# Synthesising Safety Runtime Enforcement Monitors for $\mu$HML

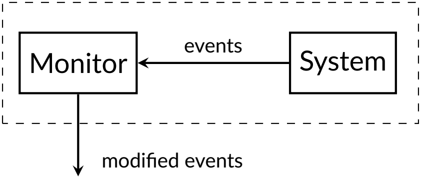
*Luke Collins*
B.Sc. Mathematics & Computer Science (Hons.)

DEPARTMENT OF COMPUTER SCIENCE
**Faculty of ICT**
L-Università ta' Malta

10th June, 2019

L-Università ta' Malta

# Runtime Enforcement




**Runtime enforcement** is the process of analysing the behaviour of software systems at runtime, and *enforcing* “correct” behaviour using software entities called **monitors**.

A monitor wraps itself around the system and analyses all its external interactions. This allows it to transform any incorrect actions by replacing them, suppressing them, or inserting other actions.

# Hennessey–Milner Logic

It is rarely feasible to build *ad hoc* monitors from scratch. Instead, the correctness specification of a system is expressed as a formula in some **logic** with precise formal semantics, and a program designed to interpret this logic synthesises the monitor automatically.

One such logic is the Hennessey–Milner logic with recursion ($\mu$HML).

$$
\begin{array}{rllll}
\varphi, \psi \in \mu\mathrm{HML} ::= & \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} & \text{(falsehood)} \\
\mid & \bigvee_{\gamma \in \Gamma} \varphi_\gamma & \text{(disjunction)} & \mid \bigwedge_{\gamma \in \Gamma} \varphi_\gamma & \text{(conjunction)} \\
\mid & \langle \{p, c\} \rangle \, \varphi & \text{(possibility)} & \mid [\{p, c\}] \, \varphi & \text{(necessity)} \\
\mid & \min X \,.\, \varphi & \text{(least f.p.)} & \mid \max X \,.\, \varphi & \text{(greatest f.p.)} \\
\mid & X & \text{(f.p. variable)} & &
\end{array}
$$

where $\Gamma$ is finite.

# An Example

Consider a server, identified by process id $i$. Whenever the server receives a request ($i\,?\,\mathsf{req}$) it outputs an answer ($i\,!\,\mathsf{ans}$), unless it receives a special request for closure ($i\,?\,\mathsf{cls}$). The server's behaviour may be expressed by the CCS equation

$$p = i\,?\,\mathsf{req} \cdot i\,!\,\mathsf{ans} \cdot p + i\,?\,\mathsf{cls} \cdot \emptyset.$$

A possible correctness specification for this system in $\mu$HML is the safety property

$$\varphi = \max X\,.\,[i\,?\,\mathsf{req}][i\,!\,\mathsf{ans}](X \wedge [i\,!\,\mathsf{ans}]\mathsf{ff}),$$

which ensures that a request ($i\,?\,\mathsf{req}$), followed an answer ($i\,!\,\mathsf{ans}$), is then followed by a request ($i\,?\,\mathsf{req}$), and never another answer ($i\,!\,\mathsf{ans}$), i.e., only one answer is sent following a request.

# Not all Formulæ are Enforceable

Not all formulæ in $\mu$HML are enforceable, i.e., they do not all correspond to "valid" monitors. The **safety fragment**, so-called sHML, is a subset which *is* enforceable.

$$
\begin{array}{rlllll}
\varphi, \psi \in \mathrm{sHML} ::= & \mathsf{tt} & \text{(truth)} & \mid \mathsf{ff} & \text{(falsehood)} \\
\mid & \bigwedge_{\gamma \in \Gamma} \varphi_\gamma & \text{(conjunction)} & \mid [\{p, c\}]\, \varphi^\dagger & \text{(necessity)} \\
\mid & \max X \,.\, \varphi & \text{(greatest f.p.)} & \mid X & \text{(f.p. variable)}
\end{array}
$$

† If $\varphi = \mathsf{ff}$, then $p$ must be an output pattern.

In *Aceto et. al.*, a synthesis function for monitors corresponding to sHML formulæ in **normal form** is given.

# Normal Form

Normal form is *yet another* restriction of $\mu$HML, i.e.,

$$\text{sHML}_{\text{nf}} \subsetneq \text{sHML} \subsetneq \mu\text{HML},$$

however this restriction is only superficial, in that every $\text{sHML}$ formula can be reformulated into an equivalent one in normal form:

$$[\![\text{sHML}_{\text{nf}}]\!] = [\![\text{sHML}]\!] \subsetneq [\![\mu\text{HML}]\!].$$

An sHML formula $\varphi$ is in normal form if the following hold.

1. Branches in a conjunction are pairwise disjoint, i.e. in $\bigwedge_{\gamma\in\Gamma}[\{p_\gamma, c_\gamma\}]\varphi_\gamma$ we have $[\![\{p_{\gamma_1}, c_{\gamma_1}\}]\!] \cap [\![\{p_{\gamma_2}, c_{\gamma_2}\}]\!] = \emptyset$ for $\gamma_1 \neq \gamma_2$;
2. For every $\max X \,.\, \varphi$, we have $X \in \text{fv}(\varphi)$;
3. Every logical variable is guarded by modal necessity.

# What I did in my APT

In *Aceto et. al.*, a theoretical construction is described which transforms a formula $\varphi$ in sHML to an equivalent one in sHML$_{\mathsf{nf}}$.

Using Haskell, I wrote a parser for sHML, and implemented this normalisation algorithm. The resulting formula can then simply be synthesised using the synthesis function below.

$$(\!|X|\!) \stackrel{\text{def}}{=} x \qquad (\!|\mathsf{tt}|\!) \stackrel{\text{def}}{=} (\!|\mathsf{ff}|\!) \stackrel{\text{def}}{=} \mathsf{id} \qquad (\!|\max X \,.\, \varphi|\!) \stackrel{\text{def}}{=} \mathsf{rec}\, x \,.\, (\!|\varphi|\!)$$

$$(\!|\textstyle\bigwedge_{\gamma\in\Gamma}[\{p_\gamma, c_\gamma\}]\varphi_\gamma|\!) \stackrel{\text{def}}{=} \mathsf{rec}\, y \,.\, \sum_{\gamma\in\Gamma} \begin{cases} \{p_\gamma, c_\gamma, \bullet\} & \text{if } \varphi_\gamma = \mathsf{ff} \\ \{p_\gamma, c_\gamma, \underline{p_\gamma}\}(\!|\varphi_\gamma|\!) & \text{otherwise} \end{cases}$$