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 abstract 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 first steps in this direction.

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 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.

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

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 cellnames $c, d \ldots$ 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.

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

Finally, we note that the fixpoints are implicitly parametrized on the cells.

3 Use of $A\mu$

To illustrate our logic, we exhibit the key points of translations from existing timed logics. 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:

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

In this slightly obscure formula, the inner maximal fixpoint and the specification clock b^{cl} are used to arrange that the main minimal fixpoint only has to be unwound if time passes.

In a similar style, the timed mu-calculus $T\mu$ of [2] with its binary 'until' operator $p \triangleright q$ can be translated.

As sketched earlier, we can also handle value-passing logics such as [1, 4]. The 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 [6] 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. 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 an approach via *abstract games*, in which classes of cell values are considered together and split only when the analysis requires it; this is a generalisation of techniques such as the region analysis used by [2]. Exploring the application of this technique is future work, but we are optimistic that it will make reasonable model-checking problems in $A\mu$ tractable.

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. Jrgensen, 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. To appear in Proceedings of CONCUR'98.
6. C. Stirling, Modal and temporal logics for processes. LNCS **1043** 149-237 (1996).