Observational mu-calculus

Julian Bradfield* and Perdita Stevens **

Laboratory for Foundations of Computer Science
University of Edinburgh
JCMB, King's Buildings
Mayfield Road
EDINBURGH EH9 3JZ
Scotland

1 The problem

The modal mu-calculus is widely considered to be a good 'assembly language' into which temporal logics can be compiled. However, the mu-calculus is not good at expressing properties of systems where the observations are structured in some way. The principal examples are real-timed systems, in which the passing of time can be observed, and value-passing systems, in which the system may be observed to input and output values along named ports. The values may even be names themselves, as in the pi-calculus. A large number of extensions of popular logics has been proposed (for example, in [2, 1, 4]), but there is as yet no common framework in which the extensions can be studied. This seems unfortunate, since in fact the extensions have a great deal in common.

In this paper we consider the problem of defining an 'assembly-language' logic for such extensions. The logic should be small and simple, and it should be possible to translate these previously studied extensions into it. This requirement will almost certainly lead to a logic in which typical properties are expressed as long formulae. This will not concern us. It is unreasonable to expect model-checking in so powerful a logic to be decidable in general; we will settle for a framework in which it is possible to identify decidable fragments sufficient to include the images of decidable high level logics. Here we describe steps in this direction, and our reasons for optimism about the strategy.

There are several possible frameworks in which one might look for a solution. The most powerful framework is full second-order logic; however, this is intractable, in many ways. Monadic second-order logic is a restriction which has a much more amenable theory; it is also used in at least one serious verification environment[3]. It can be argued that second-order quantification is too hard to understand, even for an assembly-language logic. It is also arguable that since the popular temporal logics are all expressible in terms of fixpoints, it is unnecessary to go beyond fixpoints to second order, even monadic. This would suggest the use of first-order logic with fixpoints, a logic much studied in finite model theory, though less so in the mainstream verification community. However, we maintain

* jcb@dcs.ed.ac.uk, supported by an EPSRC Advanced Research Fellowship
** Perdita.Stevens@dcs.ed.ac.uk, supported by EPSRC GR/K68547

that all these logics have one feature, which is not shared by traditional temporal logics, and which we consider undesirable: they all have variables ranging over 'states', so that a formula can capture states and keep them for later inspection. Temporal logics, including the modal mu-calculus, do not have state variables; although the semantics is defined over states, or even runs, there is no explicit access to states in the logic. This accords with the observational paradigm, in which one can inspect the behaviour of a process, but not its internal state. We therefore adopt a modal mu-calculus framework for our logic. However, when observing a value-passing or real-time process, values, which may be arbitrary datatypes, or times are part of the observation. It is therefore reasonable – and necessary to capture existing logics! – to allow our logic to have variables ranging over observable values, and to allow some logical and non-logical manipulation of these observed values. To obtain decidability results, one may need to restrict such manipulation severely; however, in general we propose the use of first-order logic, with a set of defined predicates, for the data language.

2 An assembly language mu calculus, $A\mu$

A formula is allowed to observe transitions, including values, names or times which may be part of the action. It may store these values for later use. Accordingly we allow a formula to use a set $\mathcal{C}$ of mutable cells $c, d, \ldots$. The values of these cells may change when the formula tracks a change in the process by observing a transition, or autonomously. We need to be able to state constraints on the contents of the cells. Accordingly we have a two-level logic, the higher level parametrized on the lower. Formulae $\phi \ldots$ have free variables which are cell-names $c, d \ldots$ or hooked cell names $\overleftarrow{c}, \overleftarrow{d} \ldots$; this allows us to state constraints on how cell values change, for example at a modality. (Logically, the cells can be expressed as first-order variables which are also passed through the fixpoints as parameters; the cell notation saves symbols, and imposes certain constraints on the use of these variables, as does the use of the VDM-style hooks.)

The high level logic is defined thus:

$$\Phi = \mathbf{T} \mid \mathbf{F} \mid X \mid \Phi \vee \Phi \mid \Phi \wedge \Phi \mid \langle l, C, \phi \rangle \Phi \mid [l, C, \phi]\, \Phi \mid \nu X.\, \Phi \mid \mu X.\, \Phi$$

where l is an *action expression*, $C \subseteq \mathcal{C}$ a set of cells whose contents may be altered on passing through the modality, and ϕ a low level formula over $\mathcal{C} \cup \overleftarrow{C}$ which must be satisfied by the cell contents (and ex-contents of modifiable cells) after the modality.

Action expressions depend on the domain of interpretation. For example, suppose that we interpret the logic over labelled transition systems where the labels L include $a(v)$ or $\overline{a}(v)$ where $v \in V$ is a value. (Such a transition system arises naturally from early semantics of a value-passing CCS process.) Then an action expression may be any label $l \in L$, or ϵ (a dummy label such that $P \xrightarrow{\epsilon} P$, allowing autonomous setting of cells), or $a(c)$ or $\overline{a}(c)$ for a cell name c. In the last case the purpose is to set c: we have P satisfying $\langle a(c), C, \phi \rangle \Phi$ iff

$P \xrightarrow{a(v)} P'$ for some v and there exist new values of the cells C such that ϕ holds and $(c = v)$ holds, and P' satisfies Φ (with respect to the updated cell values). Note that if $c \notin C$, we are requiring the process to read exactly the current value of c. Formally, we can say that an action expression maps a cell environment to a label, or to the dummy label ϵ: given a cell environment, an action expression yields the label we want to observe from the process, and specifing a range of possible cell environments by allowing mutable cells correspondingly allows a range of labels.

If we are concerned with pi calculus processes, we may want also to allow an action expression to be $c(d)$ where c (as well as d) may be a cellname; again, the process will do a transition with a particular name and the result will be to put that name into the cell c.

For another example, we can interpret the logic over real timed processes, modelled as labelled transition systems with instantaneous action labels $l \in L$, and delay actions $\delta(d)$ for non-negative reals d which are always possible from any state. We can then incorporate the 'specification clocks' of [2] simply by having a real-valued cells $c_1, c_2, \ldots$, and requiring that in every delay modality, C includes the c_i and the predicate ϕ enforces their updating:

$$\langle \delta(d), \{d, c_1, ..., c_n\}, c_1 = \overleftarrow{c_1} + d \wedge ... \wedge c_n = \overleftarrow{c_n} + d \rangle.$$

As syntactic sugar we may adopt the convention that cells marked $^{\mathrm{cl}}$ behave in this manner, and omit them from the delay modalities. Note that these are specification clocks. Since we are taking a rigorously observational view here, internal state of a process, such as propositions or clocks, is not observable unless the process chooses to export the information; by the usual hacks, any internal state can be exported.

Figure 1 gives the semantics in full. Note that a fixpoint is implicitly parametrized on the cells, or more exactly on a cell environment: a cell environment is a (finite) map from cell names to values. That is, satisfaction is defined relative to

- a variable environment V which maps each fixpoint variable X to a map from cell environments to sets of processes,
- a cell environment ρ.

The logic is defined relative to

- the low level logic (the language from which ϕ is drawn) with a corresponding notion of satisfaction.
- matching of actions to action expressions: an action expression and an action may or may not *match* in the context of a cell environment; formally, they match iff the action label is the image of the cell environment under the map given by the action expression.

In the usual way, the subset ordering on $\mathcal{P}(Proc)$ is extended pointwise to the set of maps from cell environments to $\mathcal{P}(Proc)$, giving a complete lattice structure so that the Knaster-Tarski theorem applies. That is, fixpoints in the observational mu-calculus are fixpoints of functions of type

$$((Var \to Val) \to \mathcal{P}(Proc)) \to ((Var \to Val) \to \mathcal{P}(Proc))$$

where the first argument gives the current values of all the (finitely many) cells which exist (wlog) anywhere in the formula.

Thus to see whether $A \models_{V,\rho} \nu X.\Phi$, we let h be the maximal fixpoint of the function

$$F : ((Var \to Val) \to \mathcal{P}(Proc)) \to ((Var \to Val) \to \mathcal{P}(Proc))$$

which is defined by

$$(Ff)(\rho') = \{B \in Proc : B \models_{V[X \mapsto f]\rho'} \Phi\}$$

Then we define $A \models_{V,\rho} \nu X.\Phi$ iff $A \in h(\rho)$.

Fig. 1. Semantics of $A\mu$

$A \models_{V,\rho} \mathbf{T}$
$A \not\models_{V,\rho} \mathbf{F}$
$A \models_{V,\rho} X$ iff $A \in V(X)(\rho)$
$A \models_{V,\rho} \Phi_1 \vee \Phi_2$ iff $A \models_{V,\rho} \Phi_1$ or $A \models_{V,\rho} \Phi_2$
$A \models_{V,\rho} \Phi \wedge \Phi$ iff $A \models_{V,\rho} \Phi_1$ and $A \models_{V,\rho} \Phi_2$
$A \models_{V,\rho} \langle l, C, \phi\rangle\Phi$ iff $\exists\rho'$ a new cell environment differing from ρ only in the values of cells in C ($c \notin C \Rightarrow \rho'(c) = \rho(c)$) and $\exists a, A'$ such that $A \xrightarrow{a} A'$ and a matches l in context ρ' and $\rho, \rho' \models \phi$ and $A' \models_{V,\rho'} \Phi$
$A \models_{V,\rho} [l, C, \phi]\Phi$ iff whenever ρ' is a new cell environment differing from ρ only in the values of cells in C ($c \notin C \Rightarrow \rho'(c) = \rho(c)$) and $A \xrightarrow{a} A'$ and a matches l in context ρ' and $\rho, \rho' \models \phi$ then $A' \models_{V,\rho'} \Phi$
$A \models_{V,\rho} \nu X.\Phi$ iff $A \in S(\rho)$ for some $S \in (Var \to Val) \to \mathcal{P}(Proc)$ such that $A' \in S(\rho)$ implies $A' \models_{V[X \mapsto S],\rho} \Phi$
$A \models_{V,\rho} \mu X.\Phi$ iff $A \in S(\rho)$ for all $S \in (Var \to Val) \to \mathcal{P}(Proc)$ such that $A' \models_{V[X \mapsto S],\rho} \Phi$ implies $A' \in S(\rho)$.

3 Use of $A\mu$

To illustrate our logic, we exhibit translations from existing logics specialised for time or for value passing.

3.1 TCTL.

Timed CTL, in the flavour of [2], is interpreted on systems which have a discrete state and a number of real-time clocks; a system either does an instantaneous action, which may include resetting clocks, or allows time to pass. The atomic predicates are state predicates, or simple comparison of clocks – a restriction which allows model-checking procedures – and the temporal connectives are $\exists\mathcal{U}$ and $\forall\mathcal{U}$. The underlying semantic model is systems of 'real-time trajectories' along which time passes or states change: 'premodels' satisfy basic sanity properties (including stutter closure), 'safe premodels' are closed under limits, and 'real-time systems' have only divergent trajectories (along which time passes; in particular, zeno paths are excluded). To get the normally desired interpretation of inevitability $\forall\mathcal{U}$, one interprets over real-time systems. In this case the 'obvious' translation of $p \,\forall\mathcal{U}\, q$ is just $\mu Z.q \vee ((p \vee q) \wedge []Z$. However, we are working with transition systems as the underlying model, so *a priori* we must have non-divergent paths, and thus the obvious translation is actually translating from safe premodels, not real-time systems. There are two options here: in the tradition of Fair CTL, one could simply decree that non-divergent paths are unfair, and adjust the model-checking procedures to ignore them. However, as we have a powerful mu-calculus, we can encode this fairness constraint, at least over reasonably well-behaved systems.

A divergence-safe real-time system (in the sense of [2]) comprises a set $P = \{p_1, \ldots, p_m\}$ of boolean propositions and a set $C = \{x_1, \ldots, x_n\}$ of real-time clocks, together with a set of trajectories satisfying the appropriate sanity conditions. A state of the system is a boolean valuation of the propositions and a non-negative real valuation of the clocks. Let S be the set of states. It is shown in [2] that the behaviour of such a system can be generated by a transition relation $\rightarrow$ on $S \times S$, which is required to be reflexive, such that each transition changes the propositions and/or resets clocks. In fact, the system can be executed by alternately taking a transition and allowing time to pass.

We translate such a system into our framework in the obvious way: we take the transition system $S, \rightarrow$ and add all delay transitions $s \xrightarrow{\delta(d)} s$ to model the passing of time. Since TCTL permits free access to the propositions and clocks of the system, we extend the state-changing transitions to carry labels exporting the current values of propositions and clocks at the end state: $s \xrightarrow{(p_1,\ldots,p_m,x_1^{\mathrm{cl}},\ldots,x_n^{\mathrm{cl}})} s'$. For obvious typographic reasons, we shall elide these labels in print. Similarly, whenever we write an $A\mu$ modality $\langle -, E, \phi \rangle$, we silently assume that it is really $\langle -(p_1, \ldots, p_m, x_1^{\mathrm{cl}}, \ldots, x_n^{\mathrm{cl}}), E \cup F, \phi \wedge \phi' \rangle$, where F is the set of proposition and clock cells, and ϕ' states that every clock is either reset or unchanged: $\bigwedge_i (x_i^{\mathrm{cl}} = 0 \vee x_i^{\mathrm{cl}} = \overleftarrow{x}_i)$.

Subject to a proviso to be explained, we can now translate a TCTL formula ϕ on the original system into a formula $\mathrm{tr}(\phi)$ of observational mu-calculus on the translated system so that the translated property holds iff the original property holds.

We assume that we have $A\mu$ cells corresponding to the propositions and clocks. Further, for every *specification clock* z of TCTL we have a cell z^{cl} (using the syntactic sugar described earlier).

The basic propositions of TCTL have the form p, for $p \in P$; $x + c \leq y + d$ for $x, y \in C$ and $c, d \in \mathbb{N}$. These are imported directly into $A\mu$ as part of the 'low-level logic'.

The formula $z.\phi$, which is true if ϕ is true after resetting the specification clock z to zero, is translated to $\langle \epsilon, \{z^{\mathrm{cl}}\}, z^{\mathrm{cl}} = 0 \rangle \mathrm{tr}\, \phi$, using the dummy transition.

The existential 'until' formula $\phi \exists\mathcal{U} \psi$ is simply translated to $\mu Z.\, \mathrm{tr}\, \psi \vee (\mathrm{tr}\, \phi \wedge (\langle -, \{\}, \mathbf{T} \rangle Z \vee \langle \delta(d), \{d\}, \mathbf{T} \rangle Z))$. As this is an existential statement, we need not concern ourselves with the existence of zeno paths: a trajectory demonstrating the truth of $\phi \exists\mathcal{U} \psi$ also demonstrates the truth of the translation, and vice versa.

As we discussed earlier, the universal 'until' formula $\mathrm{tr}(\phi \forall\mathcal{U} \psi)$ is much harder, as our models do not exclude zeno paths. Provided that our system satisfies the condition (*), which will be defined after the proof, the following complicated formula serves as a translation:

$$
\begin{aligned}
\mathrm{tr}(\phi \,\forall\mathcal{U}\, \psi) \equiv [\epsilon, b^{\mathrm{cl}}, b^{\mathrm{cl}} = 0] \mu X. \nu Z. \mathrm{tr}\, \psi \vee \\
&(\mathrm{tr}\, \phi \\
&\wedge (b^{\mathrm{cl}} > 1 \Rightarrow [\epsilon, b^{\mathrm{cl}}, b^{\mathrm{cl}} = 0] X) \\
&\wedge [-, \emptyset, \mathbf{T}] Z \\
&\wedge (\langle \epsilon, \{d\}, \mathbf{T} \rangle (\langle \delta(d), \emptyset, \mathbf{T} \rangle \mathrm{tr}\, \psi \wedge [\delta(d'), \{d'\}, d' \leq d] Z) \\
&\quad \vee [\delta(d''), \{d''\}, \mathbf{T}] Z))
\end{aligned}
$$

Let us show the correctness of this translation. A state s in the original system satisfies $\phi \,\forall\mathcal{U}\, \psi$, by the semantics of [2], if every (divergent) trajectory $\sigma = s_0 \xrightarrow{\delta(d_0)} s_1 \ldots$ satisfies $\phi \mathcal{U} \psi$, that is there is some s_i and some $d \leq d_i$ such that $s_i + d \models \psi$, and for all s_j, d' such that $j < i$ and $d' \leq d_j$ or $j = i$ and $d' \leq d$, $s_j + d' \models \phi$. Firstly, suppose that $\mathrm{tr}(\phi \forall\mathcal{U} \psi)$ fails at some state t_0 in the transition system model. We shall construct a divergent real-time trajectory in the original real-time system from t_0 on which $\phi \,\mathcal{U}\, \psi$ fails. The trajectory has the property that X, and therefore $\mathrm{tr}\, \psi$, fails at all points; and it is composed of sections, such that each section ends in a state in which either $\mathrm{tr}\, \phi$ fails, in which case $\phi \,\mathcal{U}\, \psi$ has failed, or at least one time unit has passed since the beginning of the section. Suppose the current section constructed so far is $s_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_n} s_n$, where each α_i is a transition or a delay (consecutive delays may be combined into one by the sanity conditions on real-time systems). s_n fails Z, so it fails $\mathrm{tr}\, \psi$ and fails at least one of the four conjuncts in the second half of the body of Z. If it fails $\mathrm{tr}\, \phi$, we are done, since then $\phi \,\mathcal{U}\, \psi$ fails on this finite trajectory. If it fails the second conjunct, then at least one unit of time has passed since the last time, so we set b^{cl} to 0 and start a new section. If s_n fails $[-, \emptyset, \mathbf{T}] Z$, then extend the current section by $s_n \to s_{n-1}$, where s_{n-1} fails Z. The slighly tricky case is when s_n fails the last conjunct. In this case, there exists some d'' such that $s_n + d''$ fails Z. In fact, we can make (using the property (*)) the stronger statement that also $\mathrm{tr}\, \psi$ fails between s_n and $s + d''$, for the following reason: suppose ψ holds at some $d < d''$. Then since s_n fails $\langle \epsilon, \{d\}, \mathbf{T} \rangle (\langle \delta(d), \emptyset, \mathbf{T} \rangle \mathrm{tr}(\psi) \wedge [\delta(d'), \{d'\}, d' \leq d] Z)$, there must be some $d' < d$ at which Z fails. Repeat ad infinitum to obtain an infinite number of points between s_n and $s_n + d''$ at which ψ holds, interspersed with points at which Z fails; then by property (*) applied to ψ, there is a finite interval in which ψ holds; but over this interval Z holds, which is a contradiction. Now extend the current section by $s_n \xrightarrow{\delta(d'')} s_n + d''$.

Each time we extend the current section, we unfold the failing maximal fix-point Z; hence after a finite number of steps, the current section must end, either in failure of ϕ, or in time passing.

Thus if ϕ never fails, we have an infinite divergent trajectory along which ψ always fails, so showing that $\phi \, \forall\mathcal{U} \, \psi$ fails.

The converse is easy; any trajectory demonstrating the failure of $\phi \forall\mathcal{U} \psi$ also demonstrates the failure of $\mathrm{tr}(\phi \, \forall\mathcal{U} \, \psi)$.

The property (*) required for the proof is: if ψ holds at infinitely many points in some finite interval, then ψ holds over some non-empty open subinterval. If TCTL is finitely variable over a class of systems, this property will hold over that class. Further, no physical system can fail (*). For an example of a system that does fail (*), consider a system with an initial state s_0, and such that a transition to s_1 can occur at $s_0 + 1/2^i$ for all i.

3.2 Timed mu-calculus

In a similar style, the timed mu-calculus $T\mu$ of [2] with its binary 'until' operator $\phi \triangleright \psi$ can be translated. In [2], $T\mu$ is defined over arbitrary premodels; as previously, we restrict our attention to those premodels that are generated by transition systems. We shall use the same notation as above for the system.

The basic predicates of $T\mu$ are the same as for $TCTL$, including the use of specification clocks, and are handled identically.

Disjunction and negation are translated as themselves.

$T\mu$ also has the specification clock resetting operator $z.\phi$, which is translated as above.

The temporal operator of $T\mu$ is the 'leads to' operator $\triangleright$. The semantics of this operator is defined by: $s \models_{\mathcal{E}} \phi \triangleright \psi$, where $\mathcal{E}$ is a *clock environment* giving the values of the clocks of the formula, if there is a state s' and a delay d such that $s \xrightarrow{\delta(d)} s \rightarrow s'$ and $s' \models_{\mathcal{E}+d} \psi$, and for all delays $d' \leq d$, $s + d' \models_{\mathcal{E}+d'} \phi \vee \psi$, where $\mathcal{E} + d$ is $\mathcal{E}$ with all clocks advanced by d.

This semantic definition can be expressed directly in $A\mu$, using the trick of the dummy transition to express quantification, thus:

$$\langle \epsilon, \{d\}, \mathbf{T} \rangle (\langle \delta(d), \emptyset, \mathbf{T} \rangle \langle L, \emptyset, \mathbf{T} \rangle q \wedge [\delta(d'), \{d'\}, d' \leq d] \, (p \vee q)).$$

Finally, the $T\mu$ fix-point, which is formally parametrized on clock environments, is translated directly to an $A\mu$ fix-point, formally parametrized on cell environments.

3.3 Value-passing mu calculus

As sketched earlier, we can also handle value-passing logics such as [1, 4]. As an example we translate Dam's first order mu calculus for the pi calculus, which we will call $P\mu$. The main difficulty, which is presentational rather than substantial, stems from Dam's explicit use of abstractions and concretions, both in the pi calculus and in the logic. An abstraction is an agent which requires instantiation of names to become ground; a concretion is an agent which provides names as well as a ground continuation. We give Dam's version of the pi calculus an LTS semantics defined in terms of his commit relation, by using as actions of the LTS both actions of the pi-calculus (names, co-names and τ) and the special pseudo-actions (n) and $[n]$ for n a name, representing the process being instantiated with a name n and emitting a name n, respectively:

- $A \xrightarrow{a} B$ iff for some C, $A \succ a.C$ and $C \succ B$.
- $A \xrightarrow{(n)} B$ iff for some C, $A \succ (\lambda x.C)n$ and $C\{n/x\} \succ B$.
- $A \xrightarrow{[n]} B$ iff either $A \succ [n]B$, or $A \succ (\nu n)[n]B$.

The $a(b).P$ of the usual pi calculus notation corresponds to $a.\lambda b.P$ in Dam's, and accordingly the $\xrightarrow{a(b)}$ transition of the usual semantics is split into two parts $\xrightarrow{a}\xrightarrow{(b)}$; and similarly $\overline{a}\langle b\rangle.P$ corresponds to $a.[b]P$.

In fact we are using a slight variant: Dam relativises the pi-calculus semantics to name partitions, but that is not necessary for our purposes.

In $P\mu$, λ is used for all binding, so the quantifiers $\exists$, $\forall$, Σ do not bind formula variables. Notionally, they accept a name and add it to the pending stack (the list of names that appears in the semantics of the logic). When the λ is unwound, it pops a name from the stack and uses it to instantiate a name in the body of the λ expression. The well formedness conditions ensure that it is possible to tell from the formula for each λ which constructor was responsible for the name being put on the stack. Similarly an application ϕx may be seen as a formula which puts its argument x onto the stack, whence it will be retrieved by some λ in ϕ. To simulate this, we may assign cells $s_1, \ldots, s_k$ to stack positions (k is bounded by the depth of constructor nesting), and a cell x for each variable x, and then simulate binding by assignment, thus: translate $\lambda x.\phi$ as $[(x), \{x\}, x = s_i]$, where s_i is the cell into which the matching constructor has placed its value. Note that here 'constructor' means $\exists, \forall, \Sigma$, which push one value on the stack; and application $\phi x_1 \ldots x_n$, which may push several values on the stack, and which we might prefer to write in reverse Polish notation as $x_n \ldots x_1 \phi$. This stack mechanism ensures that fixpoint unfolding is correctly handled: if a fixpoint X of arity 2, say, appears in a formula as Xxy, then y and x are 'pushed on to the stack' by the application, and then when the fixpoint is unfolded, its body finds the arguments 'on the stack'; this copes with the fact that in $P\mu$ parameters are explicitly passed through fixpoints by application , whereas in $A\mu$ they are implicitly passed through the cell environment. Accordingly, a formula that has n parameters at the top level uses $s_1, \ldots, s_n$ to receive them, and we may view the top level as a pseudo-constructor.

The semantics of $P\mu$ are defined using a name partition ϵ, which specifies which names are distinct. Since the only predicates on names which this logic allows are (in)equalities, this is sufficient in that context. Our logic stores actual values in named cells, so we have more information available and no need for the ϵ.

To build an $A\mu$ translation of a given $P\mu$ formula, we first decorate the constructors with the number of the stack cell into which it puts its value (let us write $+i$ to indicate a push), and decorate each λ with the number of its matching constructor ($-i$ to indicate a pop).

For example, consider the formula

$$\nu X.\lambda x.[in]\forall(\lambda y.Xy) \wedge [\overline{out}]\Sigma(\lambda x'.x = x' \wedge Xx').$$

This is an example of a parametrized fixpoint, as it requires a name to be instantiated. It is, in fact, the specification of the one place memory cell: 'for the initial contents x, if we *in*put a name y, we go to a state with y as the contents, and if we $\overline{out}$put a value x', then x' is equal to x and we return to the initial state (since $x' = x$!)'.

The decorated version is

$$^{+1}\nu X.\lambda^{-1}x.[in]\forall^{+1}(\lambda^{-1}y.(Xy)^{+1}) \wedge [\overline{out}]\Sigma^{+1}(\lambda^{-1}x'.x = x' \wedge (Xx')^{+1}),$$

which needs only one stack cell. Then we translate the decorated term according to Figure 2.

Fig. 2. Translating the decorated value-passing mu calculus

ϕ in Dam's logic	$\mathrm{tr}\,\phi$ in $A\mu$
$x = y$	$\langle\epsilon, \emptyset, x = y\rangle T$
$x \neq y$	$\langle\epsilon, \emptyset, x \neq y\rangle T$
$\phi \wedge \psi$	$\mathrm{tr}\,\phi \wedge \mathrm{tr}\,\psi$
$\phi \vee \psi$	$\mathrm{tr}\,\phi \vee \mathrm{tr}\,\psi$
$[a]\,\phi$	$[a, \emptyset, T]\,\mathrm{tr}\,\phi$
$\langle a\rangle\phi$	$\langle a, \emptyset, T\rangle \mathrm{tr}\,\phi$
X	X
$\nu X.\phi$	$\nu X.\mathrm{tr}\,\phi$
$\mu X.\phi$	$\mu X.\mathrm{tr}\,\phi$
$\lambda^{-i}x.\phi$	$[(x), \{x\}, x = s_i]\,\mathrm{tr}\,\phi$
$(\phi x)^{+i}$	$\langle\epsilon, \{s_i\}, s_i = x\rangle \mathrm{tr}\,\phi$
$\Sigma^{+i}\phi$	$\langle[s_i], \{s_i\}, T\rangle \mathrm{tr}\,\phi$
$\forall^{+i}\phi$	$[\epsilon, \{s_i\}, T]\,\mathrm{tr}\,\phi$
$\exists^{+i}\phi$	$\langle\epsilon, \{s_i\}, T\rangle \mathrm{tr}\,\phi$

To prove the translation correct, we need to demonstrate that

$$A \models_{V, x_1, \ldots x_n, \epsilon} \phi_L \qquad \text{iff} \qquad \mathrm{tr}\,(A, \epsilon) \models_{V, \rho(x_1, \ldots x_n, \epsilon)} \mathrm{tr}\,\phi$$

where $\rho(x_1, \ldots x_n, \epsilon)$ is any cell environment which gives values to the cells x_i in a way which respects ϵ.

This is proved by a routine, albeit somewhat intricate, induction on the structure of ϕ.

4 Games

The remaining question then is, can we treat in our logic the problems that can be treated in the original logics—in particular, the model-checking problem—both with the generality given by our framework, and in specific domains with the effectiveness of the domain logics.

It is easy to see that a minor variant of the standard model-checking game [7] characterises satisfaction of an $A\mu$ formula by a process. We need only alter the modality rules to allow the player who chooses the process transition to choose new values for the modifiable cells too, subject to satisfying the predicate on cell values, and to correct matching of an action expression to an observation.

In detail:

We wish to establish whether a process P satisfies a closed Φ of the observational mu-calculus in the presence of some initial cell environment ρ. We assume all bound fixpoint variables in Φ are distinct, renaming them if necessary to ensure this: the assumption is used in Rule 6 of Figure 4.

The *model-checking game* $\mathcal{G}(P, \Phi, \rho)$, is played by Abelard and Eloise. Abelard attempts to show that P fails to have the property Φ in environment ρ whereas Eloise tries to show that P does have Φ there. We write Player A and Player B for "a player" and "the other player" when it doesn't matter which is which.

A *play* of $\mathcal{G}(P_0, \Phi_0, \rho_0)$ is a finite or infinite length sequence of the form $(P_0, \Phi_0, \rho_0) \dots (P_n, \Phi_n, \rho_n) \dots$ where each Φ_i is a subformula of Φ_0 and each P_i is a derivative of P_0 and each ρ_i is a cell environment on the cells which appear in Φ_0. (We call such a triple a *configuration* of the game.)

Suppose a play (so far) is $(P_0, \Phi_0, \rho_0) \dots (P_j, \Phi_j, \rho_n)$. The moves are given in Figure 4: note that the form of the available moves, and which player chooses, are determined by the form of Φ_j. Each time the current game configuration is

1. if $\Phi_j = \Psi_1 \wedge \Psi_2$ then Abelard chooses Φ_{j+1} to be either Ψ_1 or Ψ_2 and $P_{j+1} = P_j$ and $\rho_{j+1} = \rho_j$.
2. if $\Phi_j = \Psi_1 \vee \Psi_2$ then Eloise chooses Φ_{j+1} to be either Ψ_1 or Ψ_2, and P_{j+1} is P_j and $\rho_{j+1} = \rho_j$.
3. if $\Phi_j = [l, C, \phi]\Psi$ then Abelard chooses a new cell environment ρ_{j+1} differing from ρ_j only in the values of cells in C ($c \notin C \Rightarrow \rho_{j+1}(c) = \rho_j(c)$) and a transition $P_j \xrightarrow{a} P_{j+1}$ where a matches l in context ρ_{j+1} and $\rho_j \rho_{j+1} \models \phi$. Φ_{j+1} is Ψ.
4. if $\Phi_j = \langle l, C, \phi \rangle \Psi$ then Eloise chooses a new cell environment ρ_{j+1} differing from ρ_j only in the values of cells in C ($c \notin C \Rightarrow \rho_{j+1}(c) = \rho_j(c)$) and a transition $P_j \xrightarrow{a} P_{j+1}$ where a matches l in context ρ_{j+1} and $\rho_j \rho_{j+1} \models \phi$. Φ_{j+1} is Ψ.
5. if $\Phi_j = \sigma Z.\Psi$ then Φ_{j+1} is Z and P_{j+1} is P_j and $\rho_{j+1} = \rho_j$.
6. if $\Phi_j = Z$ and Z is bound by $\sigma Z.\Psi$ then Φ_{j+1} is Ψ and P_{j+1} is P_j and $\rho_{j+1} = \rho_j$.

Fig. 3. Rules for the next move in a game play

$(P, \sigma Z.\Psi)$, at the next step this fixed point is abbreviated to Z, and each time the configuration is (Q, Z) the fixed point subformula it identifies is, in effect, unfolded once as the formula becomes Ψ.[1]

The conditions for winning a play are given in Figure 4. Abelard wins if a blatantly false configuration is reached, or if Eloise is stuck, and dually for Eloise. The remaining condition identifies who wins an infinite length play. We call variables bound by ν Eloise-variables and variables bound by μ Abelard-variables, and the notion of subsuming is:

Definition 1. *Suppose $\sigma X.\Psi$ and $\sigma Y.\Psi'$ are subformulae of a formula Φ. X subsumes Y if $\sigma Y.\Psi'$ is a subformula of $\sigma X.\Psi$.*

We omit the easy proof that X in winning condition 3 is indeed unique, so that the condition is well-defined.

Abelard wins

1. The play is $(P_0, \Phi_0) \ldots (P_n, \Phi_n)$ and $\Phi_n = \mathbf{F}$.
2. The play is $(P_0, \Phi_0) \ldots (P_n, \Phi_n)$ and $\Phi_n = \langle K \rangle \Psi$ and $\{Q : P \xrightarrow{a} Q \text{ and } a \in K\} = \emptyset$.
3. The play $(P_0, \Phi_0) \ldots (P_n, \Phi_n) \ldots$ has infinite length and the unique variable X which occurs infinitely often and which subsumes all other variables that occur infinitely often identifies a least fixed point subformula $\mu X.\,\Psi$.

Eloise wins

1. The play is $(P_0, \Phi_0) \ldots (P_n, \Phi_n)$ and $\Phi_n = \mathbf{T}$.
2. The play is $(P_0, \Phi_0) \ldots (P_n, \Phi_n)$ and $\Phi_n = [K]\Psi$ and $\{Q : P \xrightarrow{a} Q \text{ and } a \in K\} = \emptyset$.
3. The play $(P_0, \Phi_0) \ldots (P_n, \Phi_n) \ldots$ has infinite length and the unique variable X which occurs infinitely often and which subsumes all other variables that occur infinitely often identifies a greatest fixed point subformula $\nu X.\,\Psi$.

Fig. 4. Winning conditions

A *strategy* π for Player A is a set of rules telling Player A how to move: that is, it is a partial function from plays[2] to configurations, which given a play $p \in \mathrm{dom}\ \pi$ ending in a configuration (Q, Ψ) from which Player A must move, returns a non-empty set of legal next configurations. If every such set is a singleton, we say that π is *deterministic* (and in this case we will usually think of $\pi(p)$ as a configuration, rather than as a singleton set of configurations). We call π *history-free* if $\pi(p)$ is determined solely by the final configuration (Q, Ψ) of p, irrespective of the rest of the play. A play q *follows* π if for every proper

[1] As there are no choices here it doesn't matter who "chooses" – to fit in with the abstract game framework, we say that Eloise chooses to unwind minimal fixpoints and Abelard chooses to unwind maximal fixpoints.

[2] Plays are just sequences of moves, which need not yet be decided.

prefix p of q ending in an A-choice, $p \in \text{dom } \pi$ and the next configuration of q after p is in $\pi(p)$. π is *complete* if whenever p is a play following π and ending in a configuration from which Player A must choose, $\pi(p)$ is defined. Otherwise it is partial. π is a winning strategy if it is complete and B does not win any play which follows π. A history-free complete strategy may be regarded as a partial function from configurations to configurations.

The basic theorem we exploit is:

Theorem 1. *$P \models_{\emptyset,\rho} \Phi$ iff Eloise has a winning strategy for $\mathcal{G}(P, \Phi)$.*

This is a trivial variation on the corresponding theorem for plain mu-calculus and its games, see [7]: the crucial point is that the winning conditions for infinite plays ensure that the semantics of minimal and maximal fixpoints are reflected in the game.

4.1 Abstract games for model-checking the observational mu calculus

Whether it is possible to calculate a winning strategy – that is, to solve a model-checking problem – depends on the domain of interpretation and the lower level logic. [5] suggests a generic approach via *abstract games*, in which classes of game positions are considered together and split only when the analysis requires it. Initially positions are considered together if they have the same *shape*; the notion of shape – an equivalence relation $\approx$ of finite index on the set of reachable concrete positions – is required to satisfy certain sanity conditions, such as that the shape of a position is sufficient to determine whose turn it is to move, and that the sequence of shapes of an infinite play is enough to determine who wins it. These requirements are not hard to meet: for example, in the case of model-checking the observational mu calculus, we could take the shape of (A, ϕ, ρ) to be ϕ. In practice, we would be more likely to take a finer notion of shape which also considered the agent A; we need to be able to define the sets of concrete positions which arise in the execution of the algorithm of [5] by giving their common shape together with a *constraint* that describes what subset of the $\approx$-equivalence class we have. Thus there is a trade-off: the more precise the notion of shape, the less powerful the constraint language has to be.

The algorithm gradually refines the equivalence relation $\approx$, and is guaranteed to terminate if, essentially, there is a finite refinement of the original equivalence relation which is stable under the operations of the algorithm. In [6] such refinements were characterised independently of the algorithm: we need a finite equivalence relation $\sim$ such that:

- $\sim$ is a refinement of $\approx$, "has the same shape as"
- if $u \sim v$ and u' is a legal next position after u then there is some $v' \sim u'$ such that v' is a legal next position after v
- if $u' \sim v'$ and u' is a legal next position after u then there is some $v \sim u$ such that v' is a legal next position after v

These conditions capture the intuition that $\sim$, whilst still finite, only relates positions which are the same as far as the game is concerned: it is a relation which is "detailed enough" to answer the question. The algorithm, which can be seen as taking $\approx$ as its starting equivalence relation for refinement, can be shown never to split the classes of such an equivalence relation, and to terminate because of the finite number of equivalence classes.

This approach can be seen as a generalisation of techniques such as the region analysis used by [2] and many others. The idea is to capture the essence of situations in which apparently infinite problems are solved by taking advantage of the fact that there is a tractable abstraction: we incrementally build a representation which is "just detailed enough" to answer the question.

There is more detailed work to be done to exploit this technique, but we believe that it will yield decidability results equivalent, up to notation, to those in the papers dealing with the logics we translate. One way to justify the belief is to observe that equivalence relations of this sort are apparent in the original papers, and may be expected to be preserved by the translations. In the case of model checking $P\mu$ on finite control pi calculus terms, for example, it is intuitively clear that the reason why it works is that, if one ignores names, only finitely many terms and formulae can appear; and the only salient feature of the names is which pairs of names are equal. We expect the notion of shape to be what you get by ignoring names, and the refinement to move towards the finest possible equivalence relation, in which we know everything about which names are equal. This sketch, of course, elides a considerable amount of work which will need to be done to make the argument precise; but we do not expect to encounter essential difficulties.

5 Conclusions

We have defined a general purpose assembly language logic and shown that it is powerful enough to express the properties for which various special purpose temporal logics have been developed. We exhibited translations, and showed that there is reason to conjecture that the images of the translations can be decided uniformly, for example by the algorithm of [5].

We have not so far considered the efficiency of the generic algorithm; the one in [5] is intended only to be an existence proof for such an algorithm. One interesting strand of work would be to investigate to what extent it is possible to develop a generic algorithm whose efficiency competes with more specialised algorithms.

An area of concern is that, as with any translation-based approach to solving verification problems, the fact that the problem is being solved in a translated domain poses serious problems for the tool developer trying to provide meaningful feedback to the user of the tool.

6 Acknowledgements

We thank Mads Dam for helpful comments on issues arising from [1].

References

1. M. Dam, Model Checking Mobile Processes, *Information and Computation* **129** 35-51, 1996.
2. T. A. Henzinger, X. Nicollin, J. Sifakis and S. Yovine, Symbolic Model Checking for Real-time Systems, *Information and Computation* **111** 193–244 (1994).
3. J.G. Henriksen, J. Jensen, M. Jørgensen, N. Klarlund, B. Paige, T. Rauhe, and A. Sandholm, Mona: Monadic Second-order logic in practice, *Proc. TACAS '95*, LNCS **1019** (1995).
4. J. Rathke, *Symbolic Techniques for Value-Passing Calculi*. PhD. thesis, University of Sussex, 1997.
5. P. Stevens,Abstract Games for Infinite State Processes, in *Proc. CONCUR98*, LNCS **1466** 147–162 (1998).
6. P. Stevens, 'Abstract interpretations of games, in *Proc. 2nd International Workshop on Verification, Model Checking and Abstract Interpretation 98*, Venezia TR CS98-12.
7. C. Stirling, Modal and temporal logics for processes. LNCS **1043** 149-237 (1996).