A Logic of Modification and Creation

Gerard R. Renardel de Lavalette*†

Abstract

In this paper, MCL (modification and creation logic) is presented, a variant of quantified dynamic logic (QDL) with enhanced expressivity. In MCL, functions and predicates can be modified by actions $f := \lambda x.t$ and $p := \lambda x.\varphi$, respectively, and new objects can be created by the action Create. This contrasts with QDL, where only the value assignments of variables can be modified. Models of MCL are collections of worlds which are locally models of first-order logic. There is an axiomatisation which is sound and complete.

MCL is inspired on QDL and several other reasoning systems about the effect of actions, such as the specification language COLD, Gurevich's Abstract State Machines (formerly known as Evolving Algebras) and Dynamic Database Logic.

1 Introduction

The purpose of this paper is to present the logic MCL (modification and creation logic) in the context of other dynamic logics and related formalisms. It is intended for any reader who is interested in the logical description of dynamic aspects in natural language, programming languages or theories of action. I will therefore concentrate on the main features and properties of MCL, not on proof details (they are intended to appear elsewhere).

Like QDL, MCL is a multimodal extension of first-order logic, where the modal operator $[\alpha]$ is parametrised with action expressions α. The key feature of MCL is its rich action language, which allows for the definition of actions that act on arbitrary (first-order) structures, not only on simple memory structures like variable assignments (mappings from variables to values). This makes MCL well suited for use in contexts with rich structures. See Fensel et al. 1998, where MCL provides the unifying semantical framework for specification of reasoning in knowledge-based systems.

The rest of this section is an introduction to the main features of MCL, starting with the syntax definition:

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

†The author wishes to thank an anonymous referee for constructive criticism and suggestions for making this paper better readable.

terms	$t ::=$	$x \mid \uparrow \mid f(t) \mid \mathsf{new}$
formulae	$\varphi ::=$	$(t = t) \mid p(t) \mid \varphi \wedge \varphi \mid \neg\varphi \mid \forall x\varphi \mid [\alpha]\varphi$
actions	$\alpha ::=$	$\mathsf{Create} \mid f := \lambda x.t \mid p := \lambda x.\varphi$ $\varphi? \mid \alpha;\alpha \mid \alpha + \alpha \mid \alpha^* \mid \cup\, x.\alpha$

For simplicity, we wrote unary functions and predicates where arbitrary arity is intended, and we shall do so in the sequel. So MCL also has, for example, term $g(t, t')$, formula $q(s, t)$, and the actions $q := \lambda x_1 x_2 x_3.\psi$ and $a := t$. The last action assigns the value of t to nullary function a. Usually, nullary functions are called constants, but that term is not very appropriate in this context, and we call a a *programming variable*, in contrast with the logical variables $x, y, \ldots$ used for quantification. We shall assume that the signature of MCL contains a quaternary function for definition by cases, written in mixfix notation:

$\lambda xyzw.\mathsf{if}\ x = y\ \mathsf{then}\ z\ \mathsf{else}\ w$

The basis of MCL is first order logic with partial functions, so terms may fail to refer to a defined value. ($\uparrow$ is such a term; see 3.1 for more about this.) Beside terms and formulae, we have *actions* in MCL. They refer to a broad category: not only specific actions (like 'put the block on the table'), but also more general descriptions (or, if you want, specifications) of actions: they may involve sequential composition ('put the chair on the table and then the block on the chair'), choice ('put the block on the table or on the chair'), and they may be impossible to perform ('put the block on the ceiling'). Every action α gives rise to the modal operator $[\alpha]$ and its dual $\langle\alpha\rangle$, with the intended meaning

$[\alpha]\varphi$: whenever α is performed successfully, φ holds;
$\langle\alpha\rangle\varphi$: it is possible to do α in such a way that φ holds.

So, in MCL we can reason about the *effect* of actions *after (successful) termination*. Observe that there are no other ways in MCL to construct formulae from actions, so other aspects of actions are swept under the carpet, such as effects of unsuccessful attempts and effects during the performance of an action (these aspects are treated in process theories and temporal logic). As a consequence, all impossible actions and all nonterminating actions are identified in MCL, since they have no effect. Another consequence is that actions without effect are not an alternative in cases of choice: so 'put the block on the table or on the ceiling' gets the same meaning as 'put the block on the table'.

What is to be changed by the actions? Answer: the universe, and the signature elements (predicates and functions). We have the following atomic actions

Create: add a fresh element to the universe

$f := \lambda x.t$: let function symbol f refer to $\lambda x.t$
$p := \lambda x.\varphi$: let predicate symbol p refer to $\lambda x.\varphi$.

In order to manipulate the newly added element after Create, MCL has the atomic term new, with the following dynamic interpretation: after Create, new refers to the fresh element just added to the universe. So we then can handle the new element, giving it the name a by $a :=$ new, a specific property p by $p := \lambda x.(px \vee x =$ new$)$, or sending it to some other value b via function f by $f := \lambda x.$if $x =$ new then b else fx.

The atomic actions in MCL have the following properties:

- they are local, i.e. they only change a specified part of the state;
- they are deterministic, i.e. they can be performed in at most one way;
- they are terminating, i.e. they can be performed in at least one way.

Making actions local is the natural way to deal with the frame problem (first formulated in McCarthy and Hayes 1969): What changes when there is change, and what remains unchanged? Determinism and termination lead to predictability of atomic actions, which is an attractive property when constructing complex actions (or programs) since it localises all impredictability in the choice operators $+$,* and $\cup$. We indicate briefly how these properties of the atomic actions are realised, referring to Section 3 for a full treatment.

Modification. For modifications $f := \lambda x.t$, locality is straightforward: only the interpretation of the signature element that is modified (f in the example) may change, the universe and the interpretation of the other signature elements remain the same. Modifications can be read as explicit definitions, which are deterministic by their very nature. What about termination? One may attempt a counterexample like $p := \lambda x.\neg p(x)$, but this is completely unproblematic, for the new extension of p is the complement of the old extension of p. Compare this with the assignment $a := a + 1$, which is a perfectly normal action (assuming that a is a number) and only looks weird if $:=$ is misread as $=$.

Creation. For creation, locality is somewhat more subtle. The universe is extended with a new element, so we have to do something to adapt the interpretation of the functions and predicates. We choose for the general rule: *new elements behave like undefined*. The interpretation of signature elements, when applied to old elements in the universe, remains unchanged. As a consequence, Create is deterministic. Termination presupposes that fresh objects are always available: in the semantics this is realised with a store of countably many fresh objects. See 3.3 for the details.

There are also actions that do not change anything (hence are deterministic): tests $\varphi?$, succeeding when φ holds, failing otherwise. Moreover, new actions can be built from others with the following action operators:

$\alpha;\beta$	do first α, then β;
$\alpha+\beta$	do either α or β;
α^*	do α zero or more times;
$\cup x.\alpha$	do α for some value of x.

Observe that $+,^*$ and $\cup$ have a choice character. Some well-known programming constructions can be expressed:

$$\begin{array}{lcl} \mathsf{if}\ \varphi\ \mathsf{then}\ \alpha\ \mathsf{else}\ \beta\ \mathsf{fi} & = & (\varphi?;\alpha)+(\neg\varphi?;\beta) \\ \mathsf{while}\ \varphi\ \mathsf{do}\ \alpha & = & (\varphi?;\alpha)^*;\neg\varphi? \\ \mathsf{do}\ \alpha\ \mathsf{until}\ \varphi & = & (\alpha;\neg\varphi?)^*;\varphi? \end{array}$$

Observe that these defined operators all preserve determinism, although they contain choice operators. Some other possibilities are

do α such that φ:	$\alpha;\varphi?$
do α provided that it terminates:	$((\langle\alpha\rangle\top)?;\alpha)+(\neg\langle\alpha\rangle\top)?$

The expressiveness of MCL is demonstrated by the following definition of the natural numbers (modulo isomorphism, provided there is a zero 0 and a successor function s):

$$N := \lambda x.\langle a := 0;(a := s(a))^*\rangle x = a$$

As a consequence, Peano arithmetic can be embedded in MCL. Moreover, many inductive definitions can be expressed in MCL, for example the transitive closure T of a binary relation S:

$$T := \lambda xy.\langle Q := \lambda xy.\bot;(Q := \lambda xy.(Sxy \vee Txy))^*\rangle Qxy$$

1.1 Survey of the rest of this paper

After this general introduction, we turn to the ancestors and relatives of MCL in the next section. Then, in section 3, we present the semantics and axiomatisation in three stages: first the static part, then modification and finally creation. Section 4 contains some properties of MCL: completeness, failure of compactness and interpolation, elimination of Create. Most proofs are only sketched. In section 5 possible extensions of MCL are discussed, such as parallel modifications, logic programming and removing objects. We conclude with some suggestions for further work.

2 Ancestors and Relatives of MCL

MCL is the first nor the only reasoning system about the effect of actions. In this section, we discuss several ancestors (sources of inspiration for MCL) and relatives of MCL.

2.1 Quantified Dynamic Logic

Pratt's dynamic logic (Pratt 1976) is the first modal logic extension of first order logic intended to model and reason about actions (usually called programs). Like MCL, it has modal operators $[\alpha]$ and the program operators $?,;,,+$ and $\star$. Unlike MCL, QDL has only total functions, and its programs

change the assignment of variables to objects, not the interpretation of functions and predicates. Atomic actions in QDL are of the form $x := t$, with the effect that x refers to the value of t. Moreover, $\cup x.\alpha$ is not available in QDL (but can be added easily). Harel 1984 is an extensive survey of propositional dynamic logic PDL and quantified dynamic logic QDL; an excellent introduction can be found in Goldblatt 1992.

We sketch the semantics and axiomatisation of QDL: much of it applies to MCL, too. Let M be a model for the first-order part of QDL. In order to interpret the variables, assignments are used which map variables to values in the universe U of M. Now M is extended to a model for full QDL by taking $\mathsf{Var} \to U$, the collection of all variable assignments, as its state space (i.e. collection of possible worlds). The accessibility relations R_α on $\mathsf{Var} \to U$ are defined inductively by

$$\begin{array}{lll} R_{x:=t} & =_{\text{def}} & \{(A, A[x \mapsto [\![t]\!]_{A[x \mapsto u]}]) \mid A \in \mathsf{VA}\} \\ R_{\varphi?} & =_{\text{def}} & \{(A, A) \mid A \models \varphi\} \\ R_{\alpha;\beta} & =_{\text{def}} & R_\alpha \circ R_\beta \\ R_{\alpha+\beta} & =_{\text{def}} & R_\alpha \cup R_\beta \\ R_{\alpha^*} & =_{\text{def}} & R_\alpha^* \end{array}$$

$R_{x:=t}$ is a total functional relation which changes as assignment on its value for x. $R_{\varphi?}$ is a partial identity function, i.e. a subset of the identity relation. Furthermore, sequential composition ';' corresponds to relation composition, nondeterministic choice corresponds to relation union, and repetition * is modeled by transitive closure

$$R^* = \{(A, A') \mid A = A' \ \vee \ \exists A_1 \ldots A_n((A, A_1) \in R \wedge \ldots \wedge (A_n, A') \in R)\}.$$

The interpretation of modal formulae reads

$$M, A \models [\alpha]\varphi = \forall A'((A, A') \in R_\alpha \to M, A' \models \varphi).$$

As a consequence, the following principles are valid:

$$\begin{array}{lcl} [x := t]\varphi(x) & \leftrightarrow & \varphi(t) \\ [\varphi?]\psi & \leftrightarrow & (\varphi \to \psi) \\ [\alpha;\beta]\varphi & \leftrightarrow & [\alpha][\beta]\varphi \\ [\alpha + \beta]\varphi & \leftrightarrow & ([\alpha]\varphi \wedge [\beta]\varphi) \\ [\alpha^*]\varphi & \leftrightarrow & (\varphi \wedge [\alpha][\alpha^*]\varphi) \end{array}$$

We see that actions change variable assignments, and they do so locally in a controlled way: an action can only change assignments w.r.t. the variables that are explicitly mentioned. So, whatever may happen when $\varphi? + (x := s; y := t)^*$ is performed, we know that at most the values of x and y will be changed, not of any other variable.

Variables play a double role in QDL, being at the same time vehicle for quantification and for assignment. (These roles correspond to the distinction made in programming practice between logical variables and programming variables.) Consider $\forall x[x := x + 1]p(x)$: the x in $x + 1$ is bound by

$\forall x$, the x in $p(x)$ by $[x := \ldots]$. An equivalent and less confusing variant is $\forall x[y := x + 1]p(y)$.

A closer look at $\forall x$ and $[x := \ldots]$ reveals that it is possible to reconcile their apparently diverse meanings, viz. by reading $\forall x$ as $[x :=?]$ (and, dually, $\exists x$ as $\langle x :=? \rangle$), where the question mark ? refers to an arbitrary object. More formally, we have

$$R_{x:=?} = \{(s, s[x \mapsto u]) \mid s \in S, u \in U\}$$

which immediately yields $[x :=?]\varphi \leftrightarrow \forall x\varphi$ and $\langle x :=? \rangle\varphi \leftrightarrow \exists x\varphi$. This idea (as old as dynamic logic itself, see Pratt 1976) reduces quantification to modality: see van Benthem 1996, Ch.9 for a deconstruction of first-order logic along these lines, yielding a plethora of (sometimes decidable) subsystems.

We close this subsection with the observation that QDL can be embedded straightforwardly in MCL by systematically replacing variables x involved in assignment by nullary functions a: so $[x := f(y)]p(x, y)$ becomes $[a := f(y)]p(a, y)$, for example. (There are some subtle details, however: consider $\forall x[(x := f(x))^*]p(x)$, which translates to $\forall x[a := x; (a := f(a))^*]p(a)$.)

2.2 Two Software Specification Formalisms: COLD and ASM

Software specification, a subdiscipline of Software Engineering, intends to provide high-level, abstract descriptions of the intrinsic properties of software systems to be developed. Algebraic specification is one of the paradigms for attaining the required level of abstraction, which has as a distinctive feature that it concentrates on a purely functional description of input-output behaviour, abstracting away from the notion of state. It is a fruitful area of research and development (see Wirsing 1990), but the gap between sound but abstract theory and the more traditional state-based practice remains to be considered as a hindrance.

In the '80s, several attempts were proposed to bridge this gap by extending the algebraic paradigm with a notion of state and state change. Two of these attempts, under the acronyms COLD (Common Object-oriented Language for Design) and ASM (Abstract State Machines, formerly known as Evolving Algebra), will be presented here.

COLD was developed at Philips Research, mainly by Hans Jonkers (see Feijs et al. 1987b, Feijs et al. 1987a, Feijs and Jonkers 1992). ASM is proposed and developed by Yuri Gurevich (see Gurevich 1988, Gurevich 1991, Gurevich 1995), and widely applied and extended by Egon Börger and others (see Börger and Huggins 1998 for an extensive bibliography). The central idea behind both is the generalisation of the machine models in automata theory. An automaton (or machine) is an abstract entity with a number of possible states and transitions from states to states (and often some notion of input and output). The Turing machine is the oldest and

most famous example, where the state consists of the internal state of the machine and the sequence of symbols on its tape. There are also simpler machines (finite automata, for example, with only a finite number of states) and more complex machines (register machines, random access machines, etc., see Hopcroft and Ullman 1979 for a survey).

Now the new idea of COLD and ASM is the following: instead of some specific structure for the state of a machine (a one-way infinite tape, a collection of registers for numbers, etc.), take an arbitrary structure, described with logico-algebraical means (sorts, functions, predicates, equations, axioms), and change that state by changing the sorts, functions and predicates. Primitive state-changing actions in COLD and ASM are: extend a sort with a fresh object; change a function or predicate for some specific value of its argument(s). The usual program operators of PDL apply, and others (parallel composition) have been investigated in the context of ASM.

MCL is a generalisation of other logics based on COLD and ASM. MLCM (Groenboom and Renardel de Lavalette 1994) was the first such attempt and can be characterised as MCL without choice quantification of programs (i.e. the construct $\cup x.\alpha$) and with only *pointwise* modifications $f(t) := s$ and $p(t) :\leftrightarrow \varphi$. An attempt to formalise evolving algebras based on MLCM is given in Groenboom and Renardel de Lavalette 1995. MLPM (Fensel and Groenboom 1996) is a variant of MLCM with two kinds of bulk (i.e. non-pointwise) updates for predicates, viz. $p := \lambda x.\varphi$ (as in MCL) and $p := \varepsilon x.\varphi$. The intended meaning of the latter is: p becomes a singleton predicate that holds only for one nondeterministically chosen object x satisfying φ. This can be expressed in MCL as follows:

$$p := \varepsilon x.\varphi \;=\; \cup x.(\varphi?; p := \lambda y.(x = y))$$

Related work is in Tonino 1997, where a theory of many-sorted evolving algebras is developed comparable with MLCM, but extended with contraction updates that remove an object form a sort (see also 5.3).

2.3 Dynamic Database Logic

The contents of a (relational) database can be described straightforwardly in first-order logic: relations become predicates, attributes become functions. Extending this to database *updates* leads in Spruit et al. 1995 to the development of the logics PDDL and DDL, (propositional) database dynamic logic. In PDDL, the atomic actions are Ip (passive insertion), Dp (passive deletion), $I^H p$ (active insertion) and $D^H p$ (active deletion). Passive actions only affect the truth value of p; active actions, moreover,

perform the logic program H. With help of the notation introduced in Section 5, we can formulate this in MCL:

$$\begin{array}{lcl} Ip & \mapsto & p := \top \\ Dp & \mapsto & p := \bot \\ I^H p & \mapsto & (p := \top); \alpha^H \\ D^H p & \mapsto & (p := \bot); \alpha^H \end{array}$$

Here α_H is the least fixpoint of the predicate operator associated with H. In the predicate logic DDL, the atomic actions are:

$\&x\, Ip\, t$ where φ	(insertion) make p true for all instances of t with values of x for which φ holds;
$\&x\, Dp\, t$ where φ	(deletion) make p false for all instances of t with values of x for which φ holds;
$\&x\, Up\, t \to t'$ where φ	(update) make, for all values of x for which φ holds, p false for the corresponding instance of t and true for the corresponding instance of t';
$fs := t$	(assignment) make $f(s)$ equal to t.

Moreover, DDL has the conditional choice construct

$+x\,\alpha$ where φ: do α for one of the values of x for which φ holds.

All these constructions can be expressed in MCL:

$$\begin{array}{lcl} \&x\, Ip\, t \text{ where } \varphi & \mapsto & p := \lambda y.(py \vee \exists x(\varphi \wedge y = t)) \\ \&x\, Dp\, t \text{ where } \varphi & \mapsto & p := \lambda y.(py \wedge \neg\exists x(\varphi \wedge y = t)) \\ \&x\, Up\, t \to t' \text{ where } \varphi & \mapsto & p := \lambda y.((py \wedge \neg\exists x(\varphi \wedge y = t)) \\ & & \qquad \vee \exists x(p(t) \wedge \varphi \wedge y = t')) \\ fs := t & \mapsto & f := \lambda x.\mathsf{if}\ x = s\ \mathsf{then}\ t\ \mathsf{else}\ fx\ \mathsf{fi} \\ +x\,\alpha \text{ where } \varphi & \mapsto & \cup x.(\varphi?; \alpha) \end{array}$$

2.4 Tarskian Variations

In van Benthem and Cepparello 1994, the process of dynamification of logics is considered from the perspective of Tarski's truth definition

$$D, I, A \models \varphi$$

stating that formula φ holds in model $M = \langle D, I, A\rangle$ with domain D, interpretation I and variable assignment A. Now dynamification comes down to 'Tarskian variation', where one or more of the parameters D, I, A are varied.

Variation of A is studied extensively, leading to a variety of logics: QDL, DPL (dynamic predicate logic, Groenendijk and Stokhof 1991), UL (update logic, Veltman 1996) and DEL (dynamic epistemic logic, Gerbrandy 1999, Gerbrandy and Groeneveld 1997). Unlike the others, DPL and UL have no category of action expressions, but their formulae have a dynamic meaning expressing *change potential*, i.e. the ability to change anaphoric references or information states, respectively.

The logic TV formulated in van Benthem and Cepparello 1994 involves modification of all three parameters D, I, A via the following actions:

ηx (change the value of A at x)
μ (change the interpretation I of the signature elements)
δ (change the domain D)

Observe that ηx yields local change, while μ involves global change; furthermore, all three atomic actions are highly nondeterministic. Compare this with QDL and MCL, where all atomic actions are local and deterministic.

Local versions of μ are straightforward: μp, 'change I at p'. Observe that $[\mu p]$ has the same meaning as the second-order quantifier $\forall p$. Furthermore, there are the variants $\delta\downarrow$ ('shrink the domain'), $\delta\uparrow$ ('extend the domain').

TV is interpreted in collections W of worlds $w = \langle D_w, I_w, A_w\rangle$. The accessibility relations of the atomic actions $\eta x, \mu p, \mu, \delta$ are the so-called shift relations $=_x, =_p, =_I, =_D$, respectively, defined by

$$\begin{array}{lcl} w =_x w' & : & D_w = D_{w'}, I_w = I_{w'}, A_w(y) = A_{w'}(y) \text{ for all } y \neq x \\ w =_p w' & : & D_w = D_{w'}, A_w = A_{w'}, I_w(q) = I_{w'}(q) \text{ for all } q \in \Sigma - \{p\} \\ w =_I w' & : & D_w = D_{w'}, A_w = A_{w'} \\ w =_D w' & : & I_w = I_{w'}, A_w = A_{w'} \end{array}$$

Deterministic change of A is realisable in TV via $\eta x; x = t?$, provided that x does not occur in t. But that variable condition is restrictive: sometimes we want to define something new in terms of something old (the paradigmatic example is $x := x + 1$). In QDL and MCL this can be expressed directly, but in the case of TV, we have to do something like $\eta y; y = x?; \eta x; x = t[y/x]?$ with y a fresh variable.

3 Semantics and Axiomatisation

We present the semantics and the axioms of MCL in three steps: first the static part, then the modification part, finally the creation part. We assume that the signature Σ of MCL is divided in a static part Σ_s and a dynamic part Σ_d: only the elements of Σ_d are allowed in the left hand side of modification actions.

3.1 The static part

We assume that $\Sigma_d = \emptyset$, i.e. all signature elements are static. The static part of MCL is just first-order logic with partial functions. Its semantics is given in terms of models $M = \langle U, \star, I\rangle$ consisting of universe U, undefined object $\star \notin U$ and interpretation I of the elements of $\Sigma = \Sigma_s$. $\star$ is used to interpret partial functions as total functions on $U \cup \{\star\}$.

We follow the tradition that quantifiers only range over existing objects, whereas free variables may also refer to nonexisting objects (see Scott 1979 and Troelstra and van Dalen 1988, 2.2; in Beeson 1985, both free and bound variables refer to existing objects only). So $\forall x(x = x)$ is

valid, whereas $x = x$ is not. The convention w.r.t. free variables enables us to formulate axiom schemata involving arbitrary terms that may be undefined: for if $\varphi(x)$ is valid, then so is $\varphi(t)$ for every term t. It also allows for empty domains: $\exists x(x = x)$ is not valid (but, of course, $[\mathsf{Create}]\exists x(x = x)$ is valid, see below). The existence predicate E, defined by

$$\mathsf{E}t := \exists x(x = t)$$

is used in the instantiation axiom and the principle of universal generalisation:

Inst $(\forall x\varphi \wedge \mathsf{E}x) \to \varphi$

UG if $\Gamma, \mathsf{E}x \vdash \varphi$ and x does not occur free in Γ, then $\Gamma \vdash \forall x\varphi$

Here (and in the sequel) Γ is some collection of formulae.

In this standard setup for the semantics for partial functions, there is some design freedom in what to do when $\star$ acts as an argument. One option is to work with *strict* functions, which yield undefined whenever one of their arguments is undefined, so we have $\mathsf{E}f(s,t) \to \mathsf{E}s \wedge \mathsf{E}t$. This leads to proliferation of undefinedness: every term containing an undefined subterm will be undefined. However, there are realistic examples of nonstrict functions: constant functions, multiplication ($x \cdot 0$ is always 0), projection ($\lambda xy.x$), definition by cases (if φ then s else t). We take a permissive stance here and allow nonstrict functions in the semantics of MCL. So $I(f)(\star) \in U$ is allowed.

Another design choice concerns the evaluation of equality (and other predicates) for terms containing partial functions. Several logics exist that have a third truth value *undefined*, interpreting equality as a strict predicate. This opens many options for the definition of the logical connectives, leading to a proliferation of alternative partial logics. The first such logics by Łukasiewicz and Post date from the '20s, and were followed by proposals by Kleene and Bochvar; see the survey papers Urquhart 1986, Blamey 1986. The alternative is to stay within classical logic with two truth values, roughly by identifying the undefined truth value with false: so $t = t$ is true only when t has a defined value, and false otherwise. This is the approach in Scott 1967, Scott 1979, also adopted in COLD (see Koymans and Renardel de Lavalette 1989 for the logic underlying COLD).

Here we choose to stay close to classical logic and to work with the traditional truth values, interpreting equality as a nonstrict predicate satisfying

Eq $\forall x(x = x) \wedge (x = y \to y = x) \wedge (x = y \wedge y = z \to x = z)$

So we sacrificed full reflexivity, but that property is met by *weak equality* $\simeq$:

$$(s \simeq t) := (Es \vee Et) \to s = t$$

which is to satisfy the congruence properties:

Congr $x \simeq y \to fx \simeq fy$ (for all functions f)
$(x \simeq y \wedge px) \to py$ (for all predicates p)

Finally, there is an axiom for undefined:

Undef $\neg(x = \uparrow)$

3.2 The modification part

We turn to the modification part, so we forget about Create for the time being. The fundamental idea is: modification is realised by transition from some *world* $w = \langle U, \star, I\rangle$ to another world. So, for the semantics we need a collection W of worlds, and accessibility relations $R \subseteq W^2$ for the interpretation of actions. Neither the universe nor the static part of the interpretation will change under the modifications, so $U, \star, I_s$ are the same for all elements of W. As a consequence, we may identify W with a collection of interpretations I_d of the dynamic signature elements, and we have models of the form $M = \langle U, \star, I_s, W\rangle$. M is called a *natural model* when W is the collection of all possible interpretations of Σ_d in $U \cup \{\star\}$. In the sequel, we mainly work with natural models.

A consequence of having constant universes is the so-called Barcan formula for atomic modifications μ (not containing x free):

Barcan $\forall x[\mu]\varphi \leftrightarrow [\mu]\forall x\varphi$

Atomic modifications are to be local, so if $p := \lambda x.\varphi$ brings us from I to J then I, J should behave the same on $\Sigma_d - \{p\}$, i.e. $I =_p J$ (with $=_p$ as defined in 2.4). Analogously for $f := \lambda x.t$. So the following frame conditions are satisfied:

FrC $\varphi \leftrightarrow [f := \lambda x.t]\varphi$ (f not in φ)
$\varphi \leftrightarrow [p := \lambda x.\psi]\varphi$ (p not in φ)

The desired behaviour of atomic modifications is captured as follows:

Mod $[f := \lambda x.t](fx = y) \;\leftrightarrow\; t = y$
$[p := \lambda x.\varphi]px \;\leftrightarrow\; \varphi$

for all f, p, all φ and all y, t with y not free in t. Observe that it is not reasonable to expect

$$[a := t]a = t,$$
$$[p := \lambda x.\varphi]\forall x(px \leftrightarrow \varphi)$$

in case a occurs in t and p in φ. To see this, take $a + 1$ for t, $\neg p(x)$ for φ and we get

$$[a := a + 1]a = a + 1,$$
$$[p := \lambda x.\neg px]\forall x(px \leftrightarrow \neg px)$$

which is very undesired indeed.

3.3 The creation part

Finally we deal with Create and expanding universes. We distinguish between the static and the dynamic part of the universe. The static part $U \cup \{\star\}$ is there *ab initio*, being the habitat of the interpretations of the static signature elements. The dynamic part consists of the newly created elements. They are provided by a store $V = \{v_1, v_2, v_3, \ldots\}$: a Create action adds the first fresh store element to the universe. So the dynamic part of the universe is an initial segment V_n of V, where n corresponds with the number of applications of Create.

As a consequence, a world w has a universe $U_n = U \cup \{\star\} \cup V_n$ for some n, and an interpretation I_d of the dynamic signature elements in that universe. So any world can be represented as $w = \langle n, I_d \rangle$, and a model is a tuple $\langle U, \star, V, I_s, W \rangle$ where W is a collection of worlds. I_s is extended to an interpretation of the static signature in world w by the principle *a new element behaves like undefined*, so $I_s(f)(v_n) = I_s(f)(\star)$ for all n. A model is natural when W is maximal, i.e. contains all pairs $\langle n, I_d \rangle$ with $n \in N$ and I_d an interpretation of Σ_d into U_n.

Now that we have models with expanding universes, what to do with variable assignments? The local solution of TV, see 2.4 (each world has its own variable assignment) does not work well here, and we choose for the global option: assignments map the variables into the global universe $U \cup \{\star\} \cup V$, and they are relativised to A_w w.r.t. world w by treating not yet created elements as undefined:

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

The interpretation of new in world $\langle n, I_d \rangle$ is simply v_n, and Create is interpreted by

$$\{((n, I_d), (n + 1, I'_d)) \in W^2 \mid I_d = I'_d \restriction U_n, \forall \sigma \in \Sigma_d I'_d(\sigma)(v_{n+1}) = I'_d(\sigma)(\star)\}$$

Observe that the last part of this definition indeed reads *the new element behaves like undefined*, and that $I_d = I'_d \restriction U_n$ implies that v_{n+1} is not in the range of $I'_d(f)$ for any $f \in \Sigma_d$. This leads to the following axioms **C1-5**:

C1 $x = y \leftrightarrow [\mathsf{Create}](x = y \neq \mathsf{new})$

C2 $px \leftrightarrow [\mathsf{Create}]px$

C3 $[\mathsf{Create}]\mathsf{E}(\mathsf{new})$

C4 $[\mathsf{Create}](f(x) \neq \mathsf{new} \ \wedge f(\mathsf{new}) \simeq f(\uparrow))$

C5 $[\mathsf{Create}](p(\mathsf{new}) \leftrightarrow p(\uparrow))$

C1-2 say: Create does not affect any predicate, except equality w.r.t. new. **C3** says that, after Create, new refers to an existing object (which differs from all old objects, by **C1**). Finally **C4-5** say that, after Create, new does not occur in the range of a function, and behaves like undefined w.r.t. predicates and functions.

As explained in Section 1, we want all atomic actions π (i.e. modifications and creations) to be deterministic and terminating. This is axiomatised by

Det $\langle\pi\rangle\varphi \rightarrow [\pi]\varphi$

Term $\langle\pi\rangle\top$

In natural models, these are valid.

3.4 Interpretation of MCL

For completeness' sake, we give the interpretation of terms, formulae and programs in world $w = \langle n_w, I_w\rangle$ from model $M = \langle U, \star, V, I_s, W\rangle$, and M-assignment A. For brevity, we shall write σ_w for $I_w(\sigma)$ (if $\sigma \in \Sigma_d$) or $I_s(\sigma)$ (if $\sigma \in \Sigma_s$).

$$
\begin{array}{lcl}
[\![x]\!]_{w,A} & = & A_w(x) \\
[\![\uparrow]\!]_{w,A} & = & \star \\
[\![\mathsf{new}]\!]_{w,A} & = & v_{n_w} \\
[\![ft]\!]_{w,A} & = & f_w([\![t]\!]_{w,A})
\end{array}
$$

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

$$
\begin{array}{lll}
R_{\mathsf{Create},A} & =_{\mathrm{def}} & \{(w,w') \in W^2 \mid I_w = I_{w'} \restriction U_{w_n}, \\
 & & \quad \forall \sigma \in \Sigma_d I_{w'}(\sigma)(v_{n+1}) = I_{w'}(\sigma)(\star)\} \\
R_{f:=\lambda x.t,A} & =_{\mathrm{def}} & \{(w,w') \in W^2 \mid w =_f w', \\
 & & \quad \forall u \in U_{w'}(I_{w'}(f)(u) = [\![t]\!]_{w,A[x\mapsto u]})\} \\
R_{p:=\lambda x.\varphi,A} & =_{\mathrm{def}} & \{(w,w') \in W^2 \mid w =_p w', \\
 & & \quad \forall u \in U_{w'}(I_{w'}(p)(u) \Leftrightarrow w, A[x \mapsto u] \models \varphi)\} \\
R_{\varphi?,A} & =_{\mathrm{def}} & \{(w,w) \mid w, A \models \varphi\} \\
R_{\alpha;\beta,A} & =_{\mathrm{def}} & R_{\alpha,A} \circ R_{\beta,A} \\
R_{\alpha+\beta,A} & =_{\mathrm{def}} & R_{\alpha,A} \cup R_{\beta,A} \\
R_{\alpha^*,A} & =_{\mathrm{def}} & R^*_{\alpha,A} \\
R_{\cup x.\alpha,A} & =_{\mathrm{def}} & \{(w,w') \mid \textbf{exists } u \in (U_w - \{\star\})\, w R_{\alpha,A[x\mapsto u]} w'\}
\end{array}
$$

Observe that the interpretation of actions depends on the variable assignment.

3.5 A proof system for MCL

We give a proof system for MCL, based on sequents $\Gamma \vdash \varphi$. The axioms are: the propositional tautologies, **Inst, EQ, Congr, Undef, Barcan, FrC, Mod, C1-5, Det, Term**, furthermore

?AX $[\varphi?]\psi \leftrightarrow (\varphi \to \psi)$

;AX $[\alpha;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$

+AX $[\alpha+\beta]\varphi \leftrightarrow ([\alpha]\varphi \wedge [\beta]\varphi)$

***AX** $[\alpha^*]\varphi \leftrightarrow (\varphi \wedge [\alpha][\alpha^*]\varphi)$

∪AX $[\cup x.\alpha]\varphi \leftrightarrow \forall x[\alpha]\varphi$ (x not free in φ)

and the repetition axiom

INF $\{[\alpha^n]\varphi \mid n \in \mathrm{N}\} \vdash [\alpha^*]\varphi$

The proof rules of MCL are: **UG**, necessitation

NEC if $\Gamma \vdash \varphi$ then $[\alpha]\Gamma \vdash [\alpha]\varphi$

modus ponens, weakening, the infinitary cut rule

CUT if $\Gamma \vdash \varphi$ for all $\varphi \in \Delta$ and $\Gamma, \Delta \vdash \psi$ then $\Gamma \vdash \psi$

the deduction rule

DED if $\Gamma, \varphi \vdash \psi$ then $\Gamma \vdash \varphi \to \psi$

and finally the substitution rule

Subst if $\Gamma \vdash \varphi$ then $(t/x)\Gamma \vdash (t/x)\varphi$

With this last rule, we have to be careful. It is possible to substitute a term at an occurrence in the scope of an action that changes one or more signature elements of that term. Consider $(a/x)([a := t]px)$: it will result in the formula $[a := t]pa$, where a has been brought in the scope of the program $a := t$. By **FrC**, we have $\vdash px \leftrightarrow [a := b]px$; with the unrestricted substitution rule, we would have $\vdash pa \leftrightarrow [a := b]pa$ and also $\vdash pa \leftrightarrow pb$ (via **Mod**), which does not hold in general. Therefore we restrict the substitution rule to *safe* substitutions, where this kind of (usually undesired) dynamic binding cannot occur. See the Appendix for the formal definition of safe substitution.

4 Properties of MCL

In this section we present some theorems about MCL and weaker theories. Most proofs are sketched.

Theorem 1 *MCL is a normal multimodal logic, i.e. we have*

$$\vdash [\alpha](\varphi \to \psi) \to ([\alpha]\varphi \to [\alpha]\psi)$$
$$\vdash \varphi \;\Rightarrow\; \vdash [\alpha]\varphi$$

Proof. This follows directly from **MP**, **NEC** and **DED**. □

Theorem 2 *The axiomatisation is sound and strongly complete w.r.t. natural models.*

Proof. Soundness, i.e. the property $\Gamma \vdash \varphi \;\Rightarrow\; \Gamma \models \varphi$, can be proved as usual by induction over the size of a derivation of $\Gamma \vdash \varphi$. For **Subst**, the following substitution properties are used (assuming that (t/x) is a safe substitution for φ and α):

$$\begin{array}{lcl} [\![(t/x)s]\!]_{w,A} & = & [\![s]\!]_{w,A[x \mapsto [\![t]\!]_{w,A}]} \\ w, A \models (t/x)\varphi & \Leftrightarrow & w, A[x \mapsto [\![t]\!]_{w,A}] \models \varphi \\ wR_{(t/x)\alpha,A}w' & \Leftrightarrow & wR_{\alpha,A[x \mapsto [\![t]\!]_{w,A}]}w' \end{array}$$

Strong completeness ($\Gamma \models \varphi \Rightarrow \Gamma \vdash \varphi$) is proved via the construction of maximal consistent sets of formulae in an extension of the language with Henkin constants, along the lines of Goldblatt's proof for the weak completeness of QDL in Goldblatt 1992, ch. 14. In the resulting model, the worlds are characterised by deterministic programs (not containing $+, ^*, \cup$): an essential property is that every program is equivalent with the (infinite) sum of all deterministic programs contained in it. □

Theorem 3 *MCL is not compact.*

Proof. $\{[\alpha^n]\varphi \mid n \in \mathrm{N}\} \vdash [\alpha^*]\varphi$, but there is no finite subset of $\{[\alpha^n]\varphi \mid n \in \mathrm{N}\}$ which entails $[\alpha^*]\varphi$. □

Theorem 4 *MCL fails to satisfy the interpolation theorem.*

Proof. It is a folklore argument that any consistent logic with a finitary syntax and containing Peano arithmetic refutes interpolation, and it runs roughly as follows.

Beth's theorem (stating that every implicitly definable predicate has an explicit definition) is a consequence of interpolation, so it suffices to give an implicitly defiable predicate which has no explicit definition. MCL contains Peano Arithmetic (see the remark at the end of Section 1), so the syntax of MCL can be encoded in MCL, and we can give an implicit definition of a truth predicate. But, according to Tarski's theorem, an explicit truth definition cannot exist (for it leads to contradiction via a diagonal argument). See also Renardel de Lavalette 1989, 6.4. □

The culprit in the last two theorems is the repetition construct *: we conjecture that compactness, and interpolation hold for MCL minus *. This is evidently the case when Create is omitted, too, since the resulting logic is a definitional extension of first-order logic, as we shall see below.

Now we turn to subtheories. Let MCL^{-c} be MCL without Create, with $\uparrow$ added to the static signature and new, E added to the dynamic signature. Models of MCL^{-c} are of the form $\langle U, I, W\rangle$, where U is the universe, I the interpretation of the static signature elements and W a collection of interpretations of the dynamic signature; in natural models, W contains *all* dynamic interpretations over U. The axiomatisation of MCL^{-c} is like that of MCL, but with the usual axioms and rules for equality and quantification (i.e. not referring to E and $\simeq$). So MCL^{-c} extends first-order logic.

Theorem 5 *MCL can be embedded in* MCL^{-c}.

Proof. The idea is to emulate MCL in MCL^{-c}. For this purpose, we add the unary functions s and l to the static and the dynamic signature, respectively. s is to act as a kind of successor function on the part of the universe that plays the role of the store, and l is used to imitate the localisation A_w of assignment A in world w. The required properties of $\uparrow, \mathsf{E}, \mathsf{new}, s, l$ can be expressed in one MCL^{-c}-formula θ. As an example we give the part of θ which enforces that $\{x \mid sx \neq x\}$ can play the role of the store:

$$\forall xy(sx = sy \to x = y) \wedge s(\mathsf{new}) \neq \mathsf{new} \wedge$$
$$[(\mathsf{new} := s(\mathsf{new}))^*](s(\mathsf{new}) \neq\uparrow \wedge \neg \mathsf{E}s(\mathsf{new}))$$

Terms, actions and programs from MCL are interpreted in MCL^{-c} via the mapping $^{-c}$. Its nontrivial defining clauses are

$$\begin{array}{lcl}
x^{-c} & = & lx \\
(s=t)^{-c} & = & \mathsf{E}s^{-c} \wedge \mathsf{E}t^{-c} \wedge s^{-c} = t^{-c} \\
(\forall x\varphi)^{-c} & = & \forall x(\mathsf{E}x \to \varphi^{-c}) \\
([\alpha]\varphi)^{-c} & = & [\alpha^{-c}]\varphi^{-c} \\
\mathsf{Create}^{-c} & = & \mathsf{new} := s(\mathsf{new}); \\
 & & \mathsf{E} := \lambda x.\mathsf{E}x \vee x = \mathsf{new}; \\
 & & l := \lambda x.\mathsf{if}\ x = \mathsf{new}\ \mathsf{then}\ x\ \mathsf{else}\ lx \\
(\cup x.\alpha)^{-c} & = & \cup x.(\mathsf{E}x?;\alpha^{-c})
\end{array}$$

So all variables in the interpretation occur in the scope of l or E. The interpretation of Create reads: let new refer to the 'next' element in the store, extend E with that element and adapt the localisation function l.

The transformation M^{-c}, w^{-c} of MCL-model M and world w is straightforward, and we have, for all M, all w in M and all φ:

$$\begin{array}{lcl}
M^{-c}, w^{-c} \models \theta & & \\
M, w \models \varphi & \Leftrightarrow & M^{-c}, w^{-c} \models \varphi^{-c}
\end{array}$$

Finally we claim

$$\Gamma \vdash \varphi \Leftrightarrow \Gamma^{-c}, \theta \vdash \varphi^{-c}.$$

The $\Rightarrow$ part is proved with induction over the derivation of $\Gamma \vdash \varphi$. The $\Leftarrow$ part is proved via contraposition. Assume $\Gamma \nvdash \varphi$, then (by completeness of MCL) there is a model M with a world w such that $M, w \models \Gamma$ and $M, w \models \neg\varphi$. So there is a model M^{-c} with $M^{-c}, w^{-c} \models \Gamma^{-c}$ and $M^{-c}, w^{-c} \models \neg\varphi^{-c}$ and also $M^{-c}, w^{-c} \models \theta$. So (by soundness of MCL^{-c}) $\Gamma^{-c}, \theta \nvdash \varphi^{-c}$. □

Theorem 6 *MCL can be embedded in L_ω, first-order predicate logic with countably infinite conjunctions.*

Proof. By the previous theorem, it suffices to give an embedding $\cdot^\blacktriangledown : \mathsf{MCL}^{-c} \to \mathsf{L}_\omega$. The embedding uses function substitution $(\lambda x.t/f)$ and predicate substitution $(\lambda x.\varphi/p)$. The essential defining clauses for these substitutions are

$$\begin{array}{lcl}
(\lambda x.t/f)fs & = & ((\lambda x.t/f)s/x)t \\
(\lambda x.\varphi/p)ps & = & (s/x)\varphi
\end{array}$$

Now the definition of $^\blacktriangledown$. The nontrivial defining clauses are those for formulae $[\alpha]\varphi$:

$$\begin{array}{lcl}
([f := \lambda x.t]\varphi)^\blacktriangledown & = & ((\lambda x.t/f)\varphi)^\blacktriangledown \\
([p := \lambda x.\psi]\varphi)^\blacktriangledown & = & ((\lambda x.\psi/p)\varphi)^\blacktriangledown \\
([\psi?]\varphi)^\blacktriangledown & = & \psi^\blacktriangledown \to \varphi^\blacktriangledown \\
([\alpha;\beta]\varphi)^\blacktriangledown & = & ([\alpha]([\beta]\varphi)^\blacktriangledown)^\blacktriangledown \\
([\alpha+\beta]\varphi)^\blacktriangledown & = & ([\alpha]\varphi)^\blacktriangledown \wedge ([\beta]\varphi)^\blacktriangledown \\
([\alpha^*]\varphi)^\blacktriangledown & = & \varphi^\blacktriangledown \wedge ([\alpha]\varphi)^\blacktriangledown \wedge ([\alpha;\alpha]\varphi)^\blacktriangledown \wedge \ldots \\
([\cup x.\alpha]\varphi)^\blacktriangledown & = & \forall x([\alpha]\varphi)^\blacktriangledown
\end{array}$$

So applying $^\blacktriangledown$ removes all programs with the help of substitution. Proving the correctness of the interpretation is straightforward. □

Theorem 7 *MCL without Create and * is a definitional extension of first-order predicate logic.*

Proof. This is is an easy consequence of the proof of the previous theorem, by observing that $\varphi^\blacktriangledown$ is finite whenever * does not occur in φ. □

5 Extensions

5.1 Parallel modifications

When several atomic modification programs affect different signature elements, they can be performed in parallel. E.g., for two function modifications $f := \lambda x.s$, $g := \lambda y.t$ this can be written as $f := \lambda x.s, g := \lambda y.t$. (In the context of evolving algebras, this construct is called the *join*.) It has the following semantics (writing α for $f := \lambda x.s, g := \lambda y.t$):

$$R_{\alpha,a} = \{(w, w[f \mapsto \lambda u \in U_w.[\![s]\!]_{w,a[x\mapsto u]}][g \mapsto \lambda u \in U_w.[\![t]\!]_{w,a[y\mapsto u]}]) \mid w \in W\}$$

satisfying (z not free in s, t and $f, g \notin \mathsf{sig}(A)$)

$$[\alpha]fx = z \leftrightarrow s = z$$
$$[\alpha]gy = z \leftrightarrow t = z$$
$$A \leftrightarrow [\alpha]A$$
$$\forall z[\alpha]B \leftrightarrow [\alpha](\forall z B)$$

See Groenboom and Renardel de Lavalette 1995 for more about this parallel construct.

5.2 Logic programming via least fixpoints of predicate operators

The meaning of a logic program H can be defined as the least fixpoint of a continuous predicate operator associated with H, and this can be expressed in MCL as follows. If $\lambda p.\lambda x.A$ defines a continuous predicate operator, then its least fixpoint $\mathsf{fix}(\lambda p.\lambda x.A)$ is definable in MCL as follows:

$$\mathsf{fix}(\lambda p.\lambda x.A)x \leftrightarrow \langle (p := \lambda x.\bot); (p := \lambda x.A)^*\rangle px$$

Now, if the logic program H defines one unary predicate, and has the form

$$\begin{array}{lcl} p(t_1) & \leftarrow & A_1 \\ p(t_2) & \leftarrow & A_2 \end{array}$$

where A_1 and/or A_2 may contain p, then the meaning of H is the least fixpoint of the continuous predicate operator

$$\lambda p.\lambda x.((x = t_1 \wedge A_1) \vee (x = t_2 \wedge A_2)),$$

hence is expressible in MCL.

If H defines more than one predicate, and has the form

$$\begin{array}{lcl} p(t_1) & \leftarrow & A_1 \\ q(t_2) & \leftarrow & A_2 \end{array}$$

where A_1, A_2 may contain p, q, then its meaning is obtained via the program

$$\alpha^H = (p := \lambda x.\bot, q := \lambda x.\bot); (p := \lambda x.(x = t_1 \wedge A_1), q := \lambda x.(x = t_2 \wedge A_2))^*$$

by taking $\langle \alpha^H \rangle px, \langle \alpha^H \rangle qy$ for the meaning of H.

5.3 Removing objects

Let us now consider the possibility of removing objects with the action $\mathsf{Remove}(t)$. Its intuitive semantics reads: identify $[\![t]\!]$ with the undefined object $\star$. To make this work, the notion of model must be generalised to allow for worlds with local universes of the form $(U \cup V_n) - X$ where X is some finite set. We expect that this can be done in such a way that Remove is both deterministic and terminating. Other expected properties of Remove are:

$$x = y \neq z \leftrightarrow [\mathsf{Remove}(z)]x = y$$

$$[\mathsf{Remove}(t)]p(x) \rightarrow p(x)$$

$$\forall x([\mathsf{Remove}(t)]\mathsf{E}x \leftrightarrow x \neq t)$$

5.4 Adding and removing many objects

Create and Remove handle one object at a time. With the repetition construct *, it is possible to create or remove finitely many objects, viz. with the actions Create^* and $(\cup x.\mathsf{Remove}(x))^*$. (Observe that $\mathsf{Remove}(t)^*$ does not work: it has the same effect as $\mathsf{Remove}(t)$.)

If we want more, we can introduce the actions Add and Remove, corresponding with $\delta\uparrow$ and $\delta\downarrow$ from 2.4. With Add, it is useful to have the unary predicate New available, which holds for all newly added objects: so we , for example, $\forall x([\mathsf{Add}]\neg\mathsf{New}(x))$.

But other variants are also imaginable: Addinf(adding an infinite number of objects to the universe) and $\mathsf{Remove}(\lambda x.\varphi)$ (removing all objects x satisfying φ). Observe that, slightly paradoxically, we do not have

$$[\mathsf{Remove}(\lambda x.\varphi)]\forall x\neg\varphi(x)$$

to see this, take $\varphi := \neg\exists y.Rxy$ and consider a world with universe $\{a, b\}$ where R is interpreted by $\{(a, b)\}$; then the extension of φ is $\{b\}$, so after $\mathsf{Remove}(\lambda x.\varphi)$ we are left with a world with universe $\{a\}$ where R is empty, so φ holds for a.

6 Concluding Remarks

In this paper, we extended first-order logic — the workhorse for the application of logic in computer science, linguistics and artificial intelligence — with features intended for the description, analysis and unification of the numerous dynamic phenomena that occur in these disciplines: modification (of references, situations, contexts, memory states, database contents, information states, etc.) and creation (of pointers, records, referents, assumptions, hypotheses, information items, concepts, etc.). This resulted in the orthogonally designed logic MCL with a sound and complete axiomatisation.

Furthermore, we made a beginning with the unification of logics dealing with dynamics by indicating how QDL and DDL can be embedded in MCL. It will be interesting to try to do the same with DPL and DEL. For DPL, it will be useful to add to MCL expressions $\alpha \leq \beta$ with the intended meaning: all α-alternatives are β-alternatives. Embedding DEL in MCL will be more involved and may require recursively defined actions.

Another direction of research is extension of the action language. Some suggestions: action operators for parallel composition and removal of objects, effects during actions, attemptive actions, actions with so-called modification rights (indicating globally what may be changed by an action and what not, as in COLD).

We end with some more technical problems: prove compactness and interpolation for MCL without repetition, and determine the proof-theoretic strength of MCL (see Harel 1984, 3.2 for related work on QDL).

References

Beeson, Michael J. 1985. *Foundations of Constructive Mathematics.* Berlin: Springer-Verlag.

Blamey, S. 1986. Partial logic. In *Handbook of Philosophical Logic (vol.III)*, ed. D. Gabbay and F. Guenthner. 1–70. Dordrecht: D. Reidel Publishing Company.

Börger, E., and J.K. Huggins. 1998. Abstract State Machines 1988-1998: Commented ASM Bibliography. *Bulletin of the EATCS* 64:105–126.

Feijs, L.M.G., and H.B.M. Jonkers. 1992. *Formal Specification and Design.* Cambridge Tracts in Theoretical Computer Science, Vol. 35. Cambridge: Cambridge University Press.

Feijs, L.M.G., H.B.M. Jonkers, C.P.J. Koymans, and G.R. Renardel de Lavalette. 1987a. Formal definition of the design language COLD-K (Preliminary ver-

sion). ESPRIT document METEOR/t7/PRLE/7. Eindhoven (the Netherlands): Philips Research Laboratories, Apr. (Final version: August 1989).

Feijs, L.M.G., H.B.M. Jonkers, J.H. Obbink, C.P.J. Koymans, G.R. Renardel de Lavalette, and P.H. Rodenburg. 1987b. A survey of the design language COLD. In *ESPRIT'86: Results and Achievements*, ed. Directorate General XIII. 631 – 644. Elsevier Science Publishers (North-Holland).

Fensel, Dieter, and Rix Groenboom. 1996. MLPM: defining a semantics and axiomatisation for specifying the reasoning process of knowledge-based systems. In *Proceedings of the 12th European Conference on Artificial Intelligence (ECAI96)*, ed. W. Wahlster, 423 – 427. Budapest.

Fensel, Dieter, Rix Groenboom, and Gerard R. Renardel de Lavalette. 1998. Modal Change Logic (MCL): Specifying the Reasoning of Knowledge-based Systems. *Data and Knowledge Engineering* 26:243–269.

Gerbrandy, Jelle. 1999. *Bisimulations on Planet Kripke*. Doctoral dissertation, Institute for Logic, Language and Computation, Amsterdam, Jan.

Gerbrandy, Jelle, and Willem Groeneveld. 1997. Reasoning about Information Change. *Journal of Logic, Language and Information* 6:147–169.

Goldblatt, Robert. 1992. *Logics of time and computation*. CSLI Lecture notes, Vol. 7. Stanford: CSLI. (2nd edition).

Groenboom, Rix, and Gerard R. Renardel de Lavalette. 1994. Reasoning about dynamic features in specification languages - a modal view on creation and modification. In *Semantics of Specification Languages*, ed. D.J. Andrews, J.F. Groote, and C.A. Middelburg, 340 – 355. Workshops in Computing. Springer-Verlag.

Groenboom, Rix, and Gerard R. Renardel de Lavalette. 1995. A formalization of evolving algebras. In *Proceedings Accolade '95*, ed. S. Fischer and M. Trautwein, 17 – 28. Amsterdam. Dutch Graduate School in Logic.

Groenendijk, Jeroen, and Martin Stokhof. 1991. Dynamic predicate logic. *Linguistics and Philosophy* 14:39 – 100.

Gurevich, Yuri. 1988. Logic and the challenge of computer science. In *Trends in theoretical computer science*, ed. Egon Börger. 1 – 57. Computer Science Press.

Gurevich, Yuri. 1991. Evolving algebras; a tutorial introduction. *Bulletin of the EATCS* 43:264 – 284.

Gurevich, Yuri. 1995. Evolving algebras 1993: Lipari guide. In *Specification and validation methods*, ed. Egon Börger. Oxford: Clarendon Press.

Harel, David. 1984. Dynamic Logic. In *Handbook of Philosophical Logic*, ed. D. Gabbay and F. Guenthner. 497 – 604. Dordrecht (the Netherlands): D. Reidel Publishing Company.

Hopcroft, John E., and Jeffery D. Ullman. 1979. *Introduction to Automata Theory, Languages and Computation*. Addison-Wesley Publishing Company.

Koymans, C.P.J., and G.R. Renardel de Lavalette. 1989. The logic MPL_ω. In *Algebraic Methods: Theory, tools and applications*, ed. M. Wirsing and J.A. Bergstra. Lecture Notes in Computer Science, Vol. 394, 247 – 282. Springer-Verlag.

McCarthy, J., and P. Hayes. 1969. Some philosophical problems from the standpoint of Artificial Intelligence. In *Machine Intelligence*, ed. B. Meltzer and D. Michie. 463 – 502. Edinburgh University Press.

Pratt, V.R. 1976. Semantical considerations on Floyd-Hoare logic. In *17th annual symposium on foundations of computer science*, 109 – 121. New York. IEEE.

Renardel de Lavalette, Gerard R. 1989. Modularisation, Parametrisation, Interpolation. *Journal of Information Processing and Cybernetics EIK* 25:283 – 292.

Scott, D.S. 1967. Existence and description in formal logic. In *Bertrand Russell, Philosopher of the Century*, ed. R. Schoenman. 181 – 200. London: Allen and Unwin.

Scott, D.S. 1979. Identity and existence in intuitionistic logic. In *Applications of Sheaves*, ed. M.P. Fourman, C.J. Mulvey, and D.S. Scott, Lecture Notes in Mathematics, Vol. 753, 660 – 696. Springer-Verlag.

Spruit, Paul A., Roel J. Wieringa, and John-Jules Ch. Meijer. 1995. Axiomatisation, declarative semantics and operational semantics of passive and active updates in logic databases. *Journal of Logic and Computation* 5:27 – 50.

Tonino, Hans. 1997. *A theory of many-sorted evolving algebras.* Doctoral dissertation, Delft University of Technology, the Netherlands.

Troelstra, A.S., and D. van Dalen. 1988. *Constructivism in Mathematics, an Introduction (volume 1).* Studies in Logic and the Foundations of Mathematics, Vol. 121. North-Holland.

Urquhart, Alisdair. 1986. Many-valued logic. In *Handbook of Philosophical Logic*, ed. D. Gabbay and F. Guenthner. 71 – 116. Dordrecht (the Netherlands): D. Reidel Publishing Company.

van Benthem, Johan. 1996. *Exploring Logical Dynamics.* Studies in Logic, Language and Information. Stanford: CSLI Publications and FoLLI.

van Benthem, Johan, and Giovanna Cepparello. 1994. *Tarskian variations — Dynamic parameters in classical semantics.* Report CS-R9419. Amsterdam: Centrum voor Wiskunde en Informatica.

Veltman, Frank. 1996. Defaults in update semantics. *Journal of Philosophical Logic* 15:221 – 261.

Wirsing, Martin. 1990. Algebraic Specification. In *Handbook of Theoretical Computer Science*, ed. Jan van Leeuwen. Elsevier Science.

Appendix: definition of safe substitutions

Here we define the class of safe substitutions, announced in subsection 3.5. The idea is: a substitution is safe when it does not bring signature elements in the scope of actions that may modify them. For this purpose, we introduce the mappings $\mathsf{fvar}, \mathsf{sig}^+, \mathsf{change}$ and mod. $\mathsf{fvar}(\varphi)$ is the collection of free variables occurring in φ, idem for $\mathsf{fvar}(\alpha)$. $\mathsf{sig}^+(t)$ is the collection of extended signature elements (i.e. elements of $\Sigma \cup \{\mathsf{new}\}$) occurring in t. $\mathsf{change}(\alpha)$, an auxiliary mapping for the definition of mod, contains the

extended signature elements that are (possibly) changed by α. It is defined by

$$
\begin{array}{lcl}
\mathsf{change}(\mathsf{Create}) & = & \{\mathsf{new}\} \\
\mathsf{change}(f := \lambda x.t) & = & \{f\} \\
\mathsf{change}(p := \lambda x.\varphi) & = & \{p\} \\
\mathsf{change}(\varphi?) & = & \emptyset \\
\mathsf{change}(\alpha;\beta) & = & \mathsf{change}(\alpha) \cup \mathsf{change}(\beta) \\
\mathsf{change}(\alpha+\beta) & = & \mathsf{change}(\alpha) \cup \mathsf{change}(\beta) \\
\mathsf{change}(\alpha^*) & = & \mathsf{change}(\alpha) \\
\mathsf{change}(\cup x.\alpha) & = & \mathsf{change}(\alpha)
\end{array}
$$

$\mathsf{mod}(x,\varphi)$ is the collection of extended signature elements that may undergo modification at some free occurrence of x in φ; similar for $\mathsf{mod}(x,\alpha)$.

$$
\begin{array}{lcll}
\mathsf{mod}(x, s=t) & = & \emptyset & \\
\mathsf{mod}(x, p(t)) & = & \emptyset & \\
\mathsf{mod}(x, \neg\varphi) & = & \mathsf{mod}(x,\varphi) & \\
\mathsf{mod}(x, \varphi\wedge\psi) & = & \mathsf{mod}(x,\varphi) \cup \mathsf{mod}(x,\psi) & \\
\mathsf{mod}(x, \forall x\varphi) & = & \emptyset & \\
\mathsf{mod}(x, \forall y\varphi) & = & \mathsf{mod}(x,\varphi) & \text{if } y \not\equiv x \\
\mathsf{mod}(x, [\alpha]\varphi) & = & \mathsf{mod}(x,\alpha) \cup \mathsf{mod}(x,\varphi) \cup \mathsf{change}(\alpha) & \text{if } x \in \mathsf{fvar}(\varphi) \\
 & = & \mathsf{mod}(x,\alpha) \cup \mathsf{mod}(x,\varphi) & \text{if } x \notin \mathsf{fvar}(\varphi) \\
\mathsf{mod}(x, \mathsf{Create}) & = & \emptyset & \\
\mathsf{mod}(x, f := \lambda x.t) & = & \emptyset & \\
\mathsf{mod}(x, p := \lambda x.\varphi) & = & \emptyset & \\
\mathsf{mod}(x, \varphi?) & = & \mathsf{mod}(x,\varphi) & \\
\mathsf{mod}(x, \alpha;\beta) & = & \mathsf{mod}(x,\alpha) \cup \mathsf{mod}(x,\beta) \cup \mathsf{change}(\alpha) & \text{if } x \in \mathsf{fvar}(\beta) \\
 & = & \mathsf{mod}(x,\alpha) \cup \mathsf{mod}(x,\beta) & \text{if } x \notin \mathsf{fvar}(\beta) \\
\mathsf{mod}(x, \alpha+\beta) & = & \mathsf{mod}(x,\alpha) \cup \mathsf{mod}(x,\beta) & \\
\mathsf{mod}(x, \alpha^*) & = & \mathsf{mod}(x,\alpha) \cup \mathsf{change}(\alpha) & \text{if } x \in \mathsf{fvar}(\alpha) \\
 & = & \mathsf{mod}(x,\alpha) & \text{if } x \notin \mathsf{fvar}(\alpha) \\
\mathsf{mod}(x, \cup x.\alpha) & = & \emptyset & \\
\mathsf{mod}(x, \cup y.\alpha) & = & \mathsf{mod}(x,\alpha) & \text{if } y \not\equiv x
\end{array}
$$

Now we define *safe substitution* by

(t/x) is safe for every term;
(t/x) is safe for formula φ whenever $\mathsf{sig}^+(t) \cap \mathsf{mod}(x,\varphi) = \emptyset$;
(t/x) is safe for program α whenever $\mathsf{sig}^+(t) \cap \mathsf{mod}(x,\alpha) = \emptyset$.