ba}] b:P\alpha(flies)$$

$$(4) \quad [\neg_{-ab} \land \neg_{-ba}] a:[(flies, 1)]$$

$$(5) \quad [\neg_{-ab} \land \neg_{-ba}] a:[(flies, 1)]$$

Similarly, it can be proved that

$$R_{\text{Tweety}} \cup \{a:[(bird, 1)], a:[(penguin, 1)]\}$$

$$\vdash_{\text{DDL}} \text{BMS}_{\text{Control}} a:[(\neg flies, 1)]$$

6.2. Examples 2: Tactics in GET

We model here tactics provided in GET [6] as compounds programs in DDL. For each tactical $T$ in GET we define a compound program $T^{\text{DDL}}$ with an equivalent behaviour. To make more precise what we want to say by equivalent behaviour, we need first to fix the meaning of three different concepts related to tactics, as they are understood in GET.

(1) **Tactical failure.** In GET a tactic (or tactical) $F_T$ is a function that is said to fail for a given set of arguments $\text{args}$ if the function is not defined for $\text{args}$; where $\text{args}$ are a set of formulas. In such cases, the function defining a tactical in GET, $T$, is constructed by extending the original one returning the symbol “fail” for the undefined cases.

$$T(\text{args}) = \begin{cases} \text{fail} & \text{if } F_T(\text{args}) \text{ is undefined,} \\ F_T(\text{args}) & \text{otherwise.} \end{cases}$$

In DDL the meaning of program failure in a state $s$ can be understood as $s \vdash_{\text{DDL}} \neg (T^{\text{DDL}})_{\text{true}}$.

That is, the program is not applicable in that state. So we can set the next relation between GET and DDL:

$$T(\text{args}) = \text{fail} \iff [\text{args}] \vdash_{\text{DDL}} \perp_{\text{true}}$$

where $\perp_{\text{true}}$ is a shortcut for $[T^{\text{DDL}}]_{\text{false}}$.

(2) **Non-effectiveness.** Another aspect that needs to be clarified is that GET tacticals return their first argument as result when the program execution of $T$ does not change the working memory. In our case we will make it equivalent to the situation of the program $T^{\text{DDL}}$ being not effective, that is the application of the program $T^{\text{DDL}}$ in a state $s_{\text{args}}$ will lead us to the same state $s$. This condition about the non-effectiveness of program $T^{\text{DDL}}$ can be expressed by the following relation:

$$T(\text{args}) = \text{first}(\text{args}) \iff [\text{args}] \vdash_{\text{DDL}} (T^{\text{DDL}})_{\varphi} \rightarrow \varphi, \forall \varphi \in \Phi_0.$$ 

Notice that the axiom MON together with the condition above allow us to see the relation as:

$$T(\text{args}) = \text{first}(\text{args}) \iff \forall s_{\text{args}}, \exists t: (s_{\text{args}}, t) \in \rho(T^{\text{DDL}})$$

being $t = s_{\text{args}}$.

(3) **Effectiveness.** When the computation of an inference rule over an appropriate set of arguments produces a formula that was not present in the arguments, tacticals return that formula as result. It means, in terms of DDL, that the program is effective, that is, there is a transition from a state satisfying the arguments to a state satisfying the inferred formula. This fact can be represented by the next relation:

$$T(\text{args}) = \text{result} \iff [\text{args}] \vdash_{\text{DDL}} (T^{\text{DDL}})_{\text{true}}$$

and

$$\wedge [T^{\text{DDL}}]_{\text{true}}.$$
arguments to the deduction of a particular formula in a suitable theory in DDL. The set of formulas, in a particular unit \( u \), associated to the tacticals in GET is defined as the right part of the next equivalences

\[
T(\text{args}) = \text{fail} \leftrightarrow u; [\text{args}] \rightarrow \bot_{\text{DDL}}
\]

\[
T(\text{args}) = \text{first}(\text{args}) \leftrightarrow u; [\text{args}]
\]

\[
\rightarrow ((T_{\text{DDL}}) \varphi \rightarrow \varphi)
\]

\[
T(\text{args}) = \text{result} \leftrightarrow u; [\text{args}]
\]

\[
\rightarrow [T_{\text{DDL}}][u; [\text{result}]].
\]

The programs associated to tactics and tacticals are the next ones:

1. Names of GET primitive tactics \( T \) representing natural deduction inference rules, \( \Delta \vdash \varphi \), are represented as sets of atomic programs representing all their possible instances,\(^7\) that is \( T_{\text{DDL}} = \bigcup \{ \{ \Gamma \vdash \varphi \} | \Gamma \text{ matches } \Delta, \varphi \text{ matches } \varphi \} \). So, for example, when \( T(\Gamma) = \varphi \) then clearly \( \Gamma_{\text{DDL}}[\{ \Gamma \vdash \varphi \}] \varphi \).

2. Tacticals are the next compound programs in DDL:
   - \( T = (T_1 \text{ THEN } T_2) \) is represented as:
     \[
     T_{\text{DDL}} = \bot_{\text{DDL}} \cup T_1_{\text{DDL}} \cup T_2_{\text{DDL}}
     \]
   - \( T = (T_1 \text{ OR ELSE } T_2) \) is represented as:
     \[
     T_{\text{DDL}} = T_1_{\text{DDL}} \cup \bot_{\text{DDL}} \cup T_2_{\text{DDL}}
     \]
   - \( T = (\text{TRY } T_1) \) is represented as:
     \[
     T_{\text{DDL}} = T_1_{\text{DDL}} \cup \bot_{\text{DDL}}
     \]
   - \( T = (\text{REPEAT } T_1) \) is represented as:
     \[
     T_{\text{DDL}} = (\bot_{\text{DDL}} \cup \bot_{\text{DDL}} \cup T_1_{\text{DDL}})^*.
     \]

To check whether the representation of tacticals in DDL corresponds to the intended meaning of failure and effectiveness of tacticals, we will analyse in detail three examples in the following proposition.

**Proposition 2.** Given \( T = (\text{TRY } T_1) \) and \( T' = (T_1 \text{ THEN } T_2) \) we have that for all \( s_{\text{args}} \) such that \( f(s_{\text{args}}) \supseteq \text{args} \):

1. If \( T(\text{args}) = \text{first}(\text{args}) \), then \( s_{\text{args}} \vdash \text{result} \). By the induction hypothesis \( T_1(\text{args}) = \text{fail} \). If and only if \( s_{\text{args}} \vdash \text{false} \).

2. If \( T(\text{args}) = \text{result} \), then \( \text{first}(\text{args}) \).

3. If \( T(\text{args}) = \text{fail} \), then \( s_{\text{args}} \vdash \text{false} \).

**Proof.** The proof is made as an induction step assuming that \( T_1 \) and \( T_2 \) have the desired behaviour. The test for the initial case, that is, for primitive tactics, is straightforward. The proof is a semantical one taking into account the model for GET tacticals. Given the completeness of DDL, there is also a syntactic proof.

1. By the semantics of tacticals in GET we have that \( (\text{TRY } T_1)(\text{args}) = \text{first}(\text{args}) \) iff \( T_1(\text{args}) = \text{false} \). By the induction hypothesis \( T_1(\text{args}) = \text{false} \) if and only if \( s_{\text{args}} \vdash \text{false} \).

2. By the semantics of tacticals in GET we have that \( (T_1 \text{ THEN } T_2)(\text{args}) = \text{result} \) iff \( T_1(\text{args}) = \text{result} \). By the induction hypothesis we have that \( T_1(\text{args}) = \text{result} \) entails \( s_{\text{args}} \vdash \text{result} \). So, by the properties of union of programs \( s_{\text{args}} \vdash \text{result} \).

3. By the semantics of GET we have that \( (T_1 \text{ THEN } T_2)(\text{args}) = \text{false} \) if one of the next two cases holds:
   a. \( T_1(\text{args}) = \text{false} \). In this case we have by induction hypothesis that if \( T_1(\text{args}) = \text{false} \) then \( s_{\text{args}} \vdash \text{false} \).
   b. \( T_1(\text{args}) = \text{false} \) and \( T_2(\text{args}) = \text{true} \). Then, obviously, \( s_{\text{args}} \vdash \text{false} \).

---

\(^7\) The finiteness of the languages ensures that the set of instances is finite. However, the inference rule of implication introduction needs to be modelled by special axioms. Its detailed explanation is omitted here.
7. Conclusion

In this work we have focused on using dynamic logics to describe a particular case of computational dynamic programs called multi-language knowledge-bases systems. These systems are built upon several languages, connected through bridge rules that in some cases implement Reification/Reflection mechanisms. Many artificial intelligence architectures embed such formalisms as the basic reasoning system as it is the case in MC [7-89], BMS [18], ML2 [5], DESIRE [19], MILOID II [1, 2, 16], OMEGA [4], FOL [21], and in the object-centered language for knowledge modelling called NOOS [3].

We have presented a family of dynamic logics called descriptive dynamic logic that provides a general framework to describe and compare such multi-language architectures, because DDL allows to neatly model their operational semantics. This is the first attempt to formalize executions in multi-language knowledge-based system and in particular reflective architectures as proofs in dynamic logic. Other attempts are based on temporal semantics [20] or on general logics [14]. As a next research step, we will show how the most relevant multi-language and multi-agent architectures can be described and compared in DDL.

Acknowledgements

We are gratefully indebted to Wiebe van der Hoek, John-Jules Meyer, Yao-Hua Tan, Fausto Giunchiglia, Paolo Traverso and Luciano Serafini for many useful comments made on a previous draft of this paper. This Research has been supported by the European TMR number PL93-0186 VIM, the Esprit III Basic Research Action number 6156 DRUMS II, and spanish projects DGCYT PS93-0212 and ANALOG TIC122-93.

References


Carles Sierra is Researcher at the Institute of Research on Artificial Intelligence of the Spanish Council for Scientific Research (CSIC). He received the M.S. in Computer Science (1986) and the Ph.D. in Computer Science (1989) from the Technical University of Catalonia, Barcelona, Spain. His present research interests include: Formal methods applied to Knowledge-Based systems analysis, computational reflection, multi-agent systems and applications of AI to medicine.

Lluís Godo was born 1957 in Barcelona (Spain). He studied mathematics at the University of Barcelona and he received his M.Sc. in 1979. He got his Ph.D. degree in Artificial Intelligence by the Polytechnical University of Catalonia in 1990 with a thesis about inference models in possibilistic systems. Currently he is Scientific Researcher of the Spanish Council for Scientific Research (CSIC) at the Artificial Intelligence Research Institute (IIIA). He has received the European Artificial Intelligence Research Paper Award (1987). His interests are logical foundations of uncertain, fuzzy and defeasible reasoning in general, and more recently reflective and multi-agent system architectures.

Maria Manzano received the Ph.D. in Logic from the University of Barcelona and is currently at the University of Salamanca. She was a Fulbright Scholar in the Group in Logic and the Methodology of Science at Berkeley and has visited the CTCS in Leeds and the CSLI of Stanford. She has published a number of papers and books (among them books published by CUP, OUP, Alianza Editorial and John Wiley). Her research interests include logics with applications in CS and AI such as: higher order logics, partial and/or heterogeneous logics, modal and dynamic logics.