A variant of Quantified Dynamic Logic*

Gerard R. Renardel de Lavalette†
(joint work with André Engels and Rix Groenboom)

1 Background

The research reported here has its roots in the ESPRIT project METEOR (1985-1989) on software specification, financed by the European Community. One of the partners in METEOR was Philips Research Laboratories Eindhoven (PRLE), the Netherlands. In 1983, Hans Jonkers at PRLE initiated the definition of a wide-spectrum specification language with the name COLD (Common Object-oriented Design Language). In the project METEOR, a kernel version COLD-K was worked out by a team in which the author (then at Utrecht University) participated. This led to a full definition of COLD-K in 1987 ([3]). In order to define the semantics of COLD-K, the infinitary logic MPL_ω was developed (see [5]), a variant of $L_{\omega_1\omega}$.

COLD-K combines algebraic and state-based specification: procedures can be defined and assertions about their behaviour can be made as in dynamic logic. It was the basis for further developments at PRLE, leading to a user-oriented version COLD-1 and a software prototyping method PROTOCOLD, which is being used for the development of the software in TV sets. Moreover, a textbook was written ([2]).

2 QDL and MLCM

MLCM (see [4]) intends to formalise reasoning over assertions in COLD, and the definition of the assertion sublanguage of COLD-K was inspired on **QDL**: thus **MLCM** is based on **QDL**. The main difference between **QDL** and **MLCM** is:

in **QDL**, programming variables and logical variables coincide;
in **MLCM**, constants (and functions) play the role of programming

*Extended abstract, handout for the Dagstuhl-Seminar on Specification and Semantics, 8 - 12 July 1996

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

variables (possibly with parameters).

Both in **QDL** and **MLCM** states are models $< U, I, a >$ for first-order logic, consisting of a universe, an interpretation (of the signature elements) and an assignment (mapping variables to elements of U). In **QDL** state changes correspond with changes in a, while U and I are kept constant; in **MLCM** state changes correspond with changes in U (object creation) and/or I (modification of a function or predicate), while a remains unchanged.

3 QDL

Syntax: extension of first-order logic with programs and modal assertions $[\alpha]A$ about programs, with the intended meaning: A holds after every execution of α.

$$\begin{array}{lll} \text{TERM} & t ::= & x \mid f(t_1, \ldots, t_n) \\ \text{PROG} & \alpha ::= & x := t \mid A? \mid \alpha; \alpha \mid \alpha \cup \alpha \mid \alpha^* \\ \text{FORM} & A ::= & (t = t) \mid p(t_1, \ldots, t_n) \mid A \wedge A \mid \neg A \mid \forall x A \mid [\alpha]A \end{array}$$

Semantics: models $M =< U_M, I_M >$ where U_M is a universe of objects and I_M an interpretation of the signature elements (functions and predicates). $W_M = \text{VAR} \to U_M$ is the collection of worlds, and the accessibility relations are defined by

$$R_{x:=t} = \{(a, a[x \mapsto [\![t]\!]_{M,a}] \mid a \in W_M\}$$

4 MLCM

In order to capture assertional reasoning over dynamical aspects of COLD, **MLCM** has to satisfy the following requirements:

- separation of logical and programming variables;
- partial functions;
- three kinds of primitive procedures: object creation, function modification and predicate modification;
- program constructions as in dynamic logic.

Realizing these requirements led to the following design decisions:

- semantics like **QDL**, but with algebras as states;
- basic universe U containing $\star$ (undefined);
- a store $V = \{v_n \mid n \in N\}$ of new objects;

- local universes $U \cup V_n$ (V_n initial segment of V);
- NEW adds the first fresh v_n to the local universe;
- a newly created object behaves like $\star$;
- creation and modification are deterministic and terminating.

Syntax:

$$\begin{array}{lll} \text{TERM} & t ::= & \uparrow \mid x \mid f(t_1, \ldots, t_n) \\ \text{PROG} & \alpha ::= & \text{NEW}c \mid f(t_1, \ldots, t_n) := t \mid p(t_1, \ldots, t_n) :\leftrightarrow A \mid \\ & & A? \mid \alpha; \alpha \mid \alpha \cup \alpha \mid \alpha^* \\ \text{FORM} & A ::= & (t = t) \mid p(t_1, \ldots, t_n) \mid A \wedge A \mid \neg A \mid \forall x A \mid [\alpha] A \end{array}$$

c (in the clause for programs) stands for constant symbols, i.e. function symbols with arity 0. For the sake of simplicity, we assume here and later that functions f and predicates p are unary.

Informal meaning:

- $\uparrow$ is the undefined object;
- NEWc creates a new object and makes c refer to it;
- $fs := t$ changes the value of function f on argument s to the value of t;
- $p(t) :\leftrightarrow A$ changes the value of predicate p on argument t to the truth value of A.

We also define a *definedness* predicate by $t{\downarrow} = (t = t)$ and *weak equality* by $s \simeq t = (s{\downarrow} \vee t{\downarrow}) \rightarrow s = t$.

Substitution $[t/x]A$ of term t for x in A is not always defined in **MLCM**, only if no function symbol in t comes into the scope of a program statement that may change its meaning. So, e.g., $[c/x]([\text{NEW}c](x = y))$ nor $[f(y)/x]([f(s) := t](x = z))$ is defined.

Semantics: a model $M = M(U, V)$ of **MLCM** is generated by its *basic universe* U and its *store* V. They satisfy

$$\star \in U \quad U \cap V = \emptyset \quad V = \{v_n \mid n \in N\}$$

We put $V_n = \{v_k \mid k < n\}$. A *world* (or state) $w =< U_w, I_w >$ in M is a model for first-order logic with universe $U_w = U \cup V_n$ for some $n \in N$ and interpretation I_w mapping signature elements to functions and predicates on U_w of appropriate arity. If $U_w = U \cup V_n$, then $\lambda(w) = n$ is the *order* of w. $W = W_M$ is the collection of all worlds in M.

Extensions and updates of a world $w \in W$ are defined as follows:

- w^+ is the extended world $< U_{w^+}, I_{w^+} >$, satisfying $U_{w^+} = U_w \cup \{v_{\lambda(w)}\}$, $I_{w^+}(f)(u) = I_w(f)(u)$ for $u \in U_w$, $I_{w^+}(f)(v_{\lambda(w)}) = I_w(f)(\star)$, analogously for the interpretation of functions with other arity and predicates;
- $w[fu \mapsto v]$ $(u, v \in U_w)$ is the updated world w', satisfying $U_{w'} = U_w$, $I_{w'}(f)(u) = v$, elsewhere $I_{w'}$ behaves like I_w;
- predicate updates: analogously.

ASS = VAR $\to (U \cup V)$ is the collection of assignments. Pointwise modification $a[x \mapsto u]$ (where $x \in$ VAR, $u \in (U \cup V)$) of $a \in$ ASS is defined as usual. An assignment a can be restricted in a world w to a_w : VAR $\to U_w$ as follows:

$$\begin{array}{rcll} a_w(x) & = & a(x) & \text{if } a(x) \in U_w \\ & = & \star & \text{if } a(x) \notin U_w \end{array}$$

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

$$\begin{array}{lcl} [\![\uparrow]\!]_{w,a} & = & \star \\ [\![x]\!]_{w,a} & = & a_w(x) \\ [\![ft]\!]_{w,a} & = & f_w([\![t]\!]_{w,a}) \end{array}$$

Now $w, a \models A$ (the interpretation of formula A in world w with assignment a) and $R_{\alpha,a}$ (the accessibility relation of program α w.r.t. assignment a) are defined simultaneously:

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

$$\begin{array}{lll} R_{NEWc,a} & =_{\text{def}} & \{(w, w^+[c \mapsto v_{\lambda(w)}]) \mid w \in W\} \\ R_{f(s):=t,a} & =_{\text{def}} & \{(w, w[f_w[\![s]\!]_{w,a} \mapsto [\![t]\!]_{w,a}]) \mid w \in W\} \\ R_{p(t):\leftrightarrow A,a} & =_{\text{def}} & \{(w, w[p_w[\![t]\!]_{w,a} \mapsto (w, a \models A)]) \mid w \in W\} \\ R_{A?,a} & =_{\text{def}} & \{(w, w) \mid w, a \models A\} \\ R_{\alpha;\beta,a} & =_{\text{def}} & R_{\alpha,a} \circ R_{\beta,a} \\ R_{\alpha\cup\beta,a} & =_{\text{def}} & R_{\alpha,a} \cup R_{\beta,a} \\ R_{\alpha^*,a} & =_{\text{def}} & R^*_{\alpha,a} \end{array}$$

And now the axiomatisation of **MLCM**. Expressions containing $\Rightarrow$ are rules, the others are axioms. We do not attempt to provide a minimal set of axioms and rules. $[\alpha^n]$ is recursively defined by $[\alpha^0] = [\top?]$ and $[\alpha^{n+1}] = [\alpha][\alpha^n]$. $[\alpha]\Gamma$ stands for $\{[\alpha]A \mid A \in \Gamma\}$; analogously for $[t/x]\Gamma$. Observe that **INF** is a proof rule with countably many premises, so we have an infinitary proof system.

Taut	All tautologies of propositional logic
MP	$\Gamma \vdash A$ and $\Gamma \vdash A \to B \Rightarrow \Gamma \vdash B$
Ded	$\Gamma, A \vdash B \Rightarrow \Gamma \vdash A \to B$
Eq	$x \simeq y \to y \simeq x$ $x \simeq y \wedge y \simeq z \to x \simeq z$ $x \simeq y \to fx \simeq fy \wedge (px \leftrightarrow py)$
Undef	$\neg(\uparrow\downarrow)$
Quan	$\Gamma, x\downarrow \vdash A \Rightarrow \Gamma \vdash \forall x A$ (x not free in Γ) $(\forall x A \wedge x\downarrow) \to A$
Subst	$\Gamma \vdash A \Rightarrow [t/x]\Gamma \vdash [t/x]A$ (provided all substitutions are defined)
N	$\Gamma \vdash A \Rightarrow [\alpha]\Gamma \vdash [\alpha]A$
Atom	$<\alpha>A \leftrightarrow [\alpha]A$ (for atomic α)
C1	$x = y \leftrightarrow [\mathrm{NEW}c](x = y \neq c)$
C2	$px \leftrightarrow [\mathrm{NEW}c]px$
C3	$[\mathrm{NEW}c]c\downarrow$
C4	$[\mathrm{NEW}c](fx \neq c \ \wedge \ fc \simeq f\uparrow \wedge (pc \leftrightarrow p\uparrow)$
FM1	$A \leftrightarrow [f(s) := t]A$ for all A not containing f
FM2	$s \simeq x \to (t \simeq y \leftrightarrow [f(s) := t]fx \simeq y)$
FM3	$s \not\simeq x \to (fx \simeq y \leftrightarrow [f(s) := t]fx \simeq y)$
FM4	$\forall x[f(s) := t]A \leftrightarrow [f(s) := t](\forall x A)$ (x not free in s, t)
PM1	$A \leftrightarrow [p(s) :\leftrightarrow B]A$ for all A not containing p
PM2	$s \simeq x \to (B \leftrightarrow [p(s) :\leftrightarrow B]px)$
PM3	$s \not\simeq x \to (px \leftrightarrow [p(s) :\leftrightarrow B]px)$
PM4	$\forall x[p(s) :\leftrightarrow B]A \leftrightarrow [p(s) :\leftrightarrow B](\forall x A)$ (x not free in s, B)
?AX	$[A?]B \leftrightarrow (A \to B)$
;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	$\{A \to [\alpha^n]B \mid n \in \mathrm{N}\} \Rightarrow A \to [\alpha^*]B$

$\Gamma \vdash A$ (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 hold then so does the conclusion.

Completeness of **MLCM** has been proved by André Engels (see [1]) via a generalisation of the Henkin construction.

5 Extending MLCM with program joins

In Gurevich's Evolving Algebra, the join of two programs α, β is introduced, with the intended semantics: combine (whenever possible) the effect of α and β. We indicate rather loosely how to extend **MLCM** with this construction. Put

$$R_{(\alpha,\beta),a} = \{(w, ((w_1 - w) \cup (w_2 - w)) \lhd w) \mid wR_{\alpha,a}w_1, wR_{\beta,a}w_2, cons(w_1 - w, w_2 - w)\}$$

where $w_1 - w$, the difference between w_1 and w, contains exactly the effect of α on w, similarly for $w_2 - w$; $cons(w_1 - w, w_2 - w)$ expresses that these two effects are consistent, i.e. do not contradict each other. $\lhd$ is the operation 'completed by', so $(w, ((w_1 - w) \cup (w_2 - w)) \lhd w)$ yields the combined effect of α and β on w.
Axiomatisation:

$[\alpha]A \leftrightarrow [\alpha, \alpha]A$
$[\alpha, \beta]A \leftrightarrow [\beta, \alpha]A$
$[\alpha, (\beta, \gamma)]A \leftrightarrow [(\alpha, \beta), \gamma]A$

$[\alpha]A \rightarrow A \vee [\alpha, \beta]A$
$[\alpha]A \wedge [\beta]A \rightarrow [\alpha, \beta]A \quad (A \text{ elementary})$

A elementary means: $A \equiv p(x_1, \ldots, x_n)$ or $A \equiv (f(x_1, \ldots, x_n) = y)$.

References

[1] A. Engels, Volledigheid en Correctheid van **MLCM** (Completeness and correctness of **MLCM**, in Dutch), Master's thesis, University of Groningen, the Netherlands, 1996.

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

[3] 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).

[4] R. Groenboom, G.R. Renardel de Lavalette, Reasoning about dynamic features in specification languages, in: Semantics of Specification Languages, Utrecht 1993 (D.J. Andrews, J.F. Groote, C.A. Middelburg, eds.), Springer-Verlag (Workshops in Computing) 1994, pp. 340 – 355

[5] 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.