HÁSKÓLINN Í REYKJAVÍK
REYKJAVIK UNIVERSITY

THE EXPRESSIVE POWER OF DETERMINISTIC MONITORS FOR μHML

SÆVAR ÖRN KJARTANSSON

1 Introduction

This report builds upon the work presented in the paper [1]. In that article, Francalanza, Aceto and Ingólfsdóttir define the notion of a monitor, which runs in parallel with a system and gives a yes/no verdict which indicates whether the system satisfies some specific property expressible in Hennessy-Milner logic with recursion [2]. Here we go one step further and show that we can restrict ourselves to using only monitors that run deterministically while maintaining the expressive power of non-deterministic monitors.

The report is organized as follows. Section 2 gives definitions for the calculus we use to describe systems, the logic we use to describe properties of the systems and the monitors we use to determine these properties. Section 3 presents some of the results given in [1] on which this report builds. Sections 4 and 5 give two different proves for the claim that deterministic monitors are as expressive as non-deterministic ones. The first proof shows that each monitor is equivalent to a deterministic monitor and the second shows that each formula is equivalent to a formula in "deterministic form", for which we can easily define a deterministic monitor.

2 Definitions

In this section we give definitions for the main notations we use in this report. This includes the calculus used to model a system, the logic used to reason about the systems and finally monitors used to verify that a system satisfies some specific formula in the logic.

2.1 The Model

Definition 1. A system is a labeled transition system defined by a tuple

$$\langle \textsc{Proc}, (\textsc{Act} \cup \{\tau\}), \rightarrow \rangle$$

where $\textsc{Proc}$ is a set of states, $\textsc{Act}$ is a set of actions, τ is the silent action and $\rightarrow \subseteq (\textsc{Proc} \times (\textsc{Act} \cup \{\tau\}) \times \textsc{Proc})$ is a transition function. A system is described as a process in the regular fragment of CCS. The syntax of these processes is defined by the following grammar:

$$p, q \in \textsc{Proc} ::= \texttt{nil} \quad | \quad \alpha.p \quad | \quad p + q \quad | \quad \texttt{rec}x.p \quad | \quad x$$

where x comes from a countably infinite set of process variables.

The behaviour is defined by the following derivation rules:

$$\textsc{Act}\frac{}{\alpha.p \xrightarrow{\alpha} p} \qquad \textsc{Rec}\frac{}{\texttt{rec} \xrightarrow{\tau} p[\texttt{rec}x.p/x]}$$

$$\textsc{SelL}\frac{p \xrightarrow{\mu} p'}{p + q \xrightarrow{\mu} p'} \qquad \textsc{SelR}\frac{q \xrightarrow{\mu} q'}{p + q \xrightarrow{\mu} q'}$$

where $\alpha \in \textsc{Act}$ and $\mu \in \textsc{Act} \cup \{\tau\}$.

For each $p, p' \in \textsc{Proc}$ and $\alpha \in \textsc{Act}$, we use $p \overset{\alpha}{\Rightarrow} p'$ to mean that p can derive p' using a single α action and any number of τ actions, $p(\xrightarrow{\tau})^*.\ \xrightarrow{\alpha}\ .(\xrightarrow{\tau})^* p'$. For each $p, p' \in \textsc{Proc}$ and trace $t = \alpha_1.\alpha_2.\ldots.\alpha_r \in \textsc{Act}^*$, we use $p \overset{t}{\Rightarrow} p'$ to mean $p \overset{\alpha_1}{\Longrightarrow} . \overset{\alpha_2}{\Longrightarrow} \ldots \overset{\alpha_r}{\Longrightarrow} p'$ if t is non-empty and $p(\xrightarrow{\tau})^* p'$ if t is the empty trace.

2.2 The Logic

We use Hennessy-Milner logic with recursion (μHML) to describe properties of the processes.

Definition 2. μHML is defined as the set of all closed formulae constructed using the following grammar:

$$\begin{aligned}
\varphi, \psi \in \mu\text{HML} ::= {} & \texttt{tt} && \mid \texttt{ff} \\
\mid {} & \varphi \wedge \psi && \mid \varphi \vee \psi \\
\mid {} & \langle\alpha\rangle\varphi && \mid [\alpha]\varphi \\
\mid {} & \min X.\varphi && \mid \max X.\varphi \\
\mid {} & X
\end{aligned}$$

where X comes from a countable set of logical variables LVar.

Formulae are evaluated in a context of an environment, $\rho : \textsc{LVar} \to 2^{\textsc{Proc}}$, which gives values to the logical variables in the formula. The semantics for these formulae is defined as follows:

$$\begin{aligned}
[\![\texttt{tt}, \rho]\!] &\overset{def}{=} \textsc{Proc} & [\![\texttt{ff}, \rho]\!] &\overset{def}{=} \emptyset \\
[\![\varphi_1 \wedge \varphi_2, \rho]\!] &\overset{def}{=} [\![\varphi_1, \rho]\!] \cap [\![\varphi_2, \rho]\!] & [\![\varphi_1 \vee \varphi_2, \rho]\!] &\overset{def}{=} [\![\varphi_1, \rho]\!] \cup [\![\varphi_2, \rho]\!] \\
[\![[\alpha]\varphi, \rho]\!] &\overset{def}{=} \left\{ p \mid p \overset{\alpha}{\Rightarrow} q \text{ implies } q \in [\![\varphi, \rho]\!] \right\} & [\![\langle\alpha\rangle\varphi, \rho]\!] &\overset{def}{=} \left\{ p \mid p \overset{\alpha}{\Rightarrow} q \text{ and } q \in [\![\varphi, \rho]\!] \right\} \\
[\![\max X.\varphi, \rho]\!] &\overset{def}{=} \bigcap \{ S \mid [\![\varphi, \rho[X \mapsto S]]\!] \subseteq S \} \\
[\![\min X.\varphi, \rho]\!] &\overset{def}{=} \bigcup \{ S \mid S \subseteq [\![\varphi, \rho[X \mapsto S]]\!] \} & [\![X, \rho]\!] &\overset{def}{=} \rho(X)
\end{aligned}$$

Since we only work with closed formulae, the environment, ρ, has no effect on the semantics of a formula and we often drop it and write $[\![\varphi]\!]$ instead of $[\![\varphi, \rho]\!]$.

In some of the proofs in section 5, we use an equivalent representation for μHML formulae that uses systems of equations instead of regular formulae.

Definition 3. We denote a system of equations with a tuple $SYS = (Eq, X_1, \mathcal{Y})$ where X_1 is called the principal variable in SYS, $\mathcal{Y}$ is a set of variables and Eq is

a set of equations:

$$\begin{aligned} X_1 &= F_1 \\ X_2 &= F_2 \\ &\vdots \\ X_n &= F_n \end{aligned}$$

where F_i is an expression in μHML that can contain variables in $\mathcal{Y} \cup \{X_1, X_2, \ldots, X_n\}$.

A solution to a system of equations is a mapping $\rho : \text{LVAR} \to 2^{\text{PROC}}$ where for each equation $X = F$, we have $[\![X, \rho]\!] = [\![F, \rho]\!]$.

For each equation $X = F$ in SYS, when finding a solution for $[\![X, \rho]\!] = [\![F, \rho]\!]$, we are interested in either the maximum solution:

$$X \overset{\max}{=} F$$

or the minimum:

$$X \overset{\min}{=} F$$

For each process $p \in \text{PROC}$, p satisfies SYS iff $p \in [\![X_1, \rho]\!]$ and we say that a formula φ and a system of equations SYS are equivalent iff $[\![X_1, \rho]\!] = [\![\varphi, \rho]\!]$, where X_1 is the principal variable in SYS.

2.3 Monitors

We now define the notion of a monitor. The definition is the same as the one given in [1] except for a slight change in the behaviour of a verdict.

Definition 4. The syntax of a monitor is identical to that of a regular CCS process with the exception that the `nil` process is replaced by verdicts. A verdict can be one of `yes`, `no` and `end` which represent acceptance, rejection and termination respectively. A monitor is defined by the following grammar:

$$\begin{array}{rrrrrr} m, n \in \text{MON} ::= & v & \mid \alpha.m & \mid m + n & \mid \texttt{rec}x.m & \mid x \\ v \in \text{VERD} ::= & \texttt{end} & \mid \texttt{no} & \mid \texttt{yes} & & \end{array}$$

where x comes from a countably infinite set of monitor variables.

The behaviour of a monitor is defined by the following derivation rules:

$$\text{mACT}\frac{}{\alpha.m \xrightarrow{\alpha} m} \qquad \text{mREC}\frac{}{\texttt{rec}x.m \xrightarrow{\tau} m[\texttt{rec}x.m/x]}$$

$$\text{mSELL}\frac{m \xrightarrow{\mu} m'}{m + n \xrightarrow{\mu} m'} \qquad \text{mSELR}\frac{n \xrightarrow{\mu} n'}{m + n \xrightarrow{\mu} n'}$$

$$\text{mVERD}\frac{}{v \xrightarrow{\mu} v}$$

where $\alpha \in \text{ACT}$ and $\mu \in \text{ACT} \cup \{\tau\}$.

For each $m, m' \in \text{MON}$ and $\alpha \in \text{ACT}$, we use $m \overset{\alpha}{\Rightarrow} m'$ to mean that m can derive m' using a single α action and any number of τ actions, $m(\xrightarrow{\tau})^*.\xrightarrow{\alpha}.(\xrightarrow{\tau})^* m'$. For each $m, m' \in \text{MON}$ and trace $t = \alpha_1.\alpha_2.\ldots.\alpha_r \in \text{ACT}^*$, we use $m \overset{t}{\Rightarrow} m'$ to mean $m \overset{\alpha_1}{\Rightarrow} . \overset{\alpha_2}{\Rightarrow} \ldots \overset{\alpha_r}{\Rightarrow} m'$ if t is non-empty and $m(\xrightarrow{\tau})^* m'$ if t is the empty trace.

In [1] a monitor that has reached a verdict can perform any action $\alpha \in \text{ACT}$ but cannot perform the transparent τ action, $v \overset{\tau}{\not\to}$. However since $\mu \in \text{ACT} \cup \{\tau\}$, we allow the monitor to perform τ. This change does not affect any of the results given in [1] but is important for the proof given in section 4.

In order for a monitor to run deterministically, it cannot contain a non-deterministic choice between two sub-monitors reached by the same action, $m \xrightarrow{\alpha} n_1$ and $m \xrightarrow{\alpha} n_2$, or a non-deterministic choice between performing an action or performing the transparent action, $m \xrightarrow{\alpha} n_1$ and $m \xrightarrow{\tau} n_2$. It must also be impossible to derive these choices using the derivation rules above.

Definition 5. A monitor m is deterministic iff

- each bound variable x in m is prefixed by an action, $\alpha.x$
- each recursive operator $\texttt{rec}x.n$ in m is prefixed by an action, $\alpha.\texttt{rec}x.n$.
- for each pair of sub-monitors $\alpha_1.n_1$ and $\alpha_2.n_2$ that occur in the same sum in m, we have have $\alpha_1 \neq \alpha_2$.

This definition of a deterministic monitor might seem restrictive as there exist monitors that run deterministically that are not in this form, e.g. $\texttt{rec}x.\texttt{tt}$, but as we will see in section 5, this set of monitors is in fact maximally expressive.

2.4 Monitored system

If a monitor $m \in \textsc{Mon}$ is monitoring a process $p \in \textsc{proc}$, then it must mirror every visible action p performs. If m cannot match an action performed by p, then m becomes the inconclusive `end` verdict. We are only looking at the visible actions and so we allow m and p to perform transparent τ actions independently of each other.

Definition 6. A monitored system is a monitor $m \in \textsc{Mon}$ and a process $p \in \textsc{Proc}$ which are run in parallel, denoted $m \triangleleft p$. The behaviour of a monitored system is defined by the following derivation rules:

$$\textsc{iMon}\frac{p \xrightarrow{\alpha} p' \quad m \xrightarrow{\alpha} m'}{m \triangleleft p \xrightarrow{\alpha} m' \triangleleft p'} \qquad \textsc{iTer}\frac{p \xrightarrow{\alpha} p' \quad m \overset{\alpha}{\not\longmapsto} \quad m \overset{\tau}{\not\longmapsto}}{m \triangleleft p \xrightarrow{\alpha} \texttt{end} \triangleleft p'}$$

$$\textsc{iAsyP}\frac{p \xrightarrow{\tau} p'}{m \triangleleft p \xrightarrow{\tau} m \triangleleft p'} \qquad \textsc{iAsyM}\frac{m \xrightarrow{\tau} m'}{m \triangleleft p \xrightarrow{\tau} m' \triangleleft p}$$

If a monitored system $m \triangleleft p$ can derive the `yes` verdict, we say that m accepts p, and similarly m rejects p if the monitored system can derive `no`.

Definition 7 (Acceptance/Rejection)). $\mathbf{acc}(m,p) \overset{def}{=} \exists t, p'.m \triangleleft p \overset{t}{\Rightarrow} \texttt{yes} \triangleleft p'$ and $\mathbf{rej}(m,p) \overset{def}{=} \exists t, p'.m \triangleleft p \overset{t}{\Rightarrow} \texttt{no} \triangleleft p'$.

2.5 Correspondence

We now define a correspondence between μHML formulae and verdicts given by monitors.

A monitor m monitors soundly for a formula $\varphi \in \mu\text{HML}$ if a verdict the monitor gives in a monitored system indicates whether the monitored process satisfies φ or not.

Definition 8 (Sound monitoring). $\mathbf{smon}(m,\varphi) \overset{def}{=} \forall p.(\mathbf{acc}(p,m)$ implies $p \in [\![\varphi]\!])$ and $(\mathbf{rej}(p,m)$ implies $p \notin [\![\varphi]\!])$.

A monitor is satisfaction complete with regards to a formula if it accepts all processes that satisfy the formula and a monitor is violation complete if it rejects all processes that do not satisfy the formula. A monitor that is either satisfaction or violation complete is partially complete.

Definition 9 (Satisfaction/Violation/Partially-Complete Monitoring).

$$\mathbf{scmon}(m,\varphi) \stackrel{def}{=} \forall p.(p \in [\![\varphi]\!] \text{ implies } \mathbf{acc}(p,m))$$
$$\mathbf{vcmon}(m,\varphi) \stackrel{def}{=} \forall p.(p \notin [\![\varphi]\!] \text{ implies } \mathbf{rej}(p,m))$$
$$\mathbf{cmon}(m,\varphi) \stackrel{def}{=} \mathbf{scmon}(m,\varphi) \text{ or } \mathbf{vcmon}(m,\varphi)$$

A monitor monitors for a formula if it can do it soundly and partially completely.

Definition 10 (Monitoring). $\mathbf{mon}(m,\varphi) \stackrel{def}{=} \mathbf{smon}(m,\varphi)$ and $\mathbf{cmon}(m,\varphi)$.

3 Previous Results

The main result from [1] is to define a subset of μHML which is monitorable and show that it is maximally expressive. This subset is called mHML and consists of the safe and co-safe syntactic subsets of μHML, sHML and cHML respectively.

Definition 11 (Monitorable Logic). $\text{mHML} \stackrel{def}{=} \text{sHML} \cup \text{cHML}$

$$\theta, \vartheta \in \text{sHML} ::= \mathtt{tt} \mid \mathtt{ff} \mid [\alpha]\theta \mid \theta \wedge \vartheta \mid \max X.\theta \mid X$$
$$\pi, \varpi \in \text{cHML} ::= \mathtt{tt} \mid \mathtt{ff} \mid \langle\alpha\rangle\pi \mid \pi \vee \varpi \mid \min X.\pi \mid X$$

In order to prove that mHML is monitorable, in [1] Francalanza, Aceto and Ingólfsdóttir define a monitor synthesis function, $(\!|-|\!)$, that maps formulae to monitors and show that for each $\varphi \in \text{mHML}$, $\mathbf{mon}((\!|\varphi|\!),\varphi)$ holds.

This function is used in the proofs in sections 4 and 5 and so we will give the definition here.

Definition 12 (Monitor Synthesis).

$$(\!|\mathtt{tt}|\!) \stackrel{def}{=} \mathtt{yes} \qquad (\!|\mathtt{ff}|\!) \stackrel{def}{=} \mathtt{no} \qquad (\!|X|\!) \stackrel{def}{=} x$$

$$(\!|[\alpha]\psi|\!) \stackrel{def}{=} \begin{cases} \alpha.(\!|\psi|\!) & \text{if } (\!|\psi|\!) \neq \mathtt{yes} \\ \mathtt{yes} & \text{otherwise} \end{cases} \qquad (\!|\langle\alpha\rangle\psi|\!) \stackrel{def}{=} \begin{cases} \alpha.(\!|\psi|\!) & \text{if } (\!|\psi|\!) \neq \mathtt{no} \\ \mathtt{no} & \text{otherwise} \end{cases}$$

$$(\!|\psi_1 \wedge \psi_2|\!) \stackrel{def}{=} \begin{cases} (\!|\psi_1|\!) & \text{if } (\!|\psi_2|\!) = \mathtt{yes} \\ (\!|\psi_2|\!) & \text{if } (\!|\psi_1|\!) = \mathtt{yes} \\ (\!|\psi_1|\!) + (\!|\psi_2|\!) & \text{otherwise} \end{cases} \qquad (\!|\psi_1 \vee \psi_2|\!) \stackrel{def}{=} \begin{cases} (\!|\psi_1|\!) & \text{if } (\!|\psi_2|\!) = \mathtt{no} \\ (\!|\psi_2|\!) & \text{if } (\!|\psi_1|\!) = \mathtt{no} \\ (\!|\psi_1|\!) + (\!|\psi_2|\!) & \text{otherwise} \end{cases}$$

$$(\!|\max X.\psi|\!) \stackrel{def}{=} \begin{cases} \mathtt{rec}x.(\!|\psi|\!) & \text{if } (\!|\psi|\!) \neq \mathtt{yes} \\ \mathtt{yes} & \text{otherwise} \end{cases} \qquad (\!|\min X.\psi|\!) \stackrel{def}{=} \begin{cases} \mathtt{rec}X.\psi & \text{if } (\!|\psi|\!) \neq \mathtt{no} \\ \mathtt{no} & \text{otherwise} \end{cases}$$

Theorem 1 (Monitorability). For each $\varphi \in \text{mHML}$, $\mathbf{mon}((\!|\varphi|\!),\varphi)$ holds.

proof. See proof of thm. 1 in [1]. □

Example 1. Assume that we have a very simple web server p that alternates between accepting a request from a client, `req`, and sending a response back, `res`, until the server terminates, `cls`. We want to verify that the server cannot terminate while in the middle of serving a client (after executing `req` but before executing `res`). We can encode this property in the following sHML formula

$$\varphi = \max X.([\mathtt{req}][\mathtt{cls}]\mathtt{ff} \wedge [\mathtt{req}][\mathtt{res}]X)$$

We can then define a violation complete monitor for this formula using the monitor synthesis function:

$$m = \texttt{rec}x.(\texttt{req.cls.no} + \texttt{req.res}.x)$$

Since m contains a choice between $\texttt{req.cls.ff}$ and $\texttt{req.res}.x$, it is non-deterministic. However it is possible to define a deterministic monitor that monitors for φ. For example:

$$m' = \texttt{req}.(\texttt{res.rec}x.\texttt{req}.(\texttt{res}.x + \texttt{cls.no}) + \texttt{cls.no})$$

In general, given a monitor produced by the monitor synthesis function, we can define a deterministic monitor that is equivalent to it.

Theorem 2. For each formula $\varphi \in$ MHML, there exists a deterministic monitor, $m \in$ MON, such that $\mathbf{mon}(m, \varphi)$.

The rest of this report is devoted to providing two proofs for this theorem. The first one is presented in section 4 and shows that for any valid monitor, there exists a deterministic monitor that is equivalent to it. The second proof is presented in section 5 and show that there is a subset of MHML for which the monitor synthesis function will always produce a deterministic monitor and that this subset is a maximally expressive subset of MHML.

4 Monitor Rewriting

In this section we show that for each monitor, there exists an equivalent monitor that is deterministic. Then given a formula $\varphi \in$ MHML, we can apply the monitor synthesis function to produce a monitor for φ and then rewrite the monitor such that it runs deterministically.

We use a result given by Rabinovich in [3] which states that each regular CCS process is provably equal to a deterministic one, with regards to trace equivalence, using a small set of algebraic laws.

Definition 13. Let $D :$ PROC $\to$ PROC be a mapping from a regular CCS process to a trace equivalent deterministic regular CCS process. Such a mapping must exists as shown by Rabinovich in [3].

In order to apply D to the monitors, we must define a way to encode them as CCS processes. We do this by replacing each verdict v with a process $v.\texttt{nil}$.

Definition 14. We define a mapping $\pi :$ MON $\to$ PROC as follows:

$$\begin{aligned}
\pi(\alpha.m) &= \alpha.\pi(m) \\
\pi(m + n) &= \pi(m) + \pi(n) \\
\pi(\texttt{rec}x.m) &= \texttt{rec}x.\pi(m) \\
\pi(v) &= v.\texttt{nil} \\
\pi(x) &= x
\end{aligned}$$

We can see that π is a bijection between monitors and a regular subset of CCS processes where each verdict action v must be followed by the $\texttt{nil}$ process and the $\texttt{nil}$ process must be prefixed by a verdict action. The inverse of π is defined as

follows:

$$\begin{aligned}\pi^{-1}(\alpha.p) &= \alpha.\pi^{-1}(p)\\ \pi^{-1}(p+q) &= \pi^{-1}(p)+\pi^{-1}(q)\\ \pi^{-1}(\texttt{rec}x.p) &= \texttt{rec}x.\pi^{-1}(p)\\ \pi^{-1}(v.\texttt{nil}) &= v\\ \pi^{-1}(x) &= x\end{aligned}$$

Now if two monitor encodings are trace equivalent, we want the monitors they encode to be equivalent.

Lemma 1. Let m and n be monitors such that $Trace(\pi(m)) = Trace(\pi(n))$. Then m monitors for a formula φ if and only if n monitors for φ.

proof. Using prop. 1 from [1], we see that it is enough to prove that for each verdict v and trace t, $m \stackrel{t}{\Rightarrow} v$ if and only if $n \stackrel{t}{\Rightarrow} v$.

Assume that for a trace t and a verdict v, we have $m \stackrel{t}{\Rightarrow} v$. By examining the definition of π, we see that $\pi(m) \stackrel{t}{\Rightarrow} v.\texttt{nil}$ and so $t.v \in Trace(\pi(m))$. Since $\pi(m)$ and $\pi(n)$ are trace equivalent, we have $\pi(n) \stackrel{t}{\Rightarrow} v.\texttt{nil}$ and so $n \stackrel{t}{\Rightarrow} v$.

By symmetry, if $n \stackrel{t}{\Rightarrow} v$ for some verdict v and trace t, then $m \stackrel{t}{\Rightarrow} v$. □

Theorem 3 (Monitor Rewriting). For each monitor $m \in \textsc{Mon}$ that monitors for a formula $\varphi \in \textsc{mHML}$, there exists a deterministic monitor $n \in \textsc{Mon}$ that monitors for φ.

proof. We define a new monitor $n = \pi^{-1} \circ D \circ \pi(m)$. Now $\pi(m)$ and $\pi(n)$ are trace equivalent and so by lemma 1, m and n are equivalent.

Although $\pi(n)$ is deterministic, it is not guaranteed that n is deterministic. For example although the process $p + v.\texttt{nil}$ could be deterministic, the monitor $\pi^{-1}(p + v.\texttt{nil}) = \pi^{-1}(p) + v$ is not since v can follow any trace. However by examining the definition of π we see that the only case where non-determinism can be introduced is where we have $\pi^{-1}(p + v.\texttt{nil}) = \pi^{-1}(p) + v$ for some process $p \in \textsc{Proc}$.

We can easily remove these non-deterministic choices by applying the following rule:

$$m + v = \begin{cases} m & \text{if } v = \texttt{end} \\ v & \text{if } v = \texttt{yes} \text{ or } v = \texttt{no} \end{cases}$$

For the case where $v = \texttt{end}$, m will accept/reject exactly the same processes as $m + v$ will.

In the case where $v \in \{\texttt{yes}, \texttt{no}\}$, we have $m + v \xrightarrow{\tau} v$ since in our definition of a monitor, we allow a verdict to perform τ. We also note that if $m + v$ can accept/reject a process, then v can also accept/reject the process, since as shown in thm 2 in [1], all valid monitors are uni-verdict.

After removing each non-deterministic choice from n, the resulting monitor is deterministic and equivalent to n, which in turn is equivalent to m. □

5 Formula Rewriting

In this second approach, we show that each formula $\varphi \in \textsc{mHML}$ is equivalent to some formula in deterministic form which will yield a deterministic monitor if we apply the monitor synthesis function to it. We will only prove this for formulae in sHML but the proof for cHML is completely analogous.

We begin by defining a deterministic form for formulae in sHML.

Definition 15. A formula $\varphi \in$ sHML is in deterministic form iff:

- each bound variable X in φ is prefixed by the $[\alpha]$ operator, $[\alpha]X$
- each maximum fixed point $\max X.\psi$ in φ is prefixed by the $[\alpha]$ operator, $[\alpha]\max X\psi$
- for each pair of formulae $[\alpha_1]\psi_1$ and $[\alpha_2]\psi_2$ that occur in the same conjunction in φ, we have $\alpha_1 \neq \alpha_2$

The following lemma justifies calling these formulae deterministic by showing that applying the monitor synthesis function to them will yield a deterministic monitor.

Lemma 2. Let $\varphi \in$ sHML be in deterministic form. Then $m = (\!|\varphi|\!)$ is deterministic.

proof. By examining the definition of the monitor synthesis function, we can see that $(\!|\varphi|\!)$ does not contain a sub-monitor $\alpha.n_1 + \alpha.n_2$ since it would imply that φ contains a sub-formula $[\alpha]\psi_1 \wedge [\alpha]\psi_2$. Furthermore such a sub-monitor cannot be derived since each bound variable and each maximum fixed point in φ is prefixed by $[\alpha]$ which means each bound variable and each recursive monitor in m is prefixed by α. □

We also define a standard form for formulae in sHML.

Definition 16. A formula $\varphi \in$ sHML is in standard form if all free and unguarded variables in φ are at the top level:

$$\varphi = \varphi' \wedge \bigwedge_{i \in S} X_i$$

where φ' does not contain a free and unguarded variable.

Lemma 3. Each formula in sHML is equivalent to some formula in sHML that is in standard form.

proof. We define a function f as follows:

$$f(\varphi) = \{i \mid X_i \text{ occurs free and unguarded in } \varphi\}$$

where $X_1, X_2, \ldots$ are all the variables that can occur in the formulae.

Then formally our claim is that for each $\varphi \in$ sHML, there exists a formula, $\psi \in$ sHML such that

$$\varphi \equiv \psi \wedge \bigwedge_{i \in f(\varphi)} X_i$$

where $f(\psi) = \emptyset$.

We use induction on the size of φ to prove this and go through each case below.

$\varphi \in \{\mathtt{tt}, \mathtt{ff}\}$: This case holds trivially since $f(\varphi) = \emptyset$ and

$$\varphi \equiv \varphi \wedge \bigwedge_{i \in \emptyset} X_i$$

$\varphi = X_j$: This case holds trivially since $f(\varphi) = \{j\}$ and

$$\varphi \equiv \mathtt{tt} \wedge \bigwedge_{i \in \{j\}} X_i$$

$\varphi = [\alpha]\varphi'$: Since φ is prefixed with the $[\alpha]$ operator, all variables are guarded in φ and so $f(\varphi) = \emptyset$ and

$$\varphi \equiv \varphi \wedge \bigwedge_{i \in \emptyset} X_i$$

$\varphi = \varphi_1 \wedge \varphi_2$**:** By the induction hypothesis, there exist formulae $\psi_1, \psi_2 \in$ sHML such that

$$f(\psi_1) = f(\psi_2) = \emptyset$$
$$\varphi_1 \equiv \psi_1 \wedge \bigwedge_{i \in f(\varphi_1)} X_i$$
$$\varphi_2 \equiv \psi_2 \wedge \bigwedge_{i \in f(\varphi_2)} X_i$$

Using the fact that $\vartheta \wedge \vartheta \equiv \vartheta$ for each formula $\vartheta \in \mu$HML, we have

$$\begin{aligned}
\varphi &\equiv \varphi_1 \wedge \varphi_2 \\
&\equiv \left(\psi_1 \wedge \bigwedge_{i \in f(\varphi_1)} X_i\right) \wedge \left(\psi_2 \wedge \bigwedge_{i \in f(\varphi_2)} X_i\right) \\
&\equiv (\psi_1 \wedge \psi_2) \wedge \bigwedge_{i \in f(\varphi_1)} X_i \wedge \bigwedge_{i \in f(\varphi_2)} X_i \\
&\equiv (\psi_1 \wedge \psi_2) \wedge \bigwedge_{i \in f(\varphi_1) \cup f(\varphi_2)} X_i
\end{aligned}$$

and since $f(\psi_1) = f(\psi_2) = \emptyset$, we have $f(\psi_1 \wedge \psi_2) = \emptyset$.

Each free and unguarded variable in φ must either be free and unguarded in φ_1 or φ_2 and each such variable in φ_1 or φ_2 must also be free and unguarded in φ. This gives us $f(\varphi_1) \cup f(\varphi_2) = f(\varphi)$ and so we have

$$\varphi \equiv (\psi_1 \wedge \psi_2) \wedge \bigwedge_{i \in f(\varphi)} X_i$$

$\varphi = \max X_j.\varphi'$**:** By the induction hypothesis, there exists a formula $\psi \in$ sHML such that

$$\varphi' \equiv \psi \wedge \bigwedge_{i \in f(\varphi')} X_i$$

where $f(\psi) = \emptyset$.

We use the following equivalences:

$$\max X.\vartheta \equiv \vartheta[\max X.\theta / X] \tag{1}$$
$$\max X.(\vartheta \wedge X) \equiv \max X.\vartheta \tag{2}$$
$$Y[\vartheta/X] \equiv Y \qquad \text{where } X \neq Y \tag{3}$$

From this we get:

$$\begin{aligned}
\varphi &\equiv \max X_j.\varphi' \\
&\equiv \max X_j.\left(\psi \wedge \bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i\right) \\
&\equiv \left(\psi \wedge \bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i\right)\left[\max X_j.\left(\psi \wedge \textstyle\bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i\right) \Big/ X_j\right] \\
&\equiv \psi\left[\max X_j.\left(\psi \wedge \textstyle\bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i\right) \Big/ X_j\right] \wedge \bigwedge_{i \in f(\varphi') \setminus \{j\}} X_i
\end{aligned}$$

Since each variable in ψ is guarded, substituting a variable for a formula will not introduce unguarded variables and so

$$f\left(\psi\left[\max X_j.\left(\psi \wedge \bigwedge\nolimits_{i\in f(\varphi')\setminus\{j\}} X_i\right)\Big/X_j\right]\right) = \emptyset$$

The variables in φ that are free and unguarded are exactly the ones that are free and unguarded in φ', excluding X_j and so we have

$$f(\varphi) = f(\varphi') \setminus \{j\}$$

This gives us:

$$\varphi \equiv \psi\left[\max X_j.\left(\psi \wedge \bigwedge\nolimits_{i\in f(\varphi')\setminus\{j\}} X_i\right)\Big/X_j\right] \wedge \bigwedge_{i\in f(\varphi)} X_i$$

□

We now extend the notion of standard and deterministic forms for systems of equations. Since we will only be working with formulae from sHML which does not include minimum fixed points, we will assume that for each equation, we are interested in the maximum solution.

Definition 17. Let SYS be a system of equations that is equivalent to some formula $\varphi \in$ sHML. We say that an equation, $X_i = F_i$ is in standard form if either $F_i = \texttt{ff}$, or

$$F_i = \bigwedge_{j\in K_i} [\alpha_j]X_j \wedge \bigwedge_{j\in S_i} Y_j$$

for some finite set of indices, K_i and S_i.

We say that SYS is in standard form if every equation in SYS is in standard form.

Lemma 4. For each formula $\varphi \in$ sHML, there exists a system of equations that is equivalent to φ and is in standard form.

proof. We use structural induction to show how we can construct a system of equations from a formula φ that is in standard form. As shown in lemma 3, given a formula $\vartheta \in$ sHML we can define an equivalent formula ϑ' where each free and unguarded variable is at the top level. We can therefore assume that for each fixed point $\max X.\psi$ that occurs as a sub formula in φ, each free and unguarded variable in ψ is at the top level of ψ and using the equivalence $\max X.(\vartheta \wedge X) \equiv \max X.\vartheta$, we can also assume that X does not appear at the top level of ψ.

We now go through the base cases and each of the top level operators that can occur in φ.

- $\varphi = \texttt{tt}$**:** We define a system of equations $SYS = (\{X = \texttt{tt}\}, X, \emptyset)$. Since $\bigwedge_{j\in\emptyset} \vartheta_j \equiv \texttt{tt}$, SYS is in standard form and is equivalent to φ.
- $\varphi = \texttt{ff}$**:** We define a system of equations $SYS = (\{X = \texttt{ff}\}, X, \emptyset)$. SYS is in standard form and is equivalent to φ.
- $\varphi = Y_1$**:** We define a system of equations $SYS = (\{X = Y\}, X, \{Y\})$. SYS is in standard form and is equivalent to φ.
- $\varphi = [\alpha]\psi$**:** By the induction hypothesis, there exists a system of equations, $SYS = (Eq, X_1, \mathcal{Y})$ that is equivalent to ψ and is in standard form.

  We define a new system of equations

  $$SYS' = (Eq \cup \{X_0 = [\alpha]X_1\}, X_0, \mathcal{Y})$$

  Each equation from SYS' is in standard form and so SYS' is in standard form. Since $X_1 = F_1$ is equivalent to ψ, $X_0 = [\alpha]X_1$ is equivalent to $[\alpha]\psi$ which means SYS' is equivalent to φ.

$\varphi = \psi_1 \wedge \psi_2$**:** By the induction hypothesis, there exist systems of equations $SYS_1 = (Eq_1, X_1, \mathcal{Y}_1)$ and $SYS_2 = (Eq_2, Z_1, \mathcal{Y}_2)$ that are equivalent to ψ_1 and ψ_2 respectively and are in standard form. Let $X_1 = F_1$ be the principal equation from SYS_1 and let $Z_1 = G_1$ be the principal equation from SYS_2.

We define a new system of equations

$$SYS = (Eq, X_0, \mathcal{Y}_1 \cup \mathcal{Y}_2)$$

where

$$Eq = Eq_1 \cup Eq_2 \cup \{X_0 = F_1 \wedge G_1\}$$

Now since $X_1 = F_1$ is equivalent to ψ_1 and $Z_1 = G_1$ is equivalent to ψ_2, $X_0 = F_1 \wedge G_1$ is equivalent to $\varphi = \psi_1 \wedge \psi_2$.

Both $X_1 = F_1$ and $Z_1 = G_1$ are in standard form and so we can write them as

$$F_1 = \bigwedge_{j \in K_1} [\alpha_j] X_j \wedge \bigwedge_{j \in S_1} Y_j$$
$$G_1 = \bigwedge_{j \in K_1'} [\alpha_j] Z_j \wedge \bigwedge_{j \in S_1'} Y_j$$

This allows us to rewrite the equation for X_0 as follows:

$$\begin{aligned} X_0 &= F_1 \wedge G_1 \\ &= \bigwedge_{j \in K_1} [\alpha_j] X_j \wedge \bigwedge_{j \in S_1} Y_j \wedge \bigwedge_{j \in K_1'} [\alpha_j] Z_j \wedge \bigwedge_{j \in S_1'} Y_j \\ &= \left(\bigwedge_{j \in K_1'} [\alpha_j] X_j \wedge \bigwedge_{j \in K_1'} [\alpha_j] Z_j \right) \wedge \bigwedge_{j \in S_1 \cup S_1'} Y_j \end{aligned}$$

Now SYS is in standard form and is equivalent to φ.

$\varphi = \max Y.\psi$**:** By the induction hypothesis, there exists a system of equations $SYS = (Eq, X_1, \mathcal{Y})$ that is equivalent to ψ and is in standard form.

If ψ does not contain Y, then $\varphi \equiv \psi$ which means φ is equivalent to SYS and we are done.

If ψ does contain Y then we define a new system of equation:

$$SYS' = (Eq \cup \{Y = F_1\}, Y, \mathcal{Y} \setminus \{Y\})$$

where $X_1 = F_1$. Since $X_1 = F_1$ is equivalent to ψ, F_1 can derive Y and $Y = F_1$, the equation $Y = F_1$ is equivalent to $\max Y.\psi$.

By our assumption that for each maximum fixed point $\max X.\psi$ in φ, X does not appear unguarded in ψ, we know that Y does not appear unguarded in F_1.

However in general we cannot guarantee that Y does not appear unguarded in the equations from SYS, since $Y \in \mathcal{Y}$. To overcome this, we replace each unguarded occurrence of Y with it's corresponding formula F_1. Let $X_i = F_i$ be a formula that contains an unguarded occurrence of Y. Since

$X_i = F_i$ is in standard form in SYS, we have

$$\begin{aligned} X_i &= \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i} Y_j \\ &= \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i \setminus \{t\}} Y_j \quad \wedge Y_t \end{aligned}$$

where $Y = Y_t$. We now redefine X_i by replacing the unguarded occurrence of Y_t with F_1:

$$\begin{aligned} X_i &= \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i \setminus \{t\}} Y_j \quad \wedge F_1 \\ &= \bigwedge_{j \in K_i} [\alpha_j] X_j \wedge \bigwedge_{j \in S_i \setminus \{t\}} Y_j \wedge \bigwedge_{j \in K_1} [\alpha_j] X_j \wedge \bigwedge_{j \in S_1} Y_j \\ &= \bigwedge_{j \in K_i \cup K_1} [\alpha_j] X_j \wedge \bigwedge_{j \in (S_i \setminus \{j\}) \cup S_1} Y_j \end{aligned}$$

and X_i is in standard form.

Since $X_1 = F_1$ is in standard form in SYS, $Y = F_1$ must be in standard form in SYS'. For all other equations from SYS, we can define equivalent equations that are in standard form in SYS' by replacing any unguarded occurrence of Y with F_1. All equations in SYS' are in standard form and since $Y = F_1$ is equivalent to φ, this case holds.

□

Definition 18. Let $SYS = (Eq, X_1, \mathcal{Y})$ be a system of equations equivalent to a formula in sHML. We say that an equation $X = F$ in Eq is in deterministic form iff:

- any occurrence of some X_i in F must be prefixed with a box operator, $[\alpha]X_i$.
- each maximum fixed point $\max Y.\psi$ in F is prefixed by a box operator, $[\alpha] \max Y.F'$.
- for each pair of formulae, $[\alpha_1]F_1$ and $[\alpha_2]F_2$, that appear in the same conjunction in F, we have $\alpha_1 \neq \alpha_2$.

We say that SYS is in deterministic form if every equation in Eq is in deterministic form.

Lemma 5. For each sHML system of equations in standard form, there exists an equivalent system of equations that is in deterministic form.

proof. Let $SYS = (Eq, X_1, \mathcal{Y})$ be a system of equations in standard form that is equivalent to a formula $\varphi \in$ sHML.

We define some useful functions:

$$\begin{aligned} S(i) &= \{\alpha_j \mid [\alpha_j]X_j \text{ is a sub formula in } F_i\} \\ D(i, \alpha) &= \{r \mid [\alpha]X_r \text{ is a sub formula in } F_i\} \\ E(i) &= \{r \mid Y_r \text{ is unguarded in } F_i\} \end{aligned}$$

We also define these functions for subsets $Q \subseteq \{1, 2, \ldots, n\}$:

$$S(Q) = \bigcup_{i \in Q} S(i)$$
$$D(Q, \alpha) = \bigcup_{i \in Q} D(i, \alpha)$$
$$E(Q) = \bigcup_{i \in Q} E(i)$$

Now for each equation $X_i = F_i$ where $F_i \neq \texttt{ff}$, using these functions we can rewrite the equation as follows:

$$X_i = \bigwedge_{\alpha \in S(i)} \Big([\alpha] \bigwedge_{i \in D(i,\alpha)} X_j \Big) \wedge \bigwedge_{j \in E(i)} Y_i$$

This equation is not in deterministic form since it contains the conjunction of variables $\bigwedge_{j \in D(i,\alpha)} X_j$. To fix this, we define a new variable X_Q for each subset $Q \subseteq \{1, 2, \ldots, n\}$, such that X_Q is equivalent to $\wedge_{j \in Q} X_j$ and then we can write X_i as follows:

$$X_i = \bigwedge_{\alpha \in S(i)} [\alpha] X_{D(i,\alpha)} \wedge \bigwedge_{j \in E(i)} Y_i$$

Now if we can give a deterministic definition for X_Q, then the whole system is in deterministic form.

X_Q is equivalent to $\bigwedge_{j \in Q} X_j$ and so is equivalent to $\bigwedge_{j \in Q} F_j$. If for any $j \in Q$ we have $F_i = \texttt{ff}$ then $\bigwedge_{j \in Q} F_j = \texttt{ff}$ and we let $X_Q = \texttt{ff}$. Otherwise, we can define a deterministic form for X_Q as follows:

$$X_Q = \bigwedge_{\alpha \in S(Q)} [\alpha] X_{D(Q,\alpha)} \wedge \bigwedge_{j \in E(Q)} Y_j$$

We add all of these equations to the system. We did not change the formula that SYS describes and SYS is in deterministic form. □

Lemma 6. Let $SYS = (Eq, X_1, \mathcal{Y})$ be a sHML system of equations in deterministic form. There exists a formula $\varphi \in$ sHML that is in deterministic form and is equivalent to SYS.

proof. We use proof by induction on the number of equations in SYS.

For the base case, we assume that SYS contains a single equation, $X_1 = F_1$. Since $X_1 = F_1$ is the principal equation in SYS, SYS is equivalent to the formula $\max X_1.F_1$ and so is equivalent to $G = F_1[\max X_1.F_1/X_1]$. Since any occurence of X_1 in F_1 is prefixed with $[\alpha]$, G is in deterministic form.

Now assume that SYS contains $n > 1$ equations. Let $X_i = F_i$ be an equation that is not the principal equation in SYS. A process satisfies $X_i = F_i$ iff it satisfies $\max X_i.F_i$. We can therefore define a new system of equations, SYS', by replacing each occurence of X_i with $\max X_i.F_i$ in each equation in SYS and then remove $X_i = F_i$ from the system. Since each occurence of X_i is prefixed by $[\alpha]$ in SYS, performing the replacement will not break determinism. Using the induction hypothesis and the fact that SYS' contains fewer equations that SYS, we can convert SYS' into a formula φ that is equivalent to SYS and is in deterministic form. □

Finally, we are ready to prove the main theorem in this section.

Theorem 4. For each formula $\varphi \in$ sHML there exists a formula $\psi \in$ sHML that is equivalent to φ and is in deterministic form.

proof. Follows from lemmas 4, 5 and 6. □

6 Conclusion

The value of this work is to show that we can add a runtime monitor to a system without having a significant effect on the execution time of the system. In general, evaluating a non-deterministic monitor for some specific trace takes exponential time with regards to the length of the trace but by using a deterministic monitor, we can bring the complexity down to linear time.

If one were to implement a verification tool based on runtime monitors one of two approaches could be taken, each reflected in one of the two main proofs presented in this report. The first approach is to create a monitor for the input formula using the monitor synthesis function and then optimize the monitor such that it runs deterministically. The second approach is to rewrite the input formula such that it is in deterministic form and then use the monitor synthesis function to create a deterministic monitor.

References

[1] Adrian Francalanza, Luca Aceto, and Anna Ingolfsdottir. On verifying Hennessy-Milner logic with recursion at runtime. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification*, volume 9333 of *Lecture Notes in Computer Science*, pages 71–86. Springer International Publishing, 2015.

[2] Kim Guldstrand Larsen. Proof systems for satisfiability in Hennessy-Milner logic with recursion. *Theoretical Computer Science*, 72(2&3):265–288, 1990.

[3] Alexander Rabinovich. A complete axiomatisation for trace congruence of finite state behaviors. Technical report, Proceedings of Mathematical Foundations of Programming Semantics (IX), LNCS, 1993.

[4] Luca Aceto, Anna Ingólfsdóttir, Kim Guldstrand Larsen, and Jiri Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, New York, NY, USA, 2007.

[5] Robin Milner. A complete inference system for a class of regular behaviours. *Journal of Computer and System Sciences*, 28(3):439 – 466, 1984.

School of Computer Science, Reykjavik University, Menntavegi 1, 101 Reykjavík, Iceland

E-mail address: saevark12@ru.is