Reasoning about Dynamic Features in Specification Languages – A Modal view on Creation and Modification –*

Rix Groenboom
Gerard R. Renardel de Lavalette

Department of Computing Science, University of Groningen
P.O. Box 800, 9700 AV Groningen, The Netherlands.
E-mail: {rix,grl}@cs.rug.nl.

Abstract

Using algebras over some signature to model the notion of state is quite common in specification languages. Some specification formalisms, e.g. COLD and Evolving Algebras, also allow for the dynamic change of states. Two kinds of elementary procedures are used: creation (of a new object) and modification (of a function or predicate at some point).

In this paper we present and investigate **MLCM** (Modal Logic of Creation and Modification), a multimodal predicate logic for reasoning over programs built up from such procedures. **MLCM** deviates from traditional dynamic predicate logic in two respects: creation is added as a primitive program construct and assignment (to variables) is replaced by assignment to constants and parametrized assignment to function and predicate symbols.

We present a definition of syntax, semantics and axiomatisation of **MLCM** and establish completeness for the repetition-free fragment via a traditional Henkin construction.

1 Introduction

The idea that algebraic methods admit a clear and precise description of abstract data types, already expressed and exemplified in [11], has been incorporated in many specification languages and related formalisms. Another advantage of algebraic specifications is also exploited in many places, viz. executability via rewriting.

*appeared in: *Semantics of Specification Languages* (D.J. Andrews, J.F. Groote, C.A. Middelburg, eds.), Springer-Verlag (1994) 340 – 355

In this paper, we consider an extension of this algebraic paradigm with a dynamical aspect: algebras (containing functions and predicates) are considered as states, and so-called *procedures* may change such algebras, leading to new ones (with the same signature). This extended paradigm can be found at the core of, e.g., the specification language COLD and the specification formalism of *Evolving Algebras*. We introduce and study the multimodal logic **MLCM** (Modal Logic of Creation and Modification), a variant of dynamic logic intended to capture the ideas behind COLD and Evolving Algebras.

New in this paper are the following:

- a nontrivial variant of dynamic logic for reasoning over programs involving object creation and function and predicate modification (**MLCM**);
- a semantics for this logic which is different from the semantics for dynamic logic;
- a partial completeness result.

1.1 Survey of the rest of this paper

In section 2, we give some background information about COLD, Evolving Algebras and dynamic logic. We then present **PL** in section 3, a one-sorted predicate logic, serving as the basis for **MLCM**. **PL** has a slightly non-standard semantics: quantification is defined over existing objects of the sort, and functions and predicates are evaluated on equivalence classes (completeness is shown in Appendix A). **PL** and its semantics prelude on the logic **MLCM**, the modal extension of **PL**, the subject of section 4. We give its semantics, axiomatisation and present a partial completeness result (for the fragment without the repetition construct, in Appendix B). We end with conclusions and suggestions for further research in section 5.

1.2 Acknowledgements

This research is the spin-off of the 'Dynamic logic and Relational calculus' group. The authors are indebted to Rutger Dijkstra, Dirk Roorda, Erik Saaman and Jan Terlouw, the other members of the group, for their frequent and constructive suggestions and criticism.

2 Background

In this section we shortly discuss some formalisms that inspired us to the development of **MLCM**. First we sketch COLD and Evolving Algebra, two specification formalisms where the need for a reasoning system about programs involving certain modification constructs was felt. Then we give a survey of dynamic logic which allows reasoning over programs with assignments and show the need for a variant of dynamic logic.

2.1 COLD

The wide-spectrum specification language COLD (Common Object-oriented Language for Design) is developed at Philips Research Eindhoven in several ESPRIT-projects, during the last ten years. The main ideas for the language originate from Hans Jonkers. The formal definition of COLD-K and its semantics have has been given in 1987 in [6]. The semantics is based on the many sorted partial infinitary logic $\mathbf{MPL}_\omega$, see [16]. The textbook [5] gives a good introduction into COLD-K, the kernel language of COLD.

For state-based specifications COLD has so-called *classes*. A class is a sort of abstract machine with states and nondeterministic procedures. A state is associated with an algebra, containing sorts, predicates and functions. The procedures that operate on these states may modify the state by extending the sorts and changing the functions and predicates of the state.

To illustrate the operations of creation and modification, we give the signature of a state-based specification of memory cells:

```
CLASS cell-class

SORT  Nat
      Cell

FUNC  0          : -> Nat
      s          : Nat -> Nat
      val        : Cell -> Nat

PROC  create     : -> Cell
      store      : Cell x Nat ->
```

The intuition behind this is: a memory cell can contain a natural number, its value. For natural numbers we have a constant function `0` and the successor function `s`. The procedures that change the state of the class are: `create` and `store`. With the procedure `create` we make a new memory cell, with `store` we update the value of a cell.

To specifiy the behaviour of `create` in COLD we want to write (among other things) that it returns a cell when applied. This is done as follows:

```
[ LET c:Cell; c:=create] (EXIST x:Cell (x=c))
```

We shall see that **MLCM** gives the possibility to reason about these kind of procedures, in such a way that we may infer:

$$\vdash \quad [\mathrm{NEW}c]\exists x : \mathtt{Cell}(x = c)$$

$$\vdash \quad \forall x : \mathtt{Cell}, y : \mathtt{Nat}\big([\mathtt{store}(x,y)](\mathtt{val}(x) = y)\big)$$

A general COLD specification contains besides a signature definition also axioms that the functions and procedures must satisfy. It is also possible to give implementations of the specified procedures in terms of dynamic logic. So the behaviour of `create` may be specified by:

```
AXIOM < create > TRUE
AXIOM [ LET c:Cell; c:= create]
            ( c! AND (PREV NOT c!) AND
            FORALL d:Cell ( (PREV NOT d!) => d=c ) )
```

The first axiom ensures termination, the second expresses that `c` was not defined in the previous state but has come into existence in the current state, and that it is the only new object.

2.2 Evolving algebras

Evolving algebras have been introduced by Gurevich in [8] and [9], and applied for the specification of several programming languages like Occam [10] and Prolog [1, 2].

The basic idea of evolving algebras is that computational processes change dynamically over time. In evolving algebras, states are represented as many-sorted first-order structures S with universes and functions. States can change under the influence of application of *transaction rules*. Adopting the notation as given in [2], these are rules of the form:

```
if B then U1 and U2 and ... and Un.
```

Here each `Ui` is an *update*:

```
F(E1, ..., Ej) := E0
```

`B` is a boolean expression, `F` is a function, and each `Ei` is an expression in the signature of the structure S.

To modify universes (i.e. to let them grow by creating new objects), there are updates of the form:

```
extend A by E1,..., Em with U1 and U2 and ... and Un end.
```

The updates `Uj` depend on `Ei`'s, setting the values of some functions on newly created elements `Ei` of `A`. Typically the universes are initially empty.

Normally the updates are executed in parallel, but to execute updates sequentially, one may write:

```
seq U1 and U2 and ... and Um endseq
```

We mention in passing that this last construct can also be expressed in COLD, as well as the `let` and `if ... then ... else ...` constructs mentioned in [2].

To come back on the memory cell example, a way for specifing that a new memeory cell will initially be 0 we can give the following rule:

```
let rule = (extend Cell by c with val(c) := 0)
```

After execution of `rule` one would like to prove that

$$\vdash \quad [\texttt{rule}]\exists x : \texttt{Cell}(x = c \wedge \texttt{val}(x) = 0)$$

and this is the type of reasoning that we want to cover in **MLCM**.

2.3 Dynamic logic

Dynamic logic originates from V. Pratt (see [17]), after a suggestion of his student R. Moore in 1974. D. Harel has made many contributions to the subject, especially the version with first-order quantification (see [12]). The original idea was to express Hoare triples in a modal setting: the Hoare triple $A\{\alpha\}B$ (if A holds before executing statement α then B holds afterwards) is expressed in dynamic logic as $A \rightarrow [\alpha]B$. In terms of Dijkstra's wp-calculus ([4]), $[\alpha]A$ corresponds with $wlp.\alpha.A$, the weakest liberal precondition needed to guarantee that A holds after every possible execution of α (here *liberal* refers to the fact that termination of α is not required).

Dynamic logic is a formalism for expressing and reasoning with assertions about programs. In its usual first-order setting, programs are built up from assignments to variables ($x := t$) and guards ($A?$), using the program constructs ; (sequential composition), $\cup$ (nondeterministic choice) and * (repetition). As a consequence, variables play the double role of both logical and programming variable in dynamic logic. These roles will be separated in **MLCM**.

A second, and more important shortcoming (at least for our purposes sketched above) of dynamic logic is that is not possible to make assertions about function modification, like:

$$\forall x : \texttt{Cell}, y : \texttt{Nat}([\texttt{store}(x,y)](\texttt{val}(x) = y)$$

The axiomatisation of the behaviour of programs makes it possible to use e.g. tactical theorem provers to support proving of properties about programs. The system KIV (Karlsruhe Interactive Verifier, see [14] for a general description of the system) is an interactive system designed for formal reasoning about imperative Pascal-like programs. The KIV system uses a variant of dynamic logic, as presented by Goldblatt ([7]) in combination with tactical theorem proving. A description of this KIV logic, which uses an approximation of the infinitary rules for proving repetition by induction, can be found in [13].

3 PL: Predicate Logic

As mentioned earlier, we present the logic **MLCM** in two stages. Here we introduce the predicate logic **PL**, with a somewhat unusual semantics, prepared for extending it to the semantics of the full language **MLCM**. In fact, **PL** is the *local* logic of **MLCM**, i.e. the logic of (the algebra of) a single state. The uncommon feature of the semantics of **PL** is that functions and predicates are defined on equivalence classes instead of individual objects; moreover, functions are partial. The extension to **MLCM** follows in the next section.

3.1 The language and notation

The language of **PL** is defined by:

VAR	x	(a countably infinite collection of variables)
FUNC	f	(function symbols, with arity ≥ 0)
PRED	p	(predicate symbols, with arity ≥ 0)
TERM	t	$::= \uparrow \mid x \mid f(t_1, \ldots, t_n)$
FORM	A	$::= (t = t) \mid p(t_1, \ldots, t_n) \mid A \wedge A \mid \neg A \mid \forall x A$

The formulae $\top, \bot, A \vee B, A \to B, A \leftrightarrow B$ and $\exists x A$ are defined as usual. The symbol $\uparrow$ denotes the undefined object. We also define a unary *definedness* predicate $\cdot\downarrow$ and *weak equality* $\cdot \simeq \cdot$ (also called equivalence or partial equality):

$$\begin{array}{lll} t\downarrow & =_{\text{def}} & (t = t) \\ s \simeq t & =_{\text{def}} & (s\downarrow \vee t\downarrow) \to s = t \end{array}$$

The notions of free and bound variables of a formula are defined as usual. The same holds for *substitution*, denoted as $[t/x]s$ and $[t/x]A$ which can be read as 'substitute t for variable x in s resp. A'. A formula without free variables is called a *sentence*.

3.2 Axiomatisation of PL

PL is axiomatized as follows (for the sake of simplicity, we assume here and later that functions and predicates are unary):

Taut	All tautologies of propositional logic
MP	$A, A \to B \Rightarrow B$
Eq	$x \simeq y \to y \simeq x$ $x \simeq y \wedge y \simeq z \to x \simeq z$ $x \simeq y \to fx \simeq fy$ $x \simeq y \to (px \leftrightarrow py)$
Undef	$\neg(\uparrow\downarrow)$
Quan	$(x\downarrow \to A) \Rightarrow \forall x A$ $(\forall x A \wedge x\downarrow) \to A$
Subst	$A \Rightarrow [t/x]A$

In this axiomatisation, the expressions containing $\Rightarrow$ are rules, the others are axioms.

$\Gamma \vdash A$ (A is derivable from Γ), is defined inductively as usual: all $A \in \Gamma$ and all instances of axioms are derivable from Γ, and if all premises of a rule are derivable from Γ then so is the conclusion. A collection of formulae Γ is called *consistent* iff $\Gamma \nvdash \bot$.

Let x be not free in Γ. Observe that we do *not* have $\Gamma \vdash A \Leftrightarrow \Gamma \vdash \forall x A$ for x not free in Γ, i.e. a counterexample is $\Gamma = \emptyset, A = x{\downarrow}$. But we do have, again for x not free in Γ:

$$\Gamma \vdash A(x) \quad \Leftrightarrow \quad \Gamma \vdash (\forall x A(x)) \wedge A(\uparrow) \tag{1}$$

$$\Gamma \vdash \forall x A(x) \quad \Leftrightarrow \quad \Gamma \vdash x{\downarrow} \rightarrow A(x) \tag{2}$$

3.3 Semantics of PL

The semantics for **PL** is slightly nonstandard and has the following characteristics. The universe U is divided in equivalence classes, and each function and predicate is defined on equivalence classes. One special object $\star$ plays the role of the undefined object.

More formally: a model M for **PL** is a quadruple $M =< U, \sim, F, P >$, where

- U is the *pre-universe*, elements $u \in U$ are called pre-objects, and $\star \in U$ is the *undefined object*;
- $\sim$ is an equivalence relation on U;
- $F = \{\underline{f} | f \in \text{FUNC}\}$ with $\underline{f} : U_\sim^n \rightarrow U_\sim$ (n the arity of f);
- $P = \{\underline{p} | p \in \text{PRED}\}$ with $\underline{p} \subseteq U_\sim^n$ (n the arity of p).

Here (and later) we use the following definitions:

$\tilde{u}$	$=_{\text{def}}$	$\{u' \mid u' \sim u\}$	(the objects of the universe)
$U_\sim$	$=_{\text{def}}$	$\{\tilde{u} \mid u \in U\}$	(the universe of M)
$U_\sim^+$	$=_{\text{def}}$	$U_\sim \setminus \{\tilde{\star}\}$	(the universe with only existing objects)

Assignments $a \in \text{ASS} = \text{VAR} \rightarrow U$ and pointwise modification $a[x \mapsto u]$ (where $x \in \text{VAR}, u \in U$) of an assignment are defined as usual.

Now we can define the interpretation $[\![t]\!]_{M,a}$ of a term t and $M, a \models A$ of a formula A in model M under assignment a. The interpretation function has the following type $[\![\ .\ ]\!] : \text{TERM} \rightarrow U_\sim$. We use $\tilde{a}(x)$ to denote $\{u \mid u \sim a(x)\}$. We recursively define:

$[\![\uparrow]\!]_{M,a}$	$=$	$\tilde{\star}$
$[\![x]\!]_{M,a}$	$=$	$\tilde{a}(x)$
$[\![ft]\!]_{M,a}$	$=$	$\underline{f}([\![t]\!]_{M,a})$
$M, a \models pt$	$=_{\text{def}}$	$[\![t]\!]_{M,a} \in \underline{p}$
$M, a \models (s = t)$	$=_{\text{def}}$	$[\![s]\!]_{M,a} = [\![t]\!]_{M,a} \neq \tilde{\star}$
$M, a \models \neg A$	$=_{\text{def}}$	**not** $(M, a \models A)$
$M, a \models A \wedge B$	$=_{\text{def}}$	$M, a \models A$ **and** $M, a \models B$
$M, a \models \forall x A$	$=_{\text{def}}$	**forall** $u \in (U \setminus \tilde{\star})$ $(M, a[x \mapsto u] \models A)$
$M \models A$	$=_{\text{def}}$	**forall** $a \in \text{ASS}(M, a \models A)$
$\Gamma \models A$	$=_{\text{def}}$	**forall** M(**forall** $B \in \Gamma(M \models B) \Rightarrow M \models A)$
$\models A$	$=_{\text{def}}$	**forall** $M(M \models A)$

In this definition, Γ stands for a set of sentences. It is obvious that, for *sentences* A we have that $M, a \models A$ and $M \models A$ are equivalent. Observe the difference between $M \models A(x)$ and $M \models \forall x A(x)$:

- $M \models A(x)$ means: all objects of M satisfy A;
- $M \models \forall x A(x)$ means: all *existing* objects of M satisfy A.

However, we do have the following equivalences, cf. (1) and (2):

$$M \models A(x) \quad \Leftrightarrow \quad M \models (\forall x A(x)) \wedge A(\uparrow) \tag{3}$$

$$M \models \forall x A(x) \quad \Leftrightarrow \quad M \models x{\downarrow} \rightarrow A(x) \tag{4}$$

Equivalences (1), (2), (3) and (4) enable us to reduce formulae to sentences with the same derivability and semantical properties.

3.4 Soundness and completeness

Soundness ($\Gamma \vdash A \Rightarrow \Gamma \models A$) is proved straightforwardly with induction over the definition of derivability. For the substitution rule, the following property is needed:

$$\begin{array}{lcl} [\![[t/x]s]\!]_{M,a} & = & [\![s]\!]_{M,a[x \mapsto [\![t]\!]_{M,a}]} \\ M, a \models [t/x]A & \Leftrightarrow & M, a[x \mapsto [\![t]\!]_{M,a}] \models A \end{array}$$

This is proved with induction over s and A, respectively. Completeness ($\Gamma \models A \Rightarrow \Gamma \vdash A$) is obtained by adapting the Henkin construction. For a sketch of this proof see Appendix A.

4 MLCM: Modal Logic of Creation and Modification

The logic **MLCM** is an extension of **PL**, obtained by adding formulae $[\alpha]A$, where α is a program expression. For the composition of complex program expressions the same operations are used as in dynamic logic (see 2.3).

New in **MLCM** are the three atomic program statements:

- NEWc (creating a new object and letting the constant c refer to it);
- $f(t_1, \ldots, t_n) := t$ (changing the value of function f on the arguments $t_1, \ldots, t_n$ to the value of t);
- $p(t_1, \ldots, t_n) :\leftrightarrow A$ (changing the value of predicate p on the arguments $t_1, \ldots, t_n$ to the truth value of A).

The second of these (function modification) is related to the assignment statement $x := t$ in dynamic logic, but there are two differences.

First, the simple assignment can be modeled in **MLCM** by changing the value of a *constant* (i.e. a function with zero arguments), not a variable. In other words, program variables are treated as constant names, not as logical variables.

This leads to a clear separation between *logical* variables and *programming* variables.

Secondly, **MLCM** allows for parametrized assignments by changing functions with positive arity.

The syntax of **MLCM** reads:

VAR	x	(a countably infinite collection of variables)
FUNC	f	(function symbols, with arity ≥ 0)
PRED	p	(predicate symbols, with arity ≥ 0)
TERM	$t ::=$	$\uparrow \mid x \mid f(t_1, \ldots, t_n)$
PROG	$\alpha ::=$	$\mathrm{NEW}c \mid f(t_1, \ldots, t_n) := t \mid p(t_1, \ldots, t_n) :\leftrightarrow A \mid$ $A? \mid \alpha;\alpha \mid \alpha \cup \alpha \mid \alpha^*$
FORM	$A ::=$	$(t = t) \mid p(t_1, \ldots, t_n) \mid A \wedge A \mid \neg A \mid \forall x A \mid [\alpha]A$

Here, c stands for constant symbols, i.e. function symbols with arity 0. We have the standard abbreviations for $\top, \bot, A \vee B, A \rightarrow B, A \leftrightarrow B, \exists x A$ and $<\alpha> A$.

4.1 Semantics of MLCM

The definition of the semantics of **MLCM** has two steps. First we define the notion of *structure*, a kind of proto-model in which terms and formulae of **MLCM** can be interpreted. Then we restrict this notion to *models* by imposing requirements on the accessibility relations corresponding to the program statements: the interpretation of formulae is used in the formulation of these requirements.

A *structure* is a triple $M =< U, W, R >$, where

- U is the *global* pre-universe with $\star \in U$;
- $W \neq \emptyset$ is the collection of *worlds*: they are triples $w =<\sim_w, F_w, P_w >$ such that $< U, \sim_w, F_w, P_w >$ is a model of **PL**. Thus $F_w = \{f_w | f \epsilon \mathrm{FUNC}\}$ and $P_w = \{p_w | p \epsilon \mathrm{PRED}\}$;
- $R = \{R_{\alpha,a} | \alpha \in \mathrm{PROG}, a \in \mathrm{ASS}\}$ is a collection of binary relations on W.

We define:

$$\begin{array}{lll} u_w & =_{\mathrm{def}} & \{u' \mid u' \sim_w u\} \\ U_w & =_{\mathrm{def}} & \{u_w \mid u \in U\} \\ U_w^+ & =_{\mathrm{def}} & U_w \setminus \{\star_w\} \end{array}$$

Let $[\![t]\!]_{w,a}$, the interpretation of term t in world w with assignment a, be defined by:

$$\begin{array}{lll} [\![\uparrow]\!]_{w,a} & = & \star_w \\ [\![x]\!]_{w,a} & = & (a(x))_w \\ [\![ft]\!]_{w,a} & = & f_w([\![t]\!]_{w,a}) \end{array}$$

We let f_w and p_w range of F_w resp. P_w.
For $w, a \models A$, the interpretation of formula A in world w with assignment a, we have

$$
\begin{array}{lll}
w, a \models pt & =_{\text{def}} & [\![t]\!]_{w,a} \in p_w \\
w, a \models (s = t) & =_{\text{def}} & [\![s]\!]_{w,a} = [\![t]\!]_{w,a} \neq \star_w \\
w, a \models \neg A & =_{\text{def}} & \textbf{not } (w, a \models A) \\
w, a \models A \wedge B & =_{\text{def}} & w, a \models A \textbf{ and } w, a \models B \\
w, a \models \forall x A & =_{\text{def}} & \textbf{forall } u \in (U \setminus \star_w)(w, a[x \mapsto u] \models A) \\
w, a \models [\alpha]A & =_{\text{def}} & \textbf{forall } w' \in W(wR_{\alpha,a}w' \Rightarrow w', a \models A) \\
w \models A & =_{\text{def}} & \textbf{forall } a \in \text{ASS}(w, a \models A) \\
\Gamma \models A & =_{\text{def}} & \textbf{forall } w \in W(\textbf{forall } B \in \Gamma(w \models B) \Rightarrow w \models A) \\
\models A & =_{\text{def}} & \textbf{forall } w \in W(w \models A)
\end{array}
$$

A structure is called a *model* if it satisfies a number of requirements for the relations $R_{a,a} \epsilon R$. We sketch the ideas before giving the formal definitions.

If $wR_{NEWc,a}w'$, then there is exactly one new existing object (i.e. a new equivalence class) in $U_{w'}$, to which c refers. When going from w to w', the behaviour of functions f is as follows: $f_{w'}$ behaves like f_w on 'old' objects, and for the unique new object $c_{w'}$ we have $f_{w'}c_{w'} = f_{w'}\star_{w'}$. For predicates, the situation is analogous: their extension is equal with respect to the 'old' universe, and $p_{w'}c_{w'}$ iff $p_{w'}\star_{w'}$.

If $wR_{f(s):=t,a}w'$, then the universe is unchanged, and so are all functions and predicates except f. Compared with f_w, $f_{w'}$ is changed in one point, viz. the value of s in w, which becomes the value of t in w. Observe that both s and t may contain f, so their value in w' may be different from that in w.

Similar observations can be made when $wR_{p(s):\leftrightarrow A,a}w'$ holds.

Now for the precise formulation:

- if $wR_{NEWc,a}w'$ then:
  - $U_w^+ = U_{w'}^+ \setminus \{c_{w'}\}$
  - $c_{w'} \in U_{w'}^+$
  - $f_{w'} = i \circ f_w \circ j$ **forall** $f \in \text{FUNC} \setminus \{c\}$
  - $p_{w'} = j^{-1}(p_w)$ **forall** $p \in \text{PRED}$

  Here the functions $i : U_w \to U_{w'}$ and $j : U_{w'} \to U_w$ are defined by

$$
\begin{array}{lll}
i(u) & = & u \quad \textbf{for } u \in U_w^+ \\
i(\star_w) & = & \star_{w'} \\
j(u) & = & u \quad \textbf{for } u \in U_w^+ \\
j(c_{w'}) & = & \star_w \\
j(\star_{w'}) & = & \star_w
\end{array}
$$

  and $j^{-1}(p_w)$ means the image in $U_{w'}$ of the elements U_w for which p_w holds.

- if $wR_{f(s):=t,a}w'$ then:
  - $U_{w'} = U_w$
  - $f_{w'}([\![s]\!]_{w,a}) = [\![t]\!]_{w,a}$
  - $f_{w'}v = f_w v$ **forall** $v \in U_w \setminus \{[\![s]\!]_{w,a}\}$
  - $g_w = g_{w'}$ **forall** $g \in \mathrm{FUNC} \setminus \{f\}$
  - $p_w = p_{w'}$ **forall** $p \in \mathrm{PRED}$
- if $wR_{p(s):\leftrightarrow A,a}w'$ then:
  - $U_{w'} = U_w$
  - $([\![s]\!]_{w,a}) \in p_{w'}$ **iff** $w, a \models A$
  - $v \in p_{w'} \Leftrightarrow v \in p_w$ **forall** $v \in U_w \setminus \{[\![s]\!]_{w,a}\}$
  - $f_w = f_{w'}$ **forall** $f \in \mathrm{FUNC}$
  - $q_w = q_{w'}$ **forall** $q \in \mathrm{PRED} \setminus \{p\}$
- $R_{A?,a} = \{(w,w) | w, a \models A\}$
- $R_{\alpha;\beta,a} = R_{\alpha,a} \circ R_{\beta,a}$
- $R_{\alpha\cup\beta,a} = R_{\alpha,a} \cup R_{\beta,a}$
- $R_{\alpha^*,a} = R^*_{\alpha,a}$

4.2 Substitution

Substitution $[\mathrm{t/x}]A$ of term t for x in A is not always defined in **MLCM**, only if no function symbols in t come into the scope of program statements that may change their meaning. So, e.g., $[c/x]([\mathrm{NEW}c](x = y))$ nor $[fy/x]([f(s) := t](x = z))$ is defined. The clauses for the definition of substitution are as usual, except for formulae of the form $[\alpha]A$:

$$[\mathrm{t/x}]([\alpha]A) = [[\mathrm{t/x}]\alpha]([\mathrm{t/x}]A)$$
$$\text{if } \mathit{func}(t) \cup \mathit{mod}(\alpha) = \emptyset \text{ or } x \text{ does not occur freely in } A$$

Here $\mathit{func}(t)$ is the set of function symbols occurring in t and $\mathit{mod}(\alpha)$, the collection of signature elements that are possibly modified by α, is defined by

$$\begin{array}{ll}
\mathit{mod}(\mathrm{NEW}c) & = \{c\} \\
\mathit{mod}(f(s) := t) & = \{f\} \\
\mathit{mod}(p(s) :\leftrightarrow A) & = \{p\} \\
\mathit{mod}(A?) & = \emptyset \\
\mathit{mod}(\alpha;\beta) & = \mathit{mod}(\alpha) \cup \mathit{mod}(\beta) \\
\mathit{mod}(\alpha \cup \beta) & = \mathit{mod}(\alpha) \cup \mathit{mod}(\beta) \\
\mathit{mod}(\alpha^*) & = \mathit{mod}(\alpha)
\end{array}$$

4.3 Axiomatisation

MLCM is axiomatised as follows ($[\alpha]\Gamma$ stands for $\{[\alpha]A \mid A\epsilon\Gamma\}$):

PL	All axioms and rules of **PL** (with only those instances of the substitution rule for which the substitution is defined)
N	$\Gamma \vdash A \Rightarrow [\alpha]\Gamma \vdash [\alpha]A$
C1	$x = y \rightarrow [\mathrm{NEW}c](x = y \neq c)$
C2	$x \neq y \rightarrow [\mathrm{NEW}c](x \neq y \vee x = c)$
C3	$px \rightarrow [\mathrm{NEW}c]px$
C4	$\neg px \rightarrow [\mathrm{NEW}c]\neg px$
C5	$[\mathrm{NEW}c](c\downarrow \wedge fx \neq c \ \wedge \ fc \simeq f\uparrow \ \wedge (pc \leftrightarrow p\uparrow))$
FM1	$A \leftrightarrow [f(s) := t]A$ for all A not containing f
FM2	$s \simeq x \wedge t \simeq y \rightarrow [f(s) := t]fx \simeq y$
FM3	$s \not\simeq x \wedge fx \simeq y \rightarrow [f(s) := t]fx \simeq y$
PM1	$A \leftrightarrow [p(s) :\leftrightarrow C]A$ for all A not containing p
PM2	$s \simeq x \rightarrow (C \leftrightarrow [p(s) :\leftrightarrow C]px)$
PM3	$s \not\simeq x \rightarrow (px \leftrightarrow [p(s) :\leftrightarrow C]px)$
?AX	$[A?]B \leftrightarrow (A \rightarrow B)$
;AX	$[\alpha;\beta]A \leftrightarrow [\alpha][\beta]A$
$\cup$**AX**	$[\alpha \cup \beta]A \leftrightarrow ([\alpha]A \wedge [\beta]A)$
$*$**AX**	$[\alpha^*]A \leftrightarrow (A \wedge [\alpha][\alpha^*]A)$
INF	$\{A \rightarrow [\alpha^n]B \mid n \in \mathrm{N}\} \Rightarrow A \rightarrow [\alpha^*]B$

Here, $[\alpha^n]$ is recursively defined as: $[\alpha^0] = [\top?]$ and $[\alpha^{n+1}] = [\alpha][\alpha^n]$.

4.4 Some consequences

Given this axiomatisation we can make the following observations.
The axiom rule [**N**] is used to derive:

Nec $A \Rightarrow [\alpha]A$ (Necessitation)

Distr $[\alpha](A \rightarrow B) \rightarrow ([\alpha]A \rightarrow [\alpha]B)$ (Distribution)

From the infinitary axiom INF, we have the following induction principle:

1 $[\alpha^*](A \rightarrow [\alpha]A) \rightarrow (A \rightarrow [\alpha^*]A)$

Furthermore, we have:

2 $x{\downarrow} \rightarrow [\mathrm{NEW}c](x{\downarrow} \wedge x \neq c)$

3 $\neg x{\downarrow} \rightarrow [\mathrm{NEW}c](\neg x{\downarrow} \vee x = c)$

4 $x \simeq y \rightarrow [\mathrm{NEW}c](x \simeq y \vee (\neg x{\downarrow} \wedge y = c) \vee (\neg y{\downarrow} \wedge x = c))$

5 $x \not\simeq y \rightarrow [\mathrm{NEW}c](x \not\simeq y \wedge (x = c \rightarrow y{\downarrow}) \wedge (y = c \rightarrow x{\downarrow}))$

6 $fx \simeq y \rightarrow [\mathrm{NEW}c](fx \simeq y \vee (fx \simeq{\uparrow} \wedge y = c))$

7 $fx \not\simeq y \rightarrow [\mathrm{NEW}c](fx \not\simeq y \wedge (y = c \rightarrow fx{\downarrow}))$

4.5 Soundness and completeness

Soundness is proved straightforwardly by checking that the requirements on the $R_{\alpha,a}$ are sufficient to make the axioms valid in all models. In Appendix B we sketch a partial completeness proof for the fragment of **MLCM** without the repetition construct *.

5 Concluding remarks

We presented a multimodal logic **MLCM** intended to formalise reasoning over evolving algebras and (partially) over procedures in COLD. The work is in progress and e.g. a full completeness has not been established yet, and is subject of further investigation (the problems lie in the combination of quantification and the repetition construct). Not all dynamic language constructs COLD are covered by **MLCM**, e.g., the operator `PREV` which refers to the previous state. It seems more than plausible that this can be modeled by keeping track of the history, i.e. the sequence of states on a computation path. The extension of $\mathbf{MPL}_\omega$, the logic underlying COLD, to a modal framework is a (presumably straightforward) exercise.

The generality of arbitrary equivalence relations in the definition of universes from a pre-universe in the semantics of **MLCM** is not fully needed for the purpose of this theory: here it would suffice to work with universes where all equivalence classes are singletons, except the class containing $\star$, the undefined object. We did so with an eye on further generalisation.

For Evolving Algebras, a parallel function modification has to be introduced in the language. Then any specific evolving algebra description can easily be translated into an **MLCM** program. We hope to come back to these issues in subsequent publications.

Axiomatisation of the logic of COLD and Evolving Algebra clears the way for automated construction of proofs for these formalisms: the development of **MLCM** is only the first step towards such an axiomatisation. However, it is our intention to investigate the possibilities of theorem proving for **MLCM** and its variants. The work in the KIV project (see [14]) looks interesting in this perspective.

References

[1] E. Börger, A logical operational semantics for full prolog, In: *Proceedings CSL '89*, E. Börger, H. Kleine Büning and M.M. Richter (Eds.), LNCS 440, Springer Verlag, 1990, pp. 36 – 64.

[2] E. Börger and D. Rosenzweig, *The WAM – definition and Compiler Correctness*, Technical Report TR - 14/92, Department of Computer Science, University of Pisa, 1992.

[3] C.C. Chang and H.J. Keisler, *Model Theory*, North Holland, Second revised editon, 1990.

[4] E.W. Dijkstra and C.S. Scholten, *Predicate Calculus and Program Semantics*, Springer Verlag, 1990.

[5] L.M.G. Feijs and H.B.M. Jonkers, *Formal Specification and Design*, Cambridge Tracts in Theoretical Computer Science 35, 1992.

[6] L.M.G. Feijs, H.B.M. Jonkers, C.P.J. Koymans and G.R. Renardel de Lavalette, *Formal definition of the design language COLD-K (Preliminary version)*, ESPRIT document METEOR/t7/PRLE/7, April 1987 (Final version: August 1989).

[7] R. Goldblatt, *Axiomatising the logic of computer science*, LNCS 130, Springer Verlag, 1982.

[8] Y. Gurevich, Logic and the challenge of computer science, In: *Trends in Theoretical computer science*, E. Börger (ed.), Computer Science Press, 1988, pp. 1 – 57.

[9] Y. Gurevich, Evolving algebras; a tutorial introduction, Bulletin of the EATCS 43, Febr. 1991, pp. 264 – 284,

[10] Y. Gurevich and L. Moss, Algebraic operational semantics and Occam, In: *Peoceedings CSL '89*, E. Börger, H. Kleine Büning and M.M. Richter (Eds.), LNCS 440, Springer Verlag, 1990, pp. 176 – 192.

[11] J.V. Guttag and J.J. Horning, *The algebraic specification of abstract data types*, Acta Informatica 10, 1978, pp. 27 – 52.

[12] D. Harel, Dynamic Logic, In: *Handbook of Philosophical Logic, Vol. II*, D. Gabbay and F. Guenthner (eds.), D. Reidel Publishing Company, 1984, pp. 497 – 604.

[13] M. Heisel, W. Reif and W. Stephan, A dynamic logic for program verfication, In: *Logic at Botik 89*, A. Meyer, M. Taitslin (eds), LNCS 363, Springer Verlag, pp. 134 – 145.

[14] M. Heisel, W. Reif and W. Stephan, Tactical Theorem Proving in Program Verification, In: *Conference on Automated Deduction*, Siekmann (ed), LNAI, Spinger Verlag, 1990.

[15] L. Henkin, *The completeness of the first order functional calculus*, The Journal of Symbolic Logic 14, 1949, pp. 159 –166.

[16] C.P.J. Koymans and G.R. Renardel de Lavalette, The logic MPL_ω, In: *Algebraic Methods: Theory, tools and applications*, M. Wirsing and J.A. Bergstra (eds.), LNCS 394, Springer Verlag, 1989, pp. 247 – 282.

[17] V.R. Pratt, *Semantical considerations on Floyd-Hoare logic*, Proc. 17th annual IEEE symp. on foundations of computer science, 1976, pp. 109 – 121.

Appendix A. Completeness of PL

In order to prove the completeness ($\Gamma \models A \Rightarrow \Gamma \vdash A$) of **PL**, we first use (1), (2), (3) and (4) of sections 3.2 and 3.3 to reduce formulae to sentences. Then we show that $\Gamma \models A \Rightarrow \Gamma \vdash A$ for (collections of) sentences Γ, A by constructing for every *consistent* collection of sentences Γ in $\mathcal{L}$ (the language of **PL**) a model M with $M \models \Gamma$. This is enough to obtain completeness: if $\Gamma \models A$ then $\Gamma \cup \{\neg A\}$ holds in no model and hence is inconsistent, so $\Gamma \vdash A$.

To obtain M, we slightly adapt the wellknown Henkin construction for ordinary predicate logic (see [15], a good exposition is in [3]). The general idea is to extend a consistent set of sentences Γ to Γ'' from which a model $H = H(\Gamma'')$ can be defined that satisfies:

$$H \models A \Leftrightarrow A \in \Gamma'' \tag{5}$$

Starting with a consistent set of sentences Γ, H is obtained in four steps:

- Let $\mathcal{C}$ be a countably infinite collection of fresh constants (i.e. not occurring in $\mathcal{L}$), the so-called *Henkin constants*. We extend $\mathcal{L}$, the language of **PL**, to $\mathcal{L}^+$ by adding all substitution instances of elements of $\mathcal{L}$ with Henkin constants. $\mathcal{C}^\uparrow = \mathcal{C} \cup \{\uparrow\}$ will become the pre-universe of our model.

  So $\mathrm{ASS} = \mathrm{VAR} \to \mathcal{C}^\uparrow$, and $a \in \mathrm{ASS}$ will be used as mappings transforming $\mathcal{L}^+$-formulae in $\mathcal{L}^+$-sentences (by substitution).

- Extend Γ to $\Gamma' \subseteq \mathcal{L}^+$, satisfying

  - Γ' is consistent;
  - Γ' contains *witnesses* for $\mathcal{L}^+$ in $\mathcal{C}$, i.e. for every $A \in \mathcal{L}^+$ with at most one free variable x, there is a $c \in \mathcal{C}$ with $\Gamma' \vdash (\exists x A) \to (c{\downarrow} \wedge [c/x]A)$.

  Observe that this last property is preserved under extension of Γ'.

- Extend Γ' to a maximally consistent $\Gamma'' \subseteq \mathcal{L}^+$, i.e. satisfying

– if $\Delta \supseteq \Gamma''$ and Δ consistent, then $\Delta = \Gamma''$.

Now Γ'' satisfies:

$$\begin{array}{rcl} \neg A \in \Gamma'' & \Leftrightarrow & A \notin \Gamma'' \\ A \wedge B \in \Gamma'' & \Leftrightarrow & A \in \Gamma'' \textbf{ and } B \in \Gamma'' \\ \exists x A \in \Gamma'' & \Leftrightarrow & \textbf{for some } c \in \mathcal{C} \text{ we have } (c\downarrow \wedge [c/x]A) \in \Gamma'' \end{array}$$

These three properties will be the induction steps in the proof of (5).

- Now, for any maximally consistent set with witnesses Γ, the Henkin model $H = H(\Gamma) =< \mathcal{C}^{\uparrow}, \sim, F, P >$ is defined as follows:

  $\sim = \{(c,d) \in \mathcal{C}^{\uparrow} \times \mathcal{C}^{\uparrow} | (c \simeq d) \in \Gamma\}$
  for all $f \in \text{FUNC}, v \in \mathcal{C}^{\uparrow}/\sim \quad \underline{f}v = \{d \in \mathcal{C}^{\uparrow} | (fc \simeq d) \in \Gamma, c \in v\}$
  for all $p \in \text{PRED} \;\; \underline{p} = \{d \in \mathcal{C}^{\uparrow} | pd \in \Gamma\}/\sim$

With this definition of H, we obtain (5) for atomic A:

$$\begin{array}{rcl} a(s = t) \in \Gamma'' & \Leftrightarrow & H, a \models s = t \\ a(pt) \in \Gamma'' & \Leftrightarrow & H, a \models pt \end{array}$$

which finishes the proof for (5).

Appendix B: Partial completeness of MLCM

In this section we sketch the proof of completeness of **MLCM** for the fragment without repetition. The argument runs via an extension of the Henkin construction given above. (Observe that the Henkin constants are *not* allowed as the main constant or function in creation or modification statements.) In order to deal with the modal operators, we add the following step to the model construction:

The Henkin model $H =< \mathcal{C}, W, R >$ is defined as follows.

- $W = \{\Gamma \mid \Gamma$ maximally consistent in $\mathcal{L}^{+}$ and has witnesses in $\mathcal{C}\}$;
- $\sim_{\Gamma} = \{(c,d) \in \mathcal{C}^{\uparrow} \times \mathcal{C}^{\uparrow} | (c \simeq d) \in \Gamma\}$;
- **for all** $f \in \text{FUNC}, v \in \mathcal{C}^{\uparrow}/\sim_{\Gamma}$ we have $f_{\Gamma}(v) = \{d \in \mathcal{C}^{\uparrow} | (fc \simeq d) \in \Gamma$ **for some** $c \in v\}$;
- **for all** $p \in \text{PRED}$ we have $p_{\Gamma} = \{d \in \mathcal{C}^{\uparrow} | pd \in \Gamma\}$
- **for all** $\alpha \in \text{PROG}, a \in \text{ASS}$ we have $R_{\alpha,a} =_{\text{def}} \{(\Gamma, \Gamma') \mid$ **forall** $A \in \mathcal{L}^{+}(\; a([\alpha]A) \in \Gamma \Rightarrow a(A) \in \Gamma')\}$

In order to show (cf. 5 in Appendix A)

$$\Gamma, a \models A \Leftrightarrow a(A) \in \Gamma$$

we proceed as in Appendix A. The base caes, $A = (s = t)$ and $A = pt$, are treated likewise. The same holds for the induction steps except for $A = [\alpha]B$. For this step, we need the following. For all sentences A we have:

$$[\alpha]A \in \Gamma \Leftrightarrow \textbf{forall } \Gamma'(\Gamma R_{\alpha,a}\Gamma' \Rightarrow A \in \Gamma') \tag{6}$$

This is proved as follows.

$\Rightarrow$: directly by the definition of $R_{\alpha,a}$.

$\Leftarrow$: by contraposition. Let $[\alpha]A \notin \Gamma$; it suffices to construct a Γ' with $\Gamma R_{\alpha,a}\Gamma'$ and $A \notin \Gamma'$. So put

$$\Gamma'' = \{\neg A\} \cup \{B \mid [\alpha]B \in \Gamma\}.$$

We claim that Γ'' is consistent. To see this, assume the contrary, then we have $\{B \mid [\alpha]B\epsilon\Gamma\} \vdash A$ and (with [**N**]) $\{[\alpha]B \mid [\alpha]B \in \Gamma\} \vdash [\alpha]A$, so $\Gamma \vdash [\alpha]A$, i.e. contradiction. Now extend Γ'' to $\Gamma' \in W$. Then $\Gamma R_{\alpha,a}\Gamma'$ and $\neg A \in \Gamma'$, so $A \notin \Gamma'$. This ends the proof of (6).

The only thing we have to do yet is to show that H is not only a structure but also a model. This is done as follows: let $\Gamma R_{\alpha,a}\Gamma'$, then for all sentences A we have

$$[\alpha]A \in \Gamma \Rightarrow A \in \Gamma' \tag{7}$$

Using this and the properties of **MLCM** we must be able to conclude that $R_{\alpha,a}$ satisfies the requirements in the definition of the semantics. We consider several cases for α.

The case $\alpha = \mathrm{NEW}c$: for the first two requirements we need

$$\begin{aligned} &d \simeq e \in \Gamma \Leftrightarrow \\ &(d \simeq e \in \Gamma' \textbf{ or } (d \simeq \uparrow \wedge e \simeq c) \in \Gamma' \textbf{ or } (e \simeq \uparrow \wedge d \simeq c) \in \Gamma'), \end{aligned} \tag{8}$$

for then $U_{\Gamma'}$ is the extension of U_Γ with the object $c_{\Gamma'}$. Now (8) follows from (consequence **4**), (consequence **5**) and (7).

For the behaviour of functions (the third requirement) we need

$$fd \simeq e \in \Gamma' \Leftrightarrow (fd \simeq e \in \Gamma \textbf{ and } e \not\simeq c \in \Gamma') \tag{9}$$

for then we have that $f_{\Gamma'}c = f_{\Gamma'} \uparrow$ and $f_{\Gamma'}$ behaves like f_Γ elsewhere. Now (9) follows from (consequence **6**), (consequence **7**), $[\mathrm{NEW}c]fx \not\simeq c$ and (7).

The behaviour of predicates (the fourth requirement) is treated likewise.

For $\alpha = (f(s) := t)$ the first requirement states that the universe does not change, and this follows from

$$d \simeq e \in \Gamma \Leftrightarrow d \simeq e \in \Gamma' \tag{10}$$

which is a consequence of [**FM1**] (reading $d \simeq e$ and $d \not\simeq e$ for A) and (7). One easily checks that [**FM 2**] and [**FM 3**] take care of the behaviour of f.

The case $\alpha = (p(s) :\leftrightarrow A)$ is treated likewise.

For $\alpha = A?$, $\alpha = \beta;\gamma$ or $\alpha = \beta \cup \gamma$ the proof is straightforward.