A Monitoring Tool for a Branching-Time Logic

Duncan Attard[1] Adrian Francalanza[1]

[1]CS, ICT, University of Malta, Malta
{duncan.attard.01,adrian.francalanza}@um.edu.mt

30th September 2016

Outline

Property Specification and Monitor Synthesis

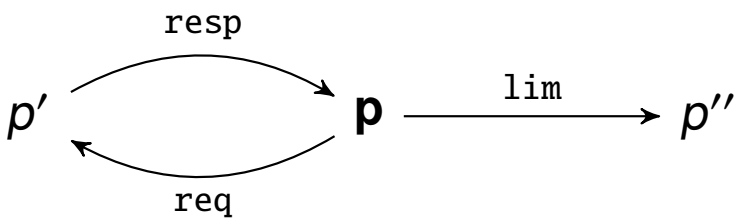
Implementation of a Tool in Erlang

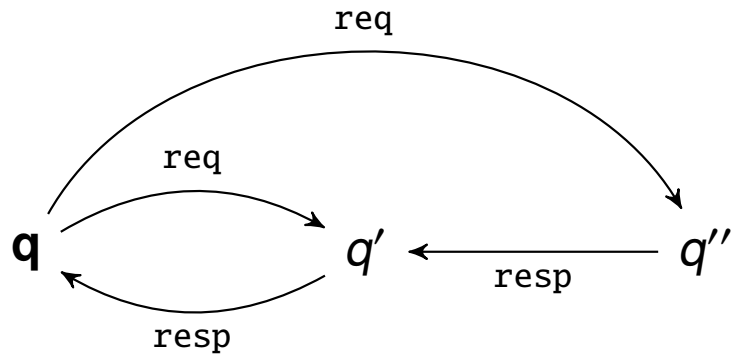
Demo

Modelling Process Behaviour

The behaviour of systems can described using LTSes that model process execution graphs

Example (Two simple servers)





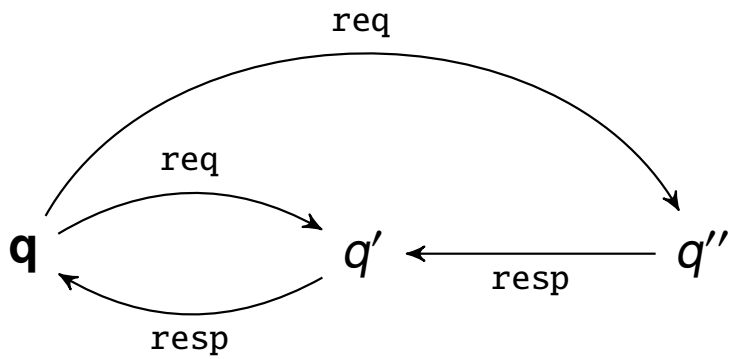
The Branching-Time Logic μHML

Syntax

$$\varphi, \phi \in \mu\text{HML} ::= \mathbf{tt} \mid [\alpha]\varphi \mid \varphi \wedge \phi \mid \mathbf{max}\, X.\varphi \mid X$$
$$\mid \mathbf{ff} \mid \langle\alpha\rangle\varphi \mid \varphi \vee \phi \mid \mathbf{min}\, X.\varphi$$

Example (A liveness property)

$$[\texttt{req}]\langle\texttt{resp}\rangle\mathbf{tt}$$



Cheat sheet

$[\alpha]\mathbf{ff}$ "*cannot* do event α"

$\langle\alpha\rangle\mathbf{tt}$ "*can* do event α"

A Monitorable Subset

The Syntax

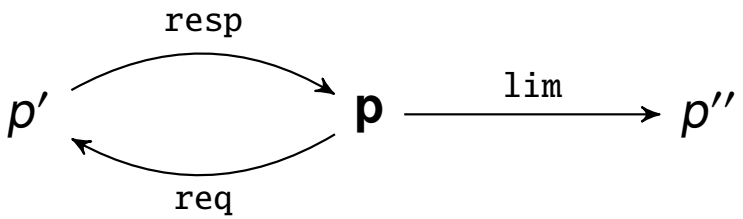
$$\psi \in \text{MHML} \stackrel{\text{def}}{=} \text{sHML} \cup \text{cHML}$$

$$\theta, \vartheta \in \text{sHML} ::= \mathbf{tt} \mid \mathbf{ff} \mid [\alpha]\theta \mid \theta \wedge \vartheta \mid \mathbf{max}\, X.\theta \mid X$$

$$\pi, \varpi \in \text{cHML} ::= \mathbf{tt} \mid \mathbf{ff} \mid \langle\alpha\rangle\pi \mid \pi \vee \varpi \mid \mathbf{min}\, X.\pi \mid X$$

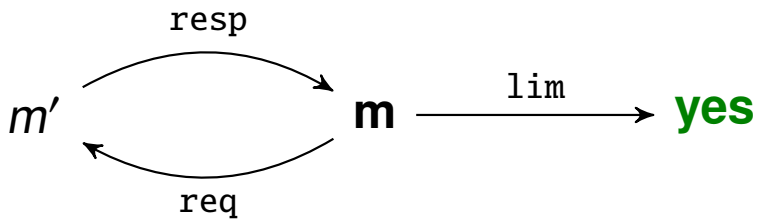
Example (A co-safety property)

$\mathbf{min}\, X.(\langle req\rangle\langle resp\rangle X \vee \langle lim\rangle \mathbf{tt})$



Monitors as LTSes

Example (A monitor for process p)

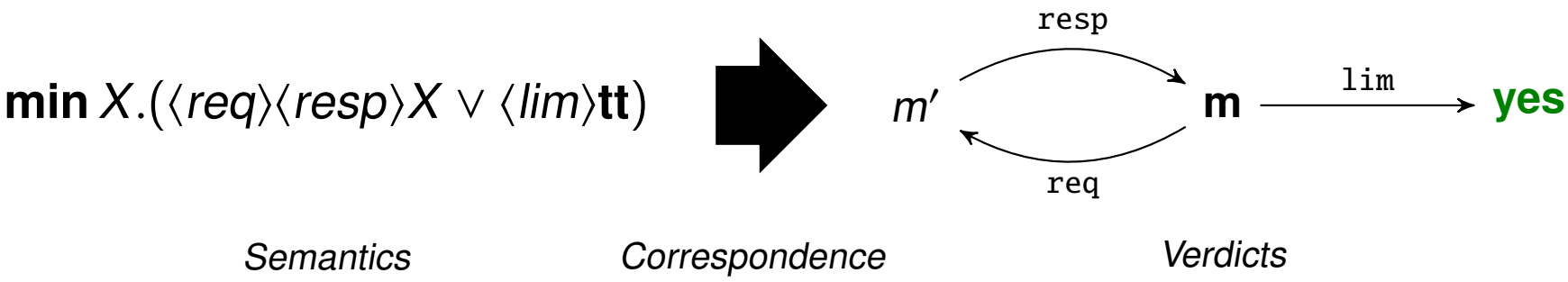


- A **subtle** difference: processes *generate* actions, monitors *analyse* these actions, and yield verdicts
- Conclusive verdicts: **no**, **yes**
- Inconclusive verdict: **end**

Monitor Synthesis

From μHML to Monitors

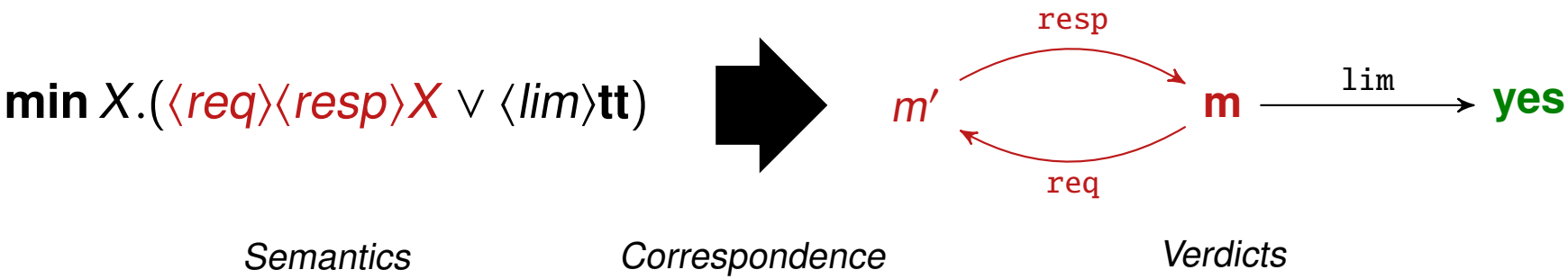
Correct Synthesis (Compositional)



Monitor Synthesis

From μHML to Monitors

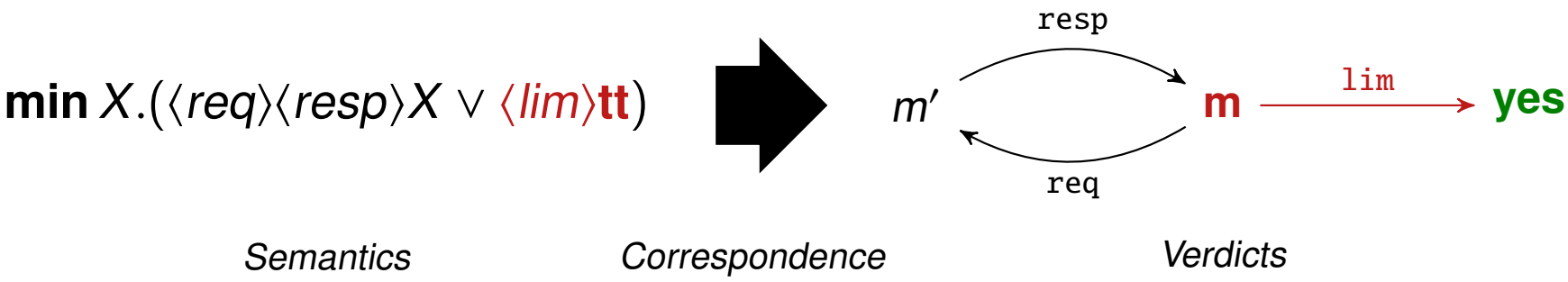
Correct Synthesis (Compositional)



Monitor Synthesis

From μHML to Monitors

Correct Synthesis (Compositional)



Outline

Property Specification and Monitor Synthesis

Implementation of a Tool in Erlang

Demo

Actions = Events with Data

- Why? Because we want to look at the *data* inside actions
- Events with structure permit us to use pattern matching
- Output events: $\alpha = \texttt{client}\ \mathbf{!}\ \{\texttt{resp}, 5\}$
- Input events: $\alpha = \text{server}\ \mathbf{?}\ \{\texttt{req}, \texttt{client}, 5\}$

Actions = Events with Data

- Why? Because we want to look at the *data* inside actions
- Events with structure permit us to use pattern matching
- Output events: $\alpha = \texttt{client}\ \mathbf{!}\ \{\texttt{resp}, 5\}$
- Input events: $\alpha =$ server $\mathbf{?}\ \{\texttt{req}, \texttt{client}, 5\}$

Example (Pattern variable binding)

A successor server *Srv* adds one to any numeric payload it receives from clients *Clt*

$$[\mathit{Srv}\ \mathbf{?}\ \{\texttt{req},\ \mathit{Clt},\ \mathit{Num}\}][\mathit{Clt}\ \mathbf{!}\ \{\texttt{resp},\ \mathit{Num}\}]\mathbf{ff}$$

Safety formula ensures that clients *do not* receive the same value *Num* sent by them to the server

The Tool Synthesis

Example (Non-recursive formula)

$[Srv\ \mathbf{?}\ \{\texttt{req}, Clt, Num\}][Clt\ \mathbf{!}\ \{\texttt{resp}, Num\}]\mathbf{ff}$

- Non-recursive formulae can only observe one interaction before terminating

The Tool Synthesis

Example (Recursive formula)

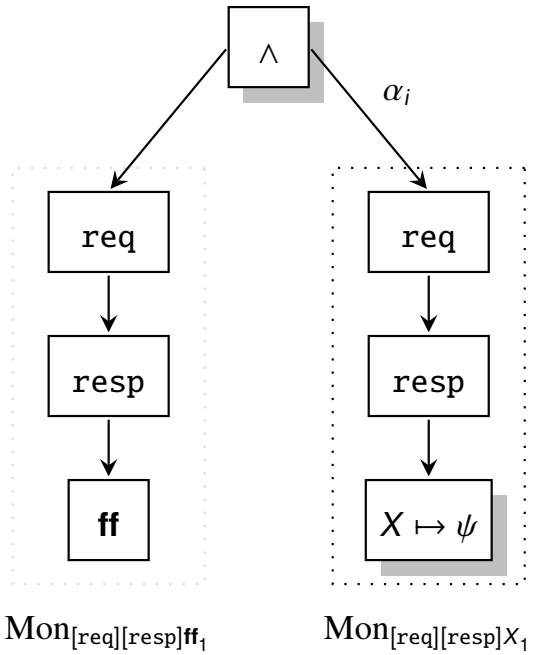
max X.(

$[Srv$ **?** $\{\texttt{req}, Clt, Num\}][Clt$ **!** $\{\texttt{resp}, Num\}]$**ff**

$\wedge$

$[Srv$ **?** $\{\texttt{req}, Clt, Num\}][Clt$ **!** $\{\texttt{resp}, Succ\}]X$

)



- Recursive formulae allow continuous monitoring
- The verdict branch matches events that lead to a detection
- The recursive branch permits the monitor to unfold *lazily*

The Tool Synthesis

Example (Recursive formula)

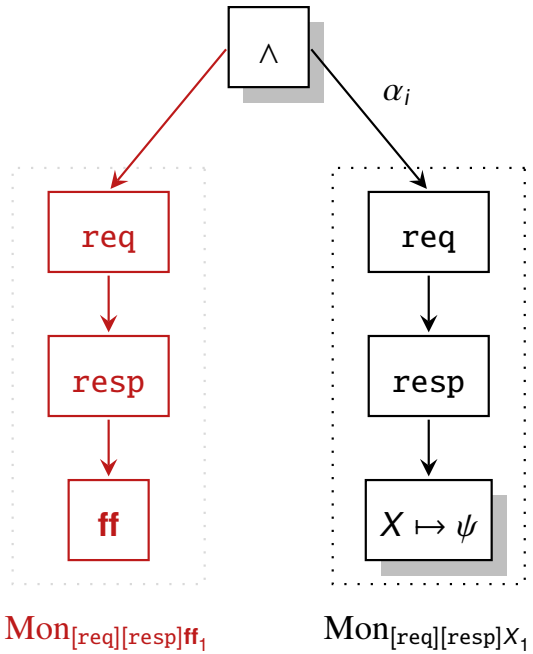
max X.(

$[Srv\ \mathbf{?}\ \{\texttt{req}, Clt, Num\}][Clt\ \mathbf{!}\ \{\texttt{resp}, Num\}]\mathbf{ff}$

$\wedge$

$[Srv\ \mathbf{?}\ \{\texttt{req}, Clt, Num\}][Clt\ \mathbf{!}\ \{\texttt{resp}, Succ\}]X$

)



- Recursive formulae allow continuous monitoring
- The verdict branch matches events that lead to a detection
- The recursive branch permits the monitor to unfold *lazily*

The Tool Synthesis

Example (Recursive formula)

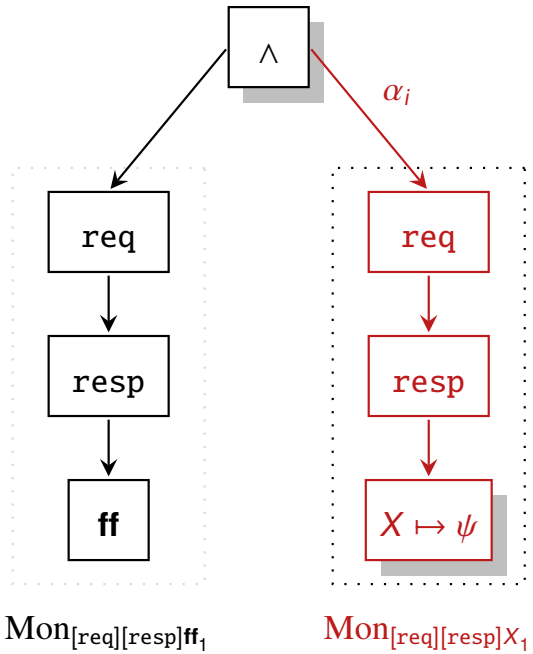
max X.(

$[Srv \; ? \; \{\texttt{req}, Clt, Num\}][Clt \; ! \; \{\texttt{resp}, Num\}]\mathbf{ff}$

$\wedge$

$[Srv \; ? \; \{\texttt{req}, Clt, Num\}][Clt \; ! \; \{\texttt{resp}, Succ\}]X$

)



- Recursive formulae allow continuous monitoring
- The verdict branch matches events that lead to a detection
- The recursive branch permits the monitor to unfold *lazily*

The Tool Synthesis

Example (Recursive formula)

max X.(

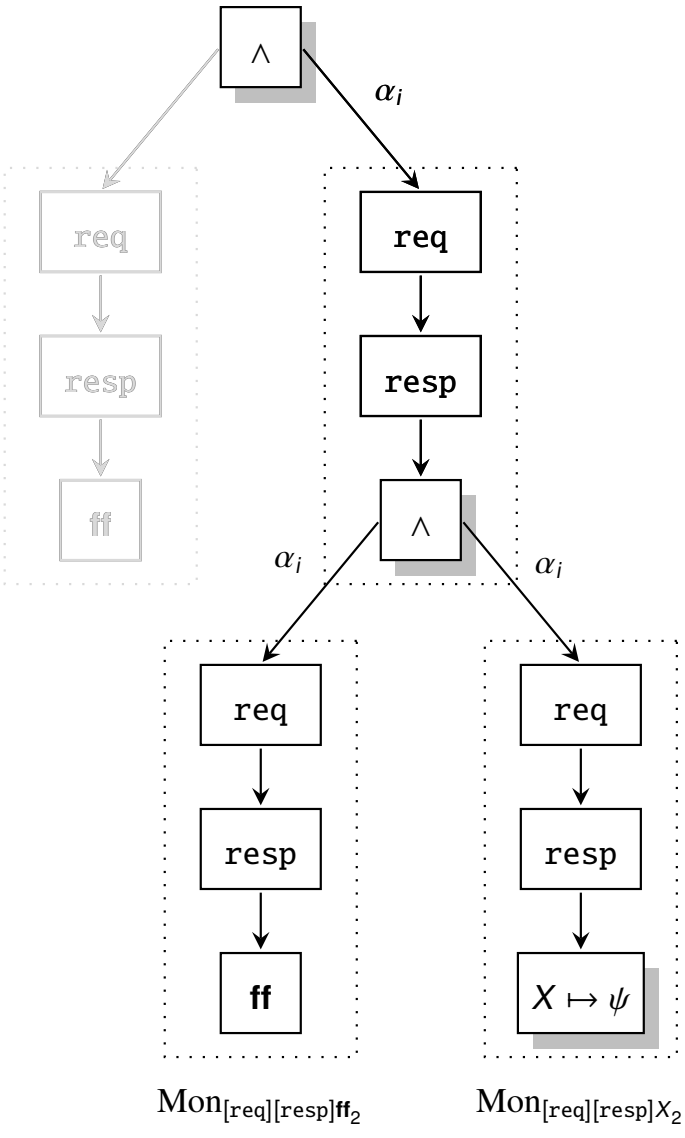
$[Srv$ **?** $\{\texttt{req}, Clt, Num\}][Clt$ **!** $\{\texttt{resp}, Num\}]$**ff**

$\wedge$

$[Srv$ **?** $\{\texttt{req}, Clt, Num\}][Clt$ **!** $\{\texttt{resp}, Succ\}]X$

)

- Recursive formulae allow continuous monitoring
- The verdict branch matches events that lead to a detection
- The recursive branch permits the monitor to unfold *lazily*



Formal *vs.* Tool Synthesis

$$[Srv\ \mathbf{?}\ \{\texttt{req}, Clt, Num\}][Clt\ \mathbf{!}\ \{\texttt{resp}, Num\}]\mathbf{ff}$$

```
formula:mon_nec(
  fun(Action) ->
    case Action of
      {recv, Srv, {req, Clt, Num}} ->
        formula:mon_nec(
          fun(Action) ->
            case Action of
              {send, Clt, {resp, Num}} -> formula:mon_ff();
              _ -> formula:mon_id()
            end
          end);
      _ -> formula:mon_id()
    end
  end)
```

Formal *vs.* Tool Synthesis

$$[Srv\ \mathbf{?}\ \{\mathtt{req},\ Clt,\ Num\}][Clt\ \mathbf{!}\ \{\mathtt{resp},\ Num\}]\mathbf{ff}$$

```
formula:mon_nec(
  fun(Action) ->
    case Action of
      {recv, Srv, {req, Clt, Num}} ->
        formula:mon_nec(
          fun(Action) ->
            case Action of
              {send, Clt, {resp, Num}} -> formula:mon_ff();
              _ -> formula:mon_id()
            end
          end);
      _ -> formula:mon_id()
    end
  end)
```

Formal *vs.* Tool Synthesis

$$[Srv\ \mathbf{?}\ \{\texttt{req}, Clt, Num\}][Clt\ \mathbf{!}\ \{\texttt{resp}, Num\}]\mathbf{ff}$$

```
formula:mon_nec(
  fun(Action) ->
    case Action of
      {recv, Srv, {req, Clt, Num}} ->
        formula:mon_nec(
          fun(Action) ->
            case Action of
              {send, Clt, {resp, Num}} -> formula:mon_ff();
              _ -> formula:mon_id()
            end
          end);
      _ -> formula:mon_id()
    end
  end)
```

Formal *vs.* Tool Synthesis

$$[Srv\ \mathbf{?}\ \{\texttt{req}, Clt, Num\}][Clt\ \mathbf{!}\ \{\texttt{resp}, Num\}]\mathbf{ff}$$

```
formula:mon_nec(
  fun(Action) ->
    case Action of
      {recv, Srv, {req, Clt, Num}} ->
        formula:mon_nec(
          fun(Action) ->
            case Action of
              {send, Clt, {resp, Num}} -> formula:mon_ff();
              _ -> formula:mon_id()
            end
          end);
      _ -> formula:mon_id()
    end
  end)
```

Outline

Thank You!

Resources

The tool can be downloaded from:
`https://bitbucket.org/duncanatt/detecter-lite`

Information and publications from:
`http://www.cs.um.edu.mt/svrg/Tools/detectEr`

Questions?