# The way forward for DETECTER

Adrian Francalanza[1], Aldrin Seychell[1], Ian Cassar[1], Kurt Cutajar[1], Simon Said[1], and Blanche Schembri[1]

Department of Computer Science, University of Malta
{adrian.francalanza | aldrin.seychell.08}@um.edu.mt

Monitor *correctness* is a prerequisite for the adoption of runtime verification as a lightweight formal technique ensuring program correctness. The tool DETECTER [4, 9] is a runtime verification tool synthesising correct monitors from $\mu$-calculus formulas [6] describing safety Erlang [2] properties. As opposed to previous work on monitor correctness [3, 5], the correctness guarantees of the synthesised monitors in DETECTER are close to the actual implementation: they are based on actual executable monitor translations in the form of Erlang programs, consider aspects impacting monitor runtime behaviour such as monitor instrumentation, and provide guarantees such as the absence of monitor interfere during system execution.

In DETECTER, one can specify safety properties such as

$$\overbrace{\mathsf{max}\Big(X, \Big( \underbrace{([\mathsf{req}][\mathsf{ans}][\mathsf{ans}]\mathsf{ff})}_{(1)} \wedge \underbrace{([\mathsf{req}][\mathsf{ans}]X)}_{(2)} \Big)\Big)}^{(3)}$$

stating that a service request action req cannot be followed by *two* service answers, ans, subformula (1), and that this property must hold indefinitely after repeated sequences of req, ans pairs, subformula (2) in the context of (3). From this formula, the tool synthesises an Erlang monitor that checks for runtime violations of the property; where possible, the monitor employs *concurrent* processes to analyse the system, spawning them judiciously according to the behaviour of the system being monitored, so as to minimise overheads. The monitor instrumentation is *asynchronous* and minimally intrusive, requiring no changes to the source code of the system being monitored; this, in turn, facilitates adoption, which may even happen dynamically, while the system is already running.

Thus far, the focus of the work on DETECTER has focussed on correctness [4, 9]. The subsequent development stages plan to use this work as the foundation for extending the tool's capabilities in terms of both expressivity and efficiency. We discuss these extensions below:

**Data:** We would like to introduce parametrisable actions and universal quantifications on data which would allow us to specify properties such as

$$\mathsf{max}\Big(X, \forall x, y.\Big( \underbrace{([\mathsf{req}(x)][\mathsf{ans}(y)][y \neq x+1]\mathsf{ff})}_{(3)} \wedge \underbrace{([\mathsf{req}(x)][\mathsf{ans}(x+1)]X)}_{(4)} \Big)\Big)$$

Monitored actions may be parametrised *e.g.*, $\mathsf{req}(x)$, and their respective values would then be instantiated at runtime. This allows us to specify symbolic constraints across actions such as that stating that whenever a system is requested a service with value $x$, it must reply with value $x+1$, subformula (4), or otherwise raise a violation, subformula (3).

**Beyond Safety:** So far the tool can only synthesise monitors for a syntactic subset of the $\mu$-calculus describing safety properties [1]. In addition we would like to be able to synthesise monitors for formulas using other $\mu$-calculus constructs, such as those describing monitorable liveness properties (co-safety) [8]. One example would be

$$\mathsf{min}\Big(X, \big( \underbrace{(\langle \mathsf{end} \rangle \mathsf{tt})}_{(5)} \vee \underbrace{(\langle \mathsf{req} \rangle \langle \mathsf{ans} \rangle X)}_{(6)} \big)\Big)$$

specifying that the servicing is correct but finite *i.e.*, there *exists* a sequence of repeated (req,ans) interactions, subformula (6) after which the system terminates by producing a terminating action end, subformula (5).

**Synchronous and Hybrid Monitoring:** Despite its advantages, asynchronous monitoring may lead to late detections from the part of the monitor. We would like to explore the introduction of synchronous monitoring [7] (at the level of individual actions) and assess its impact in terms of overheads. The introduction of synchrony may also affect important properties that come for free in a completely asynchronous setting, such as the indirect effects of monitoring on system behaviour. More specifically, in a synchronous setting, the execution of the system and the monitor become intertwined and it would then be imperative that the monitor never enters infinite loops while the system is waiting for an acknowledgement to continue executing; in a completely asynchronous setting with fair executions, monitor divergences affect system performance but not behaviour.

**Performance optimisations.** The present synthesis in terms of concurrent monitors gives us scope for introducing optimisations in terms of reduced messages amongst (monitor) processes and reduced sub-monitor spawning. Avenues that are currently being explored are those relating to the shortening of chains of forwarders in the synthesised concurrent monitors and upfront pruning of redundant formula branches.

## References

1. Aceto, L., Ingólfsdóttir, A.: Testing Hennessy-Milner Logic with Recursion. In: FoSSaCS'99. pp. 41–55. Springer (1999)
2. Armstrong, J.: Programming Erlang - Software for a Concurrent World. The Pragmatic Bookshelf (2007)
3. Bauer, A., Leucker, M., Schallhart, C.: Runtime Verification for LTL and TLTL. ACM Trans. Softw. Eng. Methodol. 20, 14:1–14:64 (September 2011)
4. Francalanza, A., Seychell, A.: Synthesising correct concurrent runtime monitors - (extended abstract). In: Runtime Verification. LNCS, vol. 8174, pp. 112–129. Springer (2013)

5. Geilen, M.: On the construction of monitors for temporal logic properties. ENTCS 55(2), 181–199 (2001)
6. Kozen, D.: Results on the propositional $\mu$-calculus. Theor. Comput. Sci. 27, 333–354 (1983)
7. Leucker, M., Schallhart, C.: A brief account of runtime verification. JLAP Spec. Ed. (FLACOS'07) 78(5), 293 – 303 (2009)
8. Manna, Z., Pnueli, A.: A hierarchy of temporal properties (invited paper, 1989). In: PODC '90. pp. 377–410. ACM, New York, NY, USA (1990)
9. Seychell, A.: Synthesising Correct Concurrent Runtime Monitors in Erlang. Master's thesis, University of Malta, Malta (2013)