Objects, Types and Modal Logics

Dan S. Andersen Lars H. Pedersen Hans Hüttel
Josva Kleist*

October 1998

Abstract

In this paper we present a modal logic for describing properties of terms in the object calculus of Abadi and Cardelli [AC96]. The logic is essentially the modal mu-calculus of [Koz83]. The fragment allows us to express the temporal modalities of the logic CTL [BAMP83]. We investigate the connection between the type system $\mathbf{Ob}_{1<:\mu}$ and the mu-calculus, providing a translation of types into modal formulae and an ordering on formulae that is sound w.r.t. to the subtype ordering of $\mathbf{Ob}_{1<:\mu}$.

1 Introduction

In [AC94b, AC94c, AC94a, AC96] Abadi and Cardelli present and investigate several versions of the ς-calculus, a calculus for describing central features of object-oriented programs, with particular emphasis on various type systems.

In this paper we present a modal logic for describing dynamic properties of terms in the object calculus. By dynamic properties we mean properties specifically related to the behavior over time of a term. We also examine the relation between the type system with recursive types $\mathbf{Ob}_{1<:\mu}$ for the ς-calculus and the modal mu-calculus [Koz83]. It turns out that there are close correspondences between the type system and a fragment of the mu-calculus using only maximal fixed-points. In particular, there is a sound and

*Address: Dep. of Computer Science, Aalborg University, Frederik Bajersvej 7, 9220 Aalborg, Denmark. Email: {hans,kleist}@cs.auc.dk

complete translation from types to logical formulae preserving typability and the subtype ordering, when we interpret subtyping as containment. Phrased differently, the translation establishes a Curry-Howard-style result in that it allows us to view ς-calculus types as realizers of certain mu-calculus formulae.

2 The ς-calculus and its reduction semantics

There are various versions of the ς-calculus. In this paper we shall consider what is essentially the simple object calculus of [AC94b] with booleans added. The set of object terms, **Obj**, is defined by the following abstract syntax:

$$
\begin{array}{rcll}
a & ::= & [l_i{=}\varsigma(x_i{:}A)b_i\ ^{i\in I}] & \text{objects} \\
 & \mid & x & \text{self variables} \\
 & \mid & a.l & \text{method activation} \\
 & \mid & a.l{\Leftarrow}\varsigma(x{:}A)b & \text{method override} \\
 & \mid & \mathsf{fold}(A,a) \mid \mathsf{unfold}(a) & \text{recursive fold/unfold} \\
 & \mid & \mathsf{if}(a,b_1,b_2) \mid \mathsf{true} \mid \mathsf{false} & \text{booleans}
\end{array}
$$

Here $x_i \in \mathbf{SVar}$ range over self variables, $l_i \in \mathbf{MNames}$ range over method names and $A_i \in \mathbf{Types}$. We let $\mathsf{m}(a)$ denote the set of method names and $\mathsf{fv}(a)$ the set of free self variables in a. A value, denoted by v, are either an object ($[l_i{=}\varsigma(x_i{:}B)b_i\ ^{i\in I}]$), a boolean value ($\mathsf{true}, \mathsf{false}$) or the fold of a value ($\mathsf{fold}(A,v)$).

The original presentation of the ς-calculus uses a small-step reduction semantics, which is also used in the labelled transition semantics in the following section.

Let $a = [l_i{=}\varsigma(x_i{:}A)b_i\ ^{i\in I}]$. Then the reduction rules are given by

$$
\begin{array}{ll}
a.l_k \ \rightsquigarrow\ b_k\{^a\!/\!_{x_k}\} & (k \in I) \\
a.l_k{\Leftarrow}\varsigma(x{:}A)b \ \rightsquigarrow\ [l_i = \varsigma(x_i{:}A_i)b_i,\ l_k = \varsigma(x{:}A)b]_{i\in I\setminus\{k\}} & (k \in I) \\
\mathsf{if}(\mathsf{true},b_1,b_2) \ \rightsquigarrow\ b_1 \qquad \mathsf{if}(\mathsf{false},b_1,b_2) \ \rightsquigarrow\ b_2 & \\
\mathsf{unfold}(\mathsf{fold}(A,v)) \ \rightsquigarrow\ v &
\end{array}
$$

The activation of the method l_k results in the method body being activated with the self variable being bound to the original object. Method override results in an object with the overridden method exchanged with the new method.

The reduction order is leftmost; this we express through evaluation contexts ($C[\cdot]$) which have the following abstract syntax (with $[\cdot]$ denoting the hole of the context):

$$C[\cdot] ::= [\cdot].l \mid [\cdot].l \Leftarrow \varsigma(x{:}A)b \mid \mathsf{unfold}([\cdot]) \mid \mathsf{fold}(A, [\cdot]) \mid \mathsf{if}([\cdot], a_1, a_2)$$

and an evaluation strategy given by the transition rule

$$\frac{a \rightsquigarrow b}{C[a] \rightsquigarrow C[b]}$$

We write $a \Downarrow v$ ("a converges to the value v") if there is a terminating reduction sequence $a \rightsquigarrow a_1 \rightsquigarrow \cdots v$. We write $a \Uparrow$ ("a diverges") if there is no v such that $a \Downarrow v$.

3 Types

One of the main motivation for the ς-calculus is that of studying various type systems of object-oriented programming languages within a unified framework. In this paper we shall consider the type system $\mathbf{Ob}_{1<:\mu}$ from [AC94b] as presented in [GR96].

3.1 The type language

The set of $\mathbf{Ob}_{1<:\mu}$ type expressions **Type** is defined via the following abstract syntax:

$$A ::= \mathsf{Bool} \mid [l_i{:}A_i \ ^{i \in I}] \mid \mathsf{Top} \mid \mu(X)A \mid X$$

Here Bool denotes the only ground type, namely that of truth values. $[l_i{:}A_i \ ^{i \in I}]$ denotes an object record type, where the method l_i has type A_i. Top denotes the most general or unspecified type, $\mu(X)A$ is a recursive type and X ranges over **TypeVar**, the set of type variables.

3.2 Assigning types to objects

$\mathbf{Ob}_{1<:\mu}$ has two kinds of judgments. *Type judgments* on the form $\Gamma \vdash a{:}A$, state that the object a has type A under the assumptions in Γ, where Γ describes typing assumptions of self variables. For instance, $\Gamma(x) = A$ states

$$\frac{X \neq Y}{X \succ Y} \qquad \frac{}{\mathsf{Top} \succ Y} \qquad \frac{}{[l_i{:}A_i\ ^{i \in I}] \succ Y} \qquad \frac{A \succ Y}{\mu(X)A \succ Y}$$

Table 1: Formal contractivity

that we assume that the free self variable x has type A. If Γ is empty we shall sometimes just write $a{:}A$ instead of $\emptyset \vdash a{:}A$.

The type system $\mathbf{Ob}_{1<:\mu}$ also incorporates a notion of *subtyping*, whose intended informal interpretation is that of capturing some types being more general than others. The expression $A <: B$ denotes that A is a subtype of B and thus that objects of type A may be used instead of objects of type B.

Subtyping judgments $\Gamma \vdash A <: B$ state that the type A is a subtype of B, given the subtyping assumptions in Γ. Here the typing assumptions in Γ describe subtyping constraints on type variables. $\Gamma(X) = A$ states that we assume $X <: A$.

In order to ensure uniqueness of recursively defined types Abadi and Cardelli define a syntactic predicate of *formal contractivity* on type variables. $A \succ Y$ should be read as 'variable Y is formally contractive in type expression A'. Informally, this means that any occurrence of Y occurs within the scope of a method label in the type expression A. The rules defining the predicate are shown in Table 1.

A type of $\mathbf{Ob}_{1<:\mu}$ is well-formed if it can be formed using the inference rules of Table 2.

(Type Object)	$\dfrac{\Gamma \vdash A_i \quad \forall i \in I}{\Gamma \vdash [l_i{:}A_i\ ^{i \in I}]}$	(Type Top)	$\dfrac{}{\Gamma \vdash \mathsf{Top}}$
(Type X)	$\dfrac{X \in \mathsf{dom}(\Gamma)}{\Gamma \vdash X}$	(Type Rec)	$\dfrac{\Gamma[X <: \mathsf{Top}] \vdash A \quad A \succ X}{\Gamma \vdash \mu(X)A}$

Table 2: Well-formed types

The subtyping relation is expressed in the inference rules of Table 3.

$$\text{(Sub Refl)}\ \frac{\Gamma \vdash A}{\Gamma \vdash A <: A} \qquad \text{(Sub Trans)}\ \frac{\Gamma \vdash A_1 <: A_2 \quad \Gamma \vdash A_2 <: A_3}{\Gamma \vdash A_1 <: A_3}$$

$$\text{(Sub } X\text{)}\ \frac{\Gamma(X) = A}{\Gamma \vdash X <: A} \qquad \text{(Sub Top)}\ \frac{\Gamma \vdash A}{\Gamma \vdash A <: \mathsf{Top}}$$

$$\text{(Sub Obj)}\ \frac{K \subseteq L \quad \Gamma \vdash A_i \quad \forall i \in L}{\Gamma \vdash [l_i{:}A_i\ ^{i \in <}] : [l_j{:}A_j\ ^{j \in K}]}$$

$$\text{(Sub Rec)}\ \frac{\Gamma \vdash \mu(X_1)A_1 \quad \Gamma \vdash \mu(X_2)A_2 \quad \Gamma[X_2 <: \mathsf{Top}, X_1 <: X_2] \vdash A_1 <: A_2}{\Gamma \vdash \mu(X_1)A_1 <: \mu(X_2)A_2}$$

Table 3: The subtyping relation

Finally, an object a has type A under assumptions Γ if $\Gamma \vdash a{:}A$ can be inferred by the type assignment rules in Table 4. An object term a is said to be *typable* if for some type A we can infer that $\emptyset \vdash a{:}A$.

4 A labelled transition semantics

The correspondence between object types and modal logic is based on the labelled transition semantics used to interpret logical formulae. As our labelled transition semantics, we shall use that proposed by Gordon and Rees in [GR96]. The syntax has been altered slightly in that boolean values can now appear. Transitions are on the form $a \xrightarrow{\alpha} b$ — this transition describes that the object term a admits the observation α and then becomes the object term b.

In the labelled transition semantics, terms are always annotated with their type. The types of $\mathbf{Ob}_{1<:\mu}$ are divided into two classes, *active* and *passive*. Active types are the types of values. Only Bool is active, so in our presentation all values are booleans. Recursive types, object types and Top are passive types. At active types a program must converge to a value before it can be observed; at passive types a program performs observable actions unconditionally, whether or not it converges.

The observations, $\alpha \in \mathbf{Obs}$, take the following forms:

$$\text{(Var)}\quad \frac{\Gamma(x) = A}{\Gamma \vdash x{:}A} \qquad \text{(Select)}\quad \frac{\Gamma \vdash a{:}[l_i{:}B_i\ ^{i\in I}] \quad j \in I}{\Gamma \vdash a.l_j{:}B_j}$$

$$\text{(Object)}\quad \frac{\Gamma[x_i{:}A] \vdash b_i{:}B_i \ \ \forall i \in I \qquad A \equiv [l_i{:}B_i\ ^{i\in I}]}{\Gamma \vdash [l_i{=}\varsigma(x_i{:}A)b_i\ ^{i\in I}] : A}$$

$$\text{(Update)}\quad \frac{\Gamma \vdash a{:}A \ \ \Gamma[x{:}A] \vdash b{:}B_j \ \ j \in I \qquad A \equiv [l_i{:}B_i\ ^{i\in I}]}{\Gamma \vdash a.l_j{\Leftarrow}\varsigma(x{:}A)b : A}$$

$$\text{(Fold)}\quad \frac{A \equiv \mu(X)B \ \ \Gamma \vdash a{:}B\{A/X\}}{\Gamma \vdash \mathsf{fold}(A, a) : A} \qquad \text{(Unfold)}\quad \frac{A \equiv \mu(X)B \ \ \Gamma \vdash a{:}A}{\Gamma \vdash \mathsf{unfold}(a) : B\{A/X\}}$$

$$\text{(If)}\quad \frac{\Gamma \vdash a{:}\mathsf{Bool} \ \ \Gamma \vdash a_1, a_2 : A}{\Gamma \vdash \mathsf{if}(a, a_1, a_2) : A} \qquad \text{(Bool)}\quad \frac{b \in \{\mathsf{true}, \mathsf{false}\}}{\Gamma \vdash b{:}\mathsf{Bool}}$$

$$\text{(Subsump)}\quad \frac{\Gamma \vdash a{:}A_1 \ \ \Gamma \vdash A_1 <: A_2}{\Gamma \vdash a : A_2}$$

Table 4: Type assignment

$$\alpha ::= \mathsf{true} \mid \mathsf{false} \mid l \mid l \Leftarrow \varsigma(x)b \mid \mathsf{unfold}$$

These observations should be interpreted as follows: An object term allows the observation true (resp. false) if the term is of type Bool and has the value true (resp. false). An object term allows the observation l if it has a method labelled l. An object term allows the observation $l \Leftarrow \varsigma(x)b$ if the object can have its method labelled l redefined as $\varsigma(x)b$. And finally, an object term always allows the observation unfold – intuitively, this corresponds to 'unfolding' the object by substituting self variables by their corresponding objects.

The labelled transition semantics is defined by the minimal family of relations

$$\{ \xrightarrow{\alpha} \mid \alpha \in \mathbf{Act}\}$$

closed under the rules in Table 5.

The notion of bisimulation equivalence $\sim$ is defined as usual [Par81, Mil89].

$$(\textsc{Trans Bool}) \quad \frac{a \Downarrow v \in \{\mathsf{true}, \mathsf{false}\}}{a_{\mathsf{Bool}} \xrightarrow{v} \mathbf{0}}$$

$$(\textsc{Trans Select}) \quad \frac{j \in I \quad A \equiv [l_i{:}B_i]_{i \in I}}{a_A \xrightarrow{l_j} a.l_j\ _{B_j}}$$

$$(\textsc{Trans Update}) \quad \frac{x{:}A \vdash b{:}B_j \quad j \in I \quad A \equiv [l_i{:}B_i]_{i \in I}}{a_A \xrightarrow{l_j \Leftarrow \varsigma(x)b} a.l_j \Leftarrow \varsigma(x{:}A)b_A}$$

$$(\textsc{Trans Unfold}) \quad \frac{A \equiv \mu(X)B \quad C \equiv B\{A/X\}}{a_A \xrightarrow{\mathsf{unfold}} \mathsf{unfold}(a)_C}$$

Table 5: The rules of the labelled transition semantics

Definition 1 (Bisimulation) *Bisimilarity $\sim$ is the greatest binary relation on ς-calculus terms that satisfies the following:*
$a \sim b$ if and only if

i. $a \xrightarrow{\alpha} a' \Rightarrow \exists\ b'$ *s.t.* $(b \xrightarrow{\alpha} b' \wedge a' \sim b')$

ii. $b \xrightarrow{\alpha} b' \Rightarrow \exists\ a'$ *s.t.* $(a \xrightarrow{\alpha} a' \wedge a' \sim b')$.

If $a \sim b$ we say that a and b are bisimilar.

We sometimes index bisimilarity w.r.t. types, writing $a \sim_A b$ if $a{:}A$ and $b{:}A$ and a and b are bisimilar. It should be noted, though, that Definition 1 not only relates terms a_A and b_A on their type A, but also on all of their super types. That is, if the subtype relation

$$A <: B_1 <: B_2 <: \cdots <: B_n$$

holds, then Definition 1 states that the following must hold:

$$a_A \sim b_A \quad \Rightarrow \quad a_{B_1} \sim b_{B_1}, \ldots, a_{B_n} \sim b_{B_n}$$

In the following we refer to the bisimulation of Definition 1 as Gordon-Rees bisimulation or bisimulation in the sense of Gordon and Rees.

5 A logic for objects

In the following we shall consider the modal mu-calculus [Koz83] interpreted over the labelled transition system defined in the previous section. Later we shall show that one of its sublogics corresponds to the $\mathbf{Ob}_{1<:\mu}$type system in a precise sense.

The modal μ-calculus was introduced by Kozen in [Koz83]. It corresponds to the logic introduced by Hennessy and Milner and [HM85] extended with (local) recursive definitions [Lar90].

5.1 Syntax and informal semantics

The set of μ-calculus formulae **Form** is given by

$$\begin{array}{rcl} F & ::= & F_1 \vee F_2 \mid F_1 \wedge F_2 \mid \langle\alpha\rangle F \mid [\alpha]F \mid X \mid \nu X.F \mid \mu X.F \\ & \mid & \mathit{tt} \mid \mathit{ff} \\ \alpha & ::= & l \mid \mathsf{unfold} \mid l \Leftarrow \varsigma(x)b \end{array}$$

Here X ranges over the set of formula recursion variables **FormVar**. tt and ff are atomic formulae, not to be confused with the boolean values of the ς-calculus. The modalities of the logic are indexed by the observations from the labelled transition semantics. Intuitively, a formula $\langle\alpha\rangle F$ is true for an object a if a allows *some* observation α such that F is true for the resulting object. Similarly, a formula $[\alpha]F$ is true for the object a if *all* observations of α will result in an object for which F is true.

5.2 Semantics of the μ-calculus

We interpret our logic over the labelled transition system of the previous section. If a formula F is true for an object a, we say that *a satisfies* F and write that $a \models F$.

The denotation of a formula F is the set of object terms that satisfy F. As formulae may contain free variables, the denotation of a formula is seen relative to an environment $\sigma : \mathbf{FormVar} \hookrightarrow \mathcal{P}(\mathbf{Obj})$ which for a given variable, returns the set of objects which satisfies the formula bound to that variable.

We can extend operations and predicates on sets of objects to environments. For any two environments σ_1, σ_2 we write $\sigma_1 \subseteq \sigma_2$ iff for all variables we have that $\sigma_1(X) \subseteq \sigma_2(X)$. If S is a set of objects, we write $\sigma \subseteq S$ if for all X we have that $\sigma(X) \subseteq S$. Similarly, $\sigma_1 \cup \sigma_2$ is the environment σ such that $\sigma(X) = \sigma_1(X) \cup \sigma_2(X)$.

The semantics of formula not using the recursion connectives can then be defined as:

$$\begin{array}{rcl}
[\![tt]\!]\sigma & = & \mathbf{Obj} \\
[\![f\!f]\!]\sigma & = & \emptyset \\
[\![F_1 \vee F_2]\!]\sigma & = & [\![F_1]\!]\sigma \cup [\![F_2]\!]\sigma \\
[\![F_1 \wedge F_2]\!]\sigma & = & [\![F_1]\!]\sigma \cap [\![F_2]\!]\sigma \\
[\![X]\!]\sigma & = & \sigma(X) \\
[\![\langle\alpha\rangle F]\!]\sigma & = & \{a \mid \exists b \text{ s.t. } a \xrightarrow{\alpha} b \text{ and } b \in [\![F]\!]\sigma\} \\
[\![[\alpha]F]\!]\sigma & = & \{a \mid \forall b \text{ with } a \xrightarrow{\alpha} b : b \in [\![F]\!]\sigma\}
\end{array}$$

The operators $\nu X.F$ and $\mu X.F$ are local recursion operators. For any recursive formula $\nu X.F$ or $\mu X.F$ we define a *declaration function*

$$\mathcal{D}_F : (\mathbf{FormVar} \hookrightarrow \mathcal{P}(\mathbf{Obj})) \rightarrow \mathcal{P}(\mathbf{Obj}) \text{ by } \mathcal{D}_F(\sigma) = [\![F]\!]\sigma$$

Both recursion operators denote a solution to the equation $X = F$, that is, we want an environment σ such that $[\![X]\!]\sigma = [\![F]\!]\sigma$. A σ with this property is called a *model*.

One may easily show that $(\mathbf{FormVar} \hookrightarrow \mathcal{P}(\mathbf{Obj}), \subseteq)$, the set of environments ordered under inclusion, constitutes a complete lattice and that the function $\mathcal{D}_F$ is a monotonic function for any recursive formula $\nu X.F$ or $\mu X.F$. Consequently, Tarski's fixed point theorem [Tar55] for complete lattices and monotonic functions, guarantees that models always exist for any recursive formula.

Theorem 2 (Maximal and minimal model) *Given a recursive formula F of the form $\nu X.F$ or $\mu X.F$, there exist models σ_{max} and σ_{min} given by:*

$$\begin{array}{rcl}
\sigma_{max} & = & \bigcup\{\sigma \mid \sigma \subseteq \mathcal{D}_F(\sigma)\} \\
\sigma_{min} & = & \bigcap\{\sigma \mid \mathcal{D}_F(\sigma) \subseteq \sigma\}
\end{array}$$

σ_{max} is the maximal model w.r.t. $\subseteq$ and σ_{min} is the minimal model w.r.t. $\subseteq$.

The ν-connective is taken to indicate that we want the model σ_{max}, whereas the μ-connective is taken to indicate that we want the model σ_{min} and thus the semantics of the recursion operators is

$$\begin{aligned}
[\![\nu X.F]\!]\sigma &= \sigma_{max}(X) \\
[\![\mu X.F]\!]\sigma &= \sigma_{min}(X)
\end{aligned}$$

From Theorem 2 we obtain the following definition of pre- and post-models.

Definition 3 (Pre- and post-models) *Given a formula $\mu X.F$ or $\nu X.F$, an environment σ is a* pre-model *if $\sigma \in \{\sigma \mid \mathcal{D}_F(\sigma) \subseteq \sigma\}$. σ is a* post-model *if $\sigma \in \{\sigma \mid \sigma \subseteq \mathcal{D}_F(\sigma)\}$.*

Consequently, Theorem 2 says that the semantics of a ν-formula is the union of all its post-models and that the semantics of a μ-formula is the intersection of all its pre-models.

This logic fully characterizes Gordon-Rees bisimulation; that is two objects are bisimilar if and only if they satisfy the same logical formulae.

Theorem 4 (Characterization) *Assume that $a, b \in \mathbf{Obj}$ and let σ be either a pre- or post-model. Then,*

$$\begin{array}{ll}
 & a \sim b \\
\Updownarrow & \\
 & \forall F \in \mathbf{Form} : a \in [\![F]\!]\sigma \Leftrightarrow b \in [\![F]\!]\sigma
\end{array}$$

The proof of Theorem 4, which is lengthy but standard, is omitted; it can be found in [AP96].

6 Specifying objects

The modal mu-calculus is very powerful when used a temporal logic of labelled transition systems. It is well-known that the temporal modalities of the propositional branching time temporal logic CTL [Eme94] are expressible within the mu-calculus (in fact, it can be shown that all of CTL* [Eme94] can be expressed within the mu-calculus).

In this short section we shall describe how properties of ς-calculus can be described this way. We let the set **Act** denote the set of possible observations; we write $[\mathbf{Act}]F$ as an abbreviation of $\bigwedge_{\alpha\in\mathbf{Act}}[\alpha]F$ and similarly, we write $\langle\mathbf{Act}\rangle F$ as an abbreviation of $\bigvee_{\alpha\in\mathbf{Act}}\langle\alpha\rangle F$.

The CTL temporal modalities can be defined as

$$
\begin{aligned}
\mathsf{AG}F &= \nu X.(F \wedge [\mathbf{Act}]X) \\
\mathsf{EG}F &= \nu X.(F \wedge ([\mathbf{Act}]\mathit{ff} \vee \langle\mathbf{Act}\rangle X)) \\
\mathsf{EF}F &= \mu X.(F \wedge \langle\mathbf{Act}\rangle X) \\
\mathsf{AF}F &= \mu X.(F \wedge (\langle\mathbf{Act}\rangle\mathit{tt} \wedge [\mathbf{Act}]X)) \\
\mathsf{U}^s(F,G) &= \mu X.(G \vee (F \wedge \langle\mathbf{Act}\rangle\mathit{tt} \wedge [\mathbf{Act}]X)) \\
\mathsf{U}^w(F,G) &= \nu X.(G \vee (F \wedge [\mathbf{Act}]X))
\end{aligned}
$$

The intuitive interpretation of these modalities is

$\mathsf{AG}F$: An object a satisfies $\mathsf{AG}F$, if F is satisfied by all states of any transition path of a.

$\mathsf{EG}F$: $\mathsf{EG}F$ is satisfied by an object a if there exists some transition path of a such that F is satisfied by all states of the path.

$\mathsf{EF}F$: The dual of $\mathsf{AG}F$. An object a satisfies $\mathsf{EF}F$ if F is satisfied by some state along some transition path.

$\mathsf{AF}F$: Guarantees that F will sooner or later be true along any transition path.

$\mathsf{U}^s(F,G), \mathsf{U}^w(F,G)$: The meanings of $\mathsf{U}^s(F,G)$ and $\mathsf{U}^w(F,G)$ are almost identical. The idea is that an object a satisfies $\mathsf{U}^s(F,G)$ or $\mathsf{U}^w(F,G)$ if F is satisfied by a until G at some point is satisfied. The difference is that $\mathsf{U}^w(F,G)$ does not guarantee that G will ever be satisfied. $\mathsf{U}^s(F,G)$ is called *strong until*, and $\mathsf{U}^w(F,G)$ *weak until*.

Notice that invariance properties are expressed using maximal recursion, whereas eventuality properties are expressed using minimal recursion.

The following example illustrates how an object can be specified using the CTL modalities:

Example 1 In [AC96] a ***calculator*** object is described. The behavior of the calculator can informally be stated as follows: Either one of the methods *enter*, *add* or *sub* can be invoked, which will result in a new calculator, or the method *equals* can be invoked resulting in termination. In the following let **Act** be defined as $\mathbf{Act} = \{enter, add, sub\}$.

The formula F shown below specifies the observations that must be allowed by a ***calculator*** object of recursive type. The formula G describes what must always hold about the methods *enter*, *add* and *sub* of the ***calculator*** object:

$$\begin{aligned} F &= \langle \mathsf{unfold} \rangle tt \wedge [\mathsf{unfold}] tt \wedge [\mathbf{Act}] ff, \\ G &= ([add] tt \wedge \langle add \rangle tt) \vee ([sub] tt \wedge \langle sub \rangle tt) \\ &\quad \vee ([enter] tt \wedge \langle enter \rangle tt) \wedge [\mathsf{unfold}] ff. \end{aligned}$$

To express that it holds that F is satisfied until G at some point will be satisfied we use the connective "strong until" and write:

$$\mathsf{U}^s(F, G).$$

We write that $\mathsf{U}^s(F, G)$ must always hold as

$$\mathsf{AGU}^s(F, G)$$

The specification of the ***calculator*** object is a combination of $\mathsf{AGU}^s(F, G)$ and the specification for the *equals* method:

$$\begin{aligned} F_{eq} &= \langle \mathsf{unfold} \rangle ([equals] tt \wedge \langle equals \rangle tt) \wedge \\ &\quad [\mathsf{unfold}]([equals] tt \wedge \langle equals \rangle tt) \end{aligned}$$

The final correctness specification looks as follows:

$$\mathsf{AGU}^s(F, G) \vee F_{eq}.$$

□

7 Types as logical formulae

In this section we shall show an intimate correspondence between the types of $\mathbf{Ob}_{1<:\mu}$ and formulae of a subset of our modal logic, in that we interpret $\mathbf{Ob}_{1<:\mu}$types as formulae. Inhabiting a type then corresponds to satisfying the corresponding formula in modal logic, and subtyping corresponds to inclusion of modal properties.

7.1 The ν-calculus and its semantics

From the point of view of temporal logic, types are certain invariance properties. When relating types and logical properties, we are therefore interested in *maximal* models. The sublogic that we shall consider therefore only has the ν-connective.

$$\begin{array}{rcl} F & ::= & F_1 \wedge F_2 \ \mid \langle \alpha \rangle F \mid [\alpha] F \mid X \mid \nu X.F \\ & \mid & \langle \mathsf{true} \rangle tt \mid \langle \mathsf{false} \rangle tt \mid tt \mid ff \\ \alpha & ::= & l \ \mid \mathsf{unfold} \mid l \Leftarrow \varsigma(x) b \end{array}$$

In order to be able to express the typings of objects in our modal logic we have introduced the atomic predicate isBool, which is the logical formula corresponding to Bool. The predicate isBool is satisfied by an object o if o has type Bool.

The reason for introducing this predicate is that the mu-calculus connectives alone cannot express that an object of type Bool will allow precisely the observations true or false – in order to achieve this, we would need infinite conjunctions. Further, the fact that an object of type Bool can diverge in the reduction semantics cannot be expressed. In other words, it seems reasonable that the base types should be modelled by atomic formulae.

Moreover, we now only consider closed formulae as the recursion operator on types is interpreted using the ν-connective. In what follows, this allows us to formulate and employ an alternative definition of the semantics, namely a coinductive definition in terms of satisfaction relation.

Definition 5 *A* satisfaction relation *is a relation $\mathcal{S} \subseteq$ **Obj**$\times$**Form** for which it holds that*

$$\begin{array}{rll} a\mathcal{S}tt & & \textit{for any } a \\ a\mathcal{S}F_1 \vee F_2 & \textit{implies} & a\mathcal{S}F_1 \textit{ and } a\mathcal{S}F_2 \\ a\mathcal{S}F_1 \vee F_2 & \textit{implies} & a\mathcal{S}F_1 \textit{ or } a\mathcal{S}F_2 \\ a\mathcal{S}\langle\alpha\rangle F & \textit{implies} & \exists a' \textit{ s.t. } a \xrightarrow{\alpha}{}' a' \textit{ and } a'\mathcal{S}F \\ a\mathcal{S}[\alpha] F & \textit{implies} & \forall a' \textit{ s.t. } a \xrightarrow{\alpha} a' \textit{ it holds that} a'\mathcal{S}F \\ a\mathcal{S}\nu X.F & \textit{implies} & a\mathcal{S}F\{\nu X.F/X\} \end{array}$$

We now have the following result.

Proposition 6 *If F is a closed ν-calculus formula, then $a \models F$ iff there exists a satisfaction relation $\mathcal{S}$ such that $a\mathcal{S}F$.*

PROOF. The *only if*-part of the proposition follows from the fact that $\models$ is a satisfaction relation. For the *if*-part, one proceeds by the induction in the structure of F. The only interesting case is that of recursive formulae $\nu X.F_1$. Here notice that if we let $S_X = \{a \mid a\mathcal{S}\nu X.F_1\}$, then $S_X \subseteq [\![F]\!]_{[X \mapsto S_X]}$; in other words, the environment $[X \mapsto S_X]$ is a post-model. $\square$

According to the above proposition, if we wish to show that $a \models F$, then it is enough to determine a satisfaction relation $\mathcal{S}$ for which $a\mathcal{S}F$.

7.2 An interpretation of types in the ν-calculus

The ν-calculus is, in fact, too large – only certain formulae can occurs as the translations of $\mathbf{Ob}_{1<:\mu}$ types. We shall call such formulae *type formulae*. The abstract syntax of type formulae is given by the grammar

$$\begin{array}{rcl} F & ::= & tt \mid \Big(\bigwedge_{i\in I}\langle l_i\rangle F_i\Big) \wedge \Big(\bigwedge_{i\in I}[l_i]F_i\Big) \mid \mathsf{isBool} \\ & \mid & X \mid \nu X.(\langle \mathsf{unfold}\rangle F \wedge [\mathsf{unfold}]F) \end{array}$$

We denote the set of type formulae by $\mathbf{Form}_t$.

We are now able to introduce the translation $\mathcal{T} : \mathbf{Type} \to \mathbf{Form}_t$ from types to type formulae as follows:

$$\begin{array}{rcl} \mathcal{T}(\mathsf{Top}) & = & tt \\ \mathcal{T}(\mathsf{Bool}) & = & \mathsf{isBool} \\ \mathcal{T}([l_i{:}A_i]_{i\in I}) & = & \Big(\bigwedge_{i\in I}\langle l_i\rangle \mathcal{T}(A_i)\Big) \wedge \Big(\bigwedge_{i\in I}[l_i]\mathcal{T}(A_i)\Big) \\ \mathcal{T}(\mu(X)A) & = & \nu X.(\langle \mathsf{unfold}\rangle \mathcal{T}(A) \wedge [\mathsf{unfold}]\mathcal{T}(A)) \\ \mathcal{T}(X) & = & X \end{array}$$

Not surprisingly, the type Top is assigned the formulae tt, since all typable objects have type (or supertype) Top. The type Bool is as a special case translated to the atomic formula isBool. The mu-calculus translation of an object type reflects the possible method invocations that can be performed with respect to objects of the object type. The specification for a recursive object type is an invariance property. It states that it must *always* be possible to perform an unfold-transition leading to the specification for an object type. As recursive types are interpreted as invariants, we have to use the maximal fixed point operator. In other words, the μ becomes a ν when passing from types to formulae.

The translation defined by $\mathcal{T}$ is similar to the notion of *characteristic formula* [IS94] for the typings of objects. It is not the case, though, that these characteristic formulae express all possible behaviors of objects. In particular, it is not possible to prove that the logic for object types fully characterizes the bisimulation of Gordon and Rees, as the types and thus the type formulae say nothing about the possibility of method overrides. (This is exemplified towards the end of this section.)

7.3 Soundness and completeness of the translation

The translation presented here is correct in a very precise sense. In order to express this, we need to formulate the logic counterparts of the typability notions. Definition 7 expresses the notion of subtyping with respect to the modal formulae for types.

Definition 7 *Let $A, B \in$ **Type** and Γ some well formed environment. We say that Γ models the subtyping relation $A <: B$ written*

$$\Gamma \models A <: B$$

if for some post-model σ it holds that $[\![\mathcal{T}(A)]\!]\sigma \subseteq [\![\mathcal{T}(B)]\!]\sigma$ and $\sigma(X) \subseteq [\![\mathcal{T}(C)]\!]\sigma$ for all $(X <: C) \in \Gamma$.

Theorem 8 relates the standard subtype judgments, which are of the form $\Gamma \vdash A <: B$, to the inclusion relation between logical properties.

Theorem 8 *Let $A, B \in$ **Type**, Γ some well formed environment and $\Gamma \vdash A <: B$. Then $\Gamma \models A <: B$.*

Proof. By induction in the proof tree of $\Gamma \vdash A <: B$, that is, by inspecting the rules for subtyping.

Sub Refl: It follows directly that $[\![\mathcal{T}(A)]\!]\sigma \subseteq [\![\mathcal{T}(A)]\!]\sigma$.

Sub Top: Since $[\![\mathcal{T}(\mathsf{Top})]\!]\sigma = \mathbf{Obj}$ it must hold that $[\![\mathcal{T}(A)]\!]\sigma \subseteq [\![\mathcal{T}(\mathsf{Top})]\!]\sigma$ for all types $A \in$ **Type**.

Sub Object: Let $[l_i{:}A_i]_{i\in I} <: [l_i{:}A_i]_{i\in J}$ for some indexing sets I, J for which it holds that $J \subseteq I$. By definition of $\mathcal{T}$, $\mathcal{T}([l_i{:}A_i]_{i\in I})$ must impose at least the same restrictions on objects as $\mathcal{T}([l_i{:}A_i]_{i\in J})$. It follows that $\mathcal{T}[\![[l_i{:}A_i]_{i\in I}]\!]\sigma \subseteq \mathcal{T}[\![[l_i{:}A_i]_{i\in J}]\!]\sigma$.

Sub X: Follows from the well formedness of Γ.

Sub Rec: Assume there is a post-model σ such that $\sigma(X) \subseteq [\![\mathcal{T}(C)]\!]\sigma$ for all $(X <: C) \in \Gamma$, $\sigma(X_1) \subseteq \sigma(X_2)$ and $[\![\mathcal{T}(A)]\!]\sigma \subseteq [\![\mathcal{T}(B)]\!]\sigma$. We must find a post-model σ' such that $\sigma'(X) \subseteq [\![\mathcal{T}(C)]\!]\sigma'$ for all $(X <: C) \in \Gamma$ and $\mathcal{T}[\![\mu(X_1)A]\!]\sigma' \subseteq \mathcal{T}[\![\mu(X_2)B]\!]\sigma'$. Take σ' defined by

$$\sigma'(X) = \begin{cases} \sigma(X_1) \cup [\![\mathcal{T}(A)]\!]\sigma & \text{if } X = X_1 \\ \sigma(X_2) \cup [\![\mathcal{T}(B)]\!]\sigma & \text{if } X = X_2 \\ \sigma(X) & \text{otherwise} \end{cases}$$

By the assumptions $\mathcal{T}[\![\mu(X_1)A]\!]\sigma' \subseteq \mathcal{T}[\![\mu(X_2)B]\!]\sigma'$. Thus, we only need to check that σ' is a post-model, i.e. that

$$\sigma'(X_1) \subseteq [\![\mathcal{T}(A)]\!]\sigma' \text{ and } \sigma'(X_2) \subseteq [\![\mathcal{T}(B)]\!]\sigma'$$

By definition of σ', $\sigma(X) \subseteq \sigma'(X)$ for all X. Then, since $[\![\cdot]\!]$ is monotonic on σ it must hold that $[\![F]\!]\sigma \subseteq [\![F]\!]\sigma'$ for all $F \in \mathbf{Form}_t$.

□

We now show that our translation is sound and complete with respect to the usual typing of objects. In particular we would like to prove a statement resembling

$$\Gamma \vdash a{:}A \quad \Leftrightarrow \quad a \in [\![\mathcal{T}(A)]\!]\sigma$$

for some Γ and σ. It turns out that this is possible for closed objects, that is for typings with an empty type environment.

Theorem 9 states us the soundness of our translation.

Theorem 9 (Soundness) *Let $a \in$ **Obj** and $A \in$ **Type**. If $\emptyset \vdash a{:}A$ then $a \models \mathcal{T}[\![A]\!]$.*

PROOF. The relation

$$\begin{array}{rcl} \mathcal{S}_{\mathcal{T}} & = & \{(a, \mathcal{T}(A) \mid \vdash a{:}A\} \\ & \cup & \{(a, \langle l_i \rangle \mathcal{T}(A_i)) \mid \vdash a{:}[l_i{:}A_i]_{i \in I}\} \\ & \cup & \{(a, [l_i] \mathcal{T}(A_i)) \mid \vdash a{:}[l_i{:}A_i]_{i \in I}\} \\ & \cup & \{(a, \langle \mathsf{unfold} \rangle \mathcal{T}(B\{\mu(X)B/X\}) \wedge [\mathsf{unfold}] \mathcal{T}(B\{\mu(X)B/X\})) \mid \vdash a{:}\mu(X)B\} \\ & \cup & \{(a, \langle \mathsf{unfold} \rangle \mathcal{T}(B\{\mu(X)B/X\})) \mid \vdash a{:}\mu(X)B\} \end{array}$$

is a satisfaction relation.

□

Completeness here means that if a typable object in $\mathbf{Ob}_{1<:\mu}$satisfies a type formula, then it also has the corresponding type:

Theorem 10 (Completeness) *Let $a \in \mathbf{Obj}$ and $F \in \mathbf{Form}_t$. If $a \in [\![F]\!]\sigma$ then there exists a type A such that $\emptyset \vdash a{:}A$, where $\mathcal{T}(A) = F$.*

PROOF. By induction in the structure of F.

Top: If $F = tt$ then it must hold that $a{:}\mathsf{Top}$, where $\mathcal{T}(\mathsf{Top}) = tt$.

Bool: If $F = \mathsf{isBool}$ then, obviously, $a{:}\mathsf{Bool}$, where $\mathcal{T}(\mathsf{Bool}) = \mathsf{isBool}$.

Object: Assume that

$$F = \Big(\bigwedge_{i \in I} \langle l_i \rangle \mathcal{T}(A_i) \Big) \wedge \Big(\bigwedge_{i \in I} [l_i] \mathcal{T}(A_i) \Big)$$

It must be the case that a can perform the transitions $a \xrightarrow{l_i} a.l_i$, where $a.l_i \in [\![\mathcal{T}(A_i)]\!]\sigma$, for all $i \in I$. That is, a must at least have the type $[l_i{:}A_i]_{i \in I}$. We do not know if a may have further possibilities for transitions, $a \xrightarrow{l_j} a.l_j$ for $j \in J$, $I \cap J = \emptyset$, which results in the typing

$$[l_i{:}A_i, l_j{:}A_j]_{i \in I, j \in J}$$

but we do not care about this, since

$$[l_i{:}A_i, l_j{:}A_j]_{i \in I, j \in J} <: [l_i{:}A_i]_{i \in I}$$

Unfold: Assume that

$$F = \nu X.(\langle\mathsf{unfold}\rangle\mathcal{T}(A) \wedge [\mathsf{unfold}]\mathcal{T}(A))$$

Then, obviously, $a{:}\mu(X)A$.

Var: Trivial.

□

As mentioned earlier, the characteristic formulae for object types do not fully characterize the bisimulation equivalence of objects due to Gordon and Rees. This is due to the fact that the typing of an object does not take into account the possible method overrides that can be performed on the object. In fact it is possible, with respect to an object, to do infinitely many method overrides. That is, the ς-calculus is not finite branching. This lack of information about the possible behavior of an object carries through to the type formulae.

Example 2 Consider the two objects:

$$\begin{array}{rcl} a & = & [l_1 = \mathsf{true}, l_2 = \mathsf{true}] \\ b & = & [l_1 = \mathsf{true}, l_2 = \varsigma(x{:}A)\ x.l_1] \end{array}$$

Both have the type

$$A \ = \ [l_1{:}\mathsf{Bool}, l_2{:}\mathsf{Bool}],$$

and supertypes B and C:

$$\begin{array}{rcl} B & = & [l_1{:}\mathsf{Bool}], \\ C & = & [l_2{:}\mathsf{Bool}] \end{array}$$

It can be shown that $a \sim_B b$ and $a \sim_C b$ but $a \not\sim_A b$, since after the method overrides $a.l_1 \Leftarrow \varsigma(x)x.l_2$ and $b.l_1 \Leftarrow \varsigma(x)x.l_2$, $a.l_2$ converges while $b.l_2$ diverges.

The encodings of A,B and C look as follows

$$\begin{array}{rcl} \mathcal{T}(A) & = & ((\langle l_1\rangle\mathcal{T}(\mathsf{Bool}) \wedge \langle l_2\rangle\mathcal{T}(\mathsf{Bool})) \wedge ([l_1]\mathcal{T}(\mathsf{Bool}) \wedge [l_2]\mathcal{T}(\mathsf{Bool})) \\ & = & ((\langle l_1\rangle\mathsf{isBool} \wedge \langle l_2\rangle\mathsf{isBool}) \wedge ([l_1]\mathsf{isBool} \wedge [l_2]\mathsf{isBool}) \\ \mathcal{T}(B) & = & \langle l_1\rangle\mathcal{T}(\mathsf{Bool}) \wedge [l_1]\mathcal{T}(\mathsf{Bool}) \\ & = & \langle l_1\rangle\mathsf{isBool} \wedge [l_1]\mathsf{isBool} \\ \mathcal{T}(C) & = & \langle l_2\rangle\mathcal{T}(\mathsf{Bool}) \wedge [l_2]\mathcal{T}(\mathsf{Bool}) \\ & = & \langle l_2\rangle\mathsf{isBool} \wedge [l_2]\mathsf{isBool} \end{array}$$

If the type formulae should characterize Gordon-Rees bisimulation, then it should be the case that $a, b \in [\![\mathcal{T}(B)]\!]\sigma$ and $a, b \in [\![\mathcal{T}(C)]\!]\sigma$, but $a, b \notin [\![\mathcal{T}(A)]\!]\sigma$. Obviously this is not the case, since $a, b \in [\![\mathcal{T}(A)]\!]\sigma$. □

8 Conclusions and further work

In this paper we have described how certain properties of ς-calculus terms can be described within the modal mu-calculus. A natural next step is to investigate how one can use the mu-calculus to verify interesting properties of objects. The notion of model checking, that is, algorithmically checking whether a term satisfies a given modal formula [SW89], is already well understood in the context of process calculi. It remains to be seen how far we can proceed within the ς-calculus.

We have also shown a correspondence between the type system $\mathbf{Ob}_{1<:\mu}$ for the ς-calculus and the modal mu-calculus, which captures both type assignment and subtyping.

A natural question is: How much further we can go in this direction ?s we can easily deal with arrow types, if we introduce a simple notion of intuitionistic implication into our logic. Let us say that F implies G for some object a if a is an object abstraction which, whenever given an argument satisfying the property F becomes an object satisfying the property G.

$$[\![F \Rightarrow G]\!]\sigma = \{a \mid \forall b \in [\![F]\!]\sigma : ab \in [\![G]\!]\sigma\}$$

We then get

$$\mathcal{T}(A \to B) = \mathcal{T}(A) \Rightarrow \mathcal{T}(B)$$

It is straightforward to see that this extended interpretation is sound w.r.t. the subtyping rules for arrow types, and in particular for the rule

$$\text{(SUB ARROW)} \quad \frac{\Gamma \vdash A' <: A \quad \Gamma \vdash B <: B'}{\Gamma \vdash A \to B <: A' \to B'}$$

which states that the arrow type is contravariant in its first argument and covariant in its second argument.

This leads straight to the question of the possibility of dealing with the *variance annotations* suggested by Abadi and Cardelli in [AC96]. This seems to require the introduction of negation into the syntax of the logic and is a topic of further study.

A related question is whether one can capture the polymorphic type systems of the ς-calculusproposed in [AC96] within a higher-order modal predicate logic.

Finally, the interpretation of types as modal formulae suggests a somewhat alternative account of the semantics of types to that presented in [AC96], where the semantics of a type is a per (partial equivalence relation). This is a topic of ongoing work by one of the authors.

References

[AC94a] Martín Abadi and Luca Cardelli. A semantics of object types. In *Proceedings of the 9th IEEE Symposium on Logics in Computer Science*, pages 332–341. IEEE Computer Society Press, 1994.

[AC94b] Martín Abadi and Luca Cardelli. A theory of primitive objects – untyped and first-order systems. In *Theoretical Aspects of Computer Software*, pages 296–320. Springer-Verlag, 1994.

[AC94c] Martín Abadi and Luca Cardelli. A theory of primitive objects: Second-order systems. In *Proceedings of European Symposium on Programming*, number 788 in Lecture Notes in Computer Science, pages 1–25. Springer-Verlag, 1994.

[AC96] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer Verlag, 1996.

[ACV96] M. Abadi, L. Cardelli, and R. Viswanathan. An interpretation of Objects and Objects Types. In *Proc. 23th POPL*. ACM Press, 1996.

[AP96] Dan S. Andersen and Lars H. Pedersen. An operational approach to the ς-calculus. Master's thesis, Aalborg University, September 1996. Available as technical report R-96-2034.

[BAMP83] M. Ben-Ari, Z. Manna, and A. Pnueli. The temporal logic of branching time. *Acta Informatica*, 20:207–226, 1983.

[Eme94] E.A. Emerson. *Handbook of Theoretical Computer Science*, chapter Temporal and Modal Logic, pages 995–1072. Elsevier, 1994.

[GR96] Andrew D. Gordon and Gareth D. Rees. Bisimilarity for a first-order calculus of objects with subtyping. In *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Programming Languages*, 1996.

[HM85] M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *Journal of the ACM*, 32(1):137–161, January 1985.

[IS94] Anna Ingólfsdóttir and Bernhard Steffen. Characteristic formulae for processes with divergence. *Information and Computation*, 110:149–163, April 1994.

[Koz83] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 1983.

[Lar90] Kim G. Larsen. Proof systems for satisfiability in hennessy-milner logic with recursion. *Theoretical Computer Science*, 72:265–288, 1990.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[Par81] David Park. Concurrency and automata on infinite sequences. In P. Deussen, editor, *Confernce on Theoretical Computer Science*, number 104 in Lecture Notes in Computer Science, pages 167–183. Springer-Verlag, 1981.

[SW89] C. Stirling and D. Walker. Local model checking in the modal mu-calculus. In *LNCS 351*, pages 369–383. Springer-Verlag, 1989.

[Tar55] A. Tarski. A lattice-theoretical fixpoint and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.