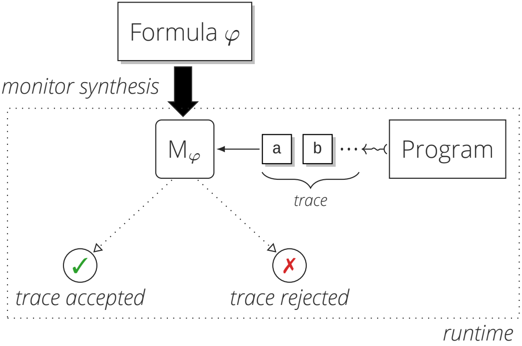
A Monitoring Tool for Linear-Time μ-HML

L. Aceto *et al.* · Monday, September 12th 2022
CS, Reykjavík University and CS, University of Malta

Overview: Runtime verification (RV)

> *RV = property as formula φ + current program trace*



Our monitor verdicts **cannot be changed** once given

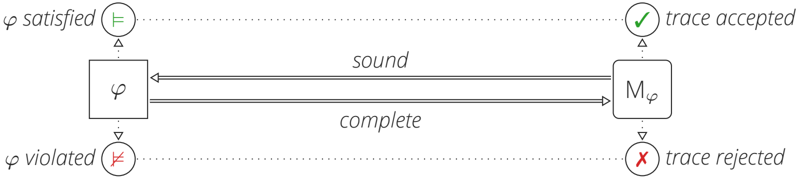
The aspects of modular RV

Monitorability of the logic

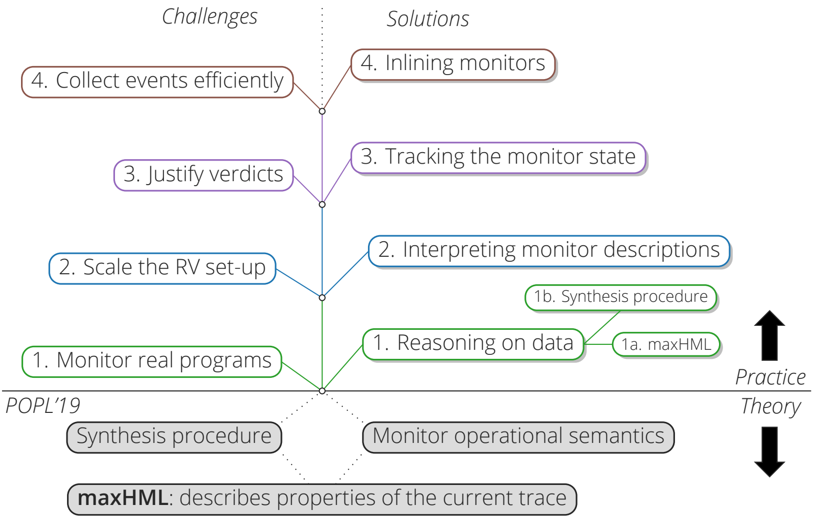
Establishing the set of properties that can be runtime checked

Correctness of monitors

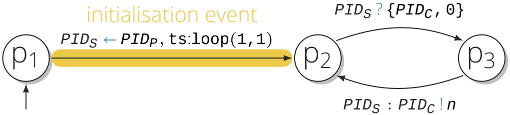
Ensuring that the monitor represents the specified property φ



Making the theory come alive



Token Server: An example in Erlang

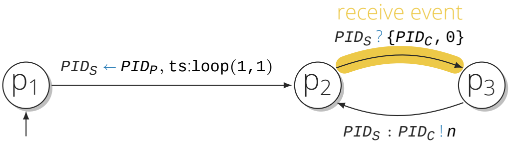


Erlang token server (ts.erl)

```
start(Tok) -> spawn(ts, loop, [Tok, Tok]).

loop(OwnTok, NextTok) ->
    receive
      {Clt, 0} ->
        Clt ! NextTok,
        loop(OwnTok, NextTok + 1)
    end.
```

Token Server: An example in Erlang

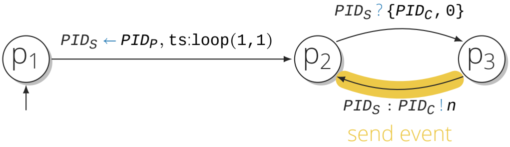


Erlang token server (`ts.erl`)

```
start(Tok) -> spawn(ts, loop, [Tok, Tok]).

loop(OwnTok, NextTok) ->
    receive
      {Clt, 0} ->
        Clt ! NextTok,
        loop(OwnTok, NextTok + 1)
    end.
```

Token Server: An example in Erlang

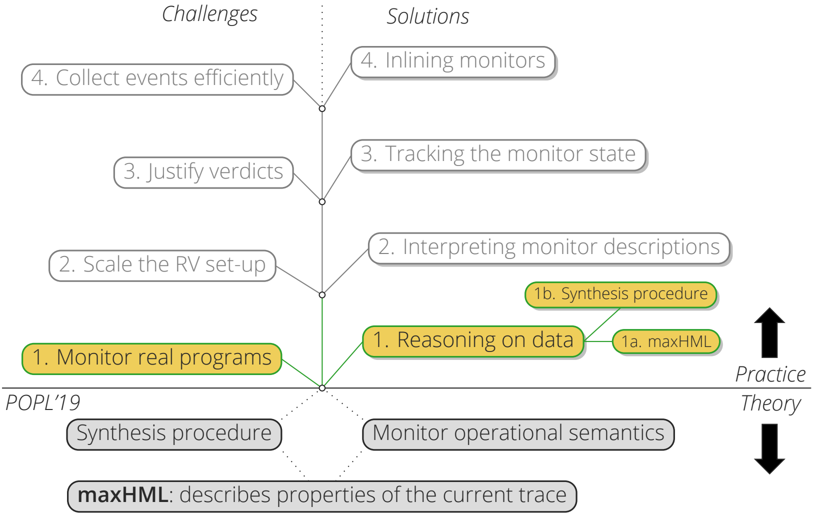


Erlang token server (ts.erl)

```
start(Tok) -> spawn(ts, loop, [Tok, Tok]).

loop(OwnTok, NextTok) ->
    receive
      {Clt, 0} ->
        Clt ! NextTok,
        loop(OwnTok, NextTok + 1)
    end.
```

Making the theory come alive



1. Reasoning on data

Formulae $[\{P \texttt{ when } C\}]\varphi$ in the logic use **symbolic actions**

$$[\{\underbrace{P}\ \texttt{when}\ C\}]\varphi$$

pattern P **matches** the shape of a trace event:

- $\leftarrow$ initialisation event pattern
- ! send event pattern
- ? receive event pattern

1. Reasoning on data

Formulae $[\{P \texttt{ when } C\}]\varphi$ in the logic use **symbolic actions**

$$[\{P \texttt{ when } \underbrace{\boxed{C}}\}]\varphi$$

C is a **decidable** Boolean constraint expression:

- `Var1`, `Var2`, *etc.* — data variables
- `1`, `{1,b}`, *etc.* — data values
- `==`, `/=`, `>`, *etc.* — Boolean and relational operators

1. Reasoning on data

Formulae $[\{P \texttt{ when } C\}]\varphi$ in the logic use **symbolic actions**

binds the free variables

$[\{P \texttt{ when } C\}]\varphi$

1. Reasoning on data

Formulae $[\{P \texttt{ when } C\}]\varphi$ in the logic use **symbolic actions**

binds the free variables

$$\underbrace{[\{P \texttt{ when } C\}]}_{\textbf{necessity}\ \text{modality}}\varphi$$

{*P* when *C*} defines a set of concrete of program events

An event is in this set when:

1. *P* matches the event, instantiating the variables in *C*, **and**
2. *C* is satisfied

1. Reasoning on data

Formulae $[\{P \texttt{ when } C\}]\varphi$ in the logic use **symbolic actions**

binds the free variables

$$[\{P \texttt{ when } C\}]\underbrace{\varphi}_{\text{formula continuation}}$$

$\{P$ when $C\}$ defines a set of concrete of program events

An event is in this set when:

1. P matches the event, instantiating the variables in C, **and**
2. C is satisfied

1. Reasoning on data

Formulae $[\{P \text{ when } C\}]\varphi$ in the logic use **symbolic actions**

$$\underbrace{[\{P \text{ when } C\}]\,\mathbf{ff}}$$

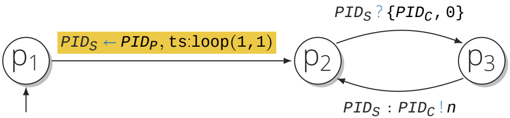
event does not match P or if it does, C is not satisfied

$\{P$ when $C\}$ defines a set of concrete of program events

An event is in this set when:

1. P matches the event, instantiating the variables in C, **and**
2. C is satisfied

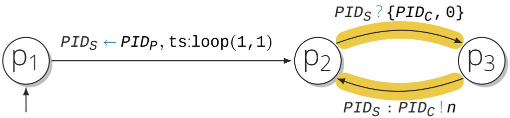
1a. maxHML: an example trace property



The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}]
```

1a. maxHML: an example trace property



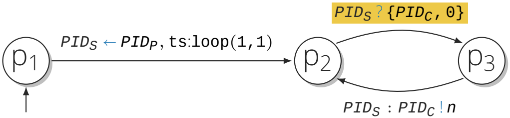
The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(




      ).
```

1a. maxHML: an example trace property



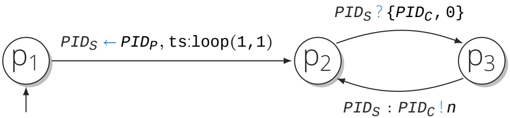
The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}]



   ).
```

1a. maxHML: an example trace property



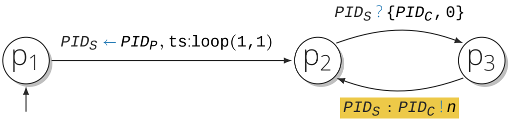
The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](

        and

    )).
```

1a. maxHML: an example trace property

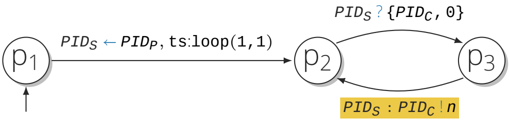


The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and

    )).
```

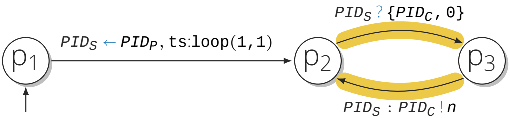
1a. maxHML: an example trace property



The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```

1a. maxHML: an example trace property



The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```

1b. Synthesis procedure

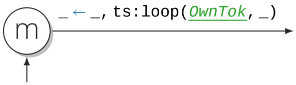
The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```

1b. Synthesis procedure

The server private token is not leaked in client replies

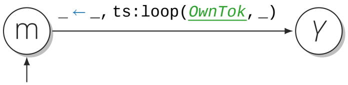
```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```



1b. Synthesis procedure

The server private token is not leaked in client replies

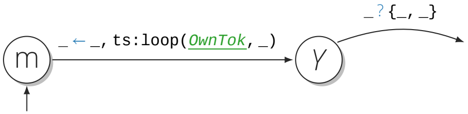
```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```



1b. Synthesis procedure

The server private token is not leaked in client replies

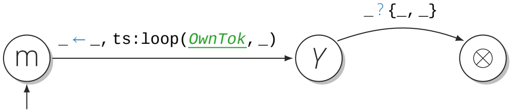
```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
)).
```



1b. Synthesis procedure

The server private token is not leaked in client replies

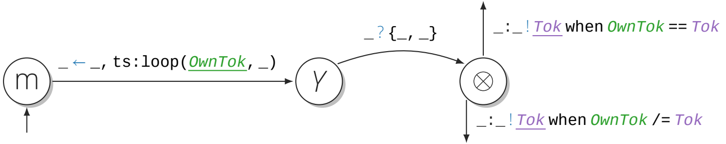
```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```



1b. Synthesis procedure

The server private token is not leaked in client replies

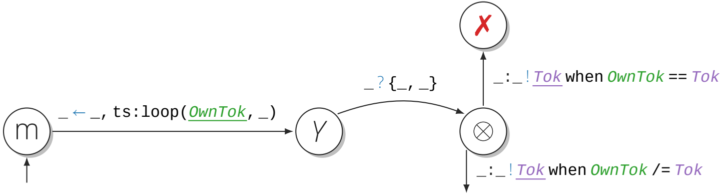
```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```



1b. Synthesis procedure

The server private token is not leaked in client replies

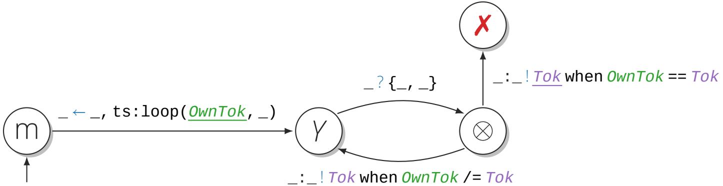
```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```



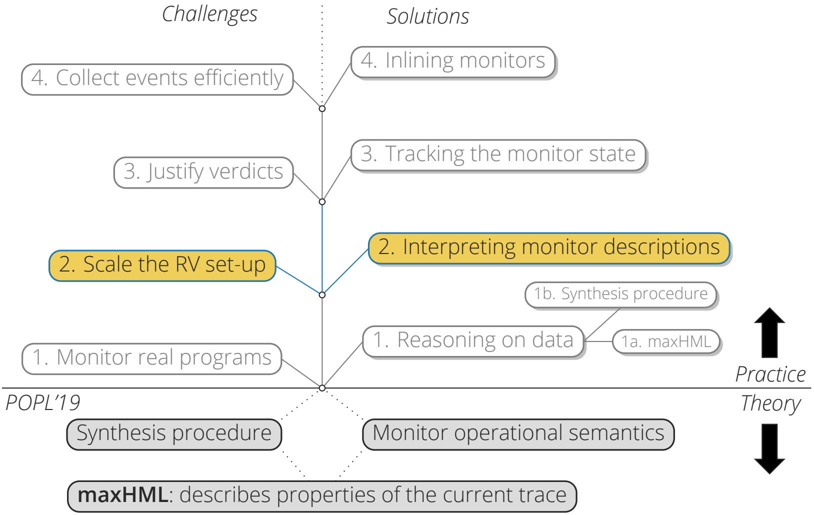
1b. Synthesis procedure

The server private token is not leaked in client replies

```
[{_ ← _, ts:loop(OwnTok, _)}] max Y.(
    [{_ ? {_, _}}](
        [{_:_ ! Tok when OwnTok == Tok}] ff
        and
        [{_:_ ! Tok when OwnTok /= Tok}] Y
    )).
```



Making the theory come alive

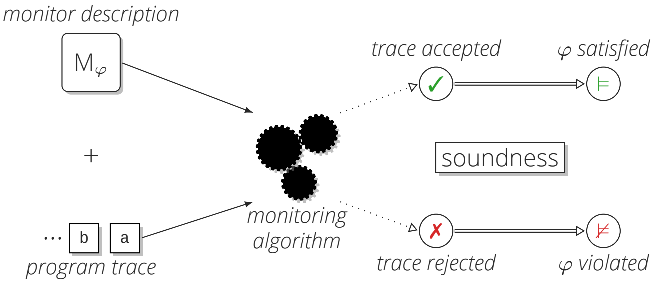


2. Interpreting monitor descriptions

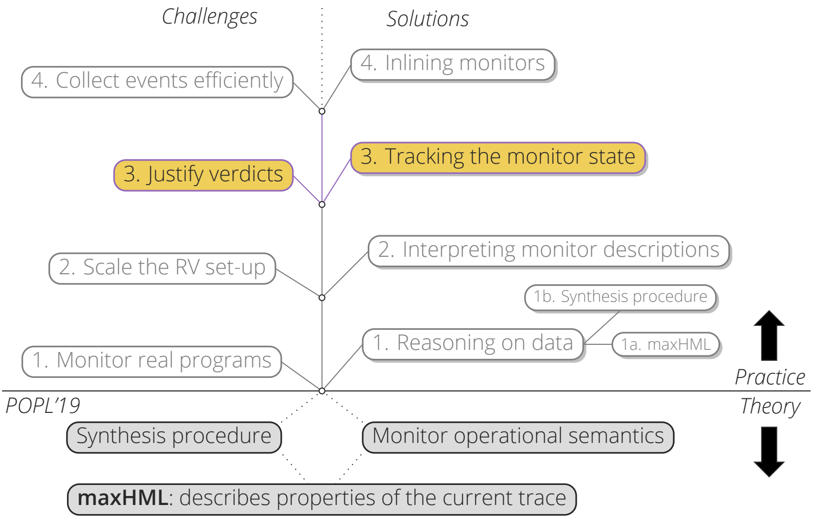
Our algorithm determinises monitors **on-the-fly**

Monitor descriptions are **instantiated** with trace event data

Scalability: we **emulate** disjunctive and conjunctive parallelism

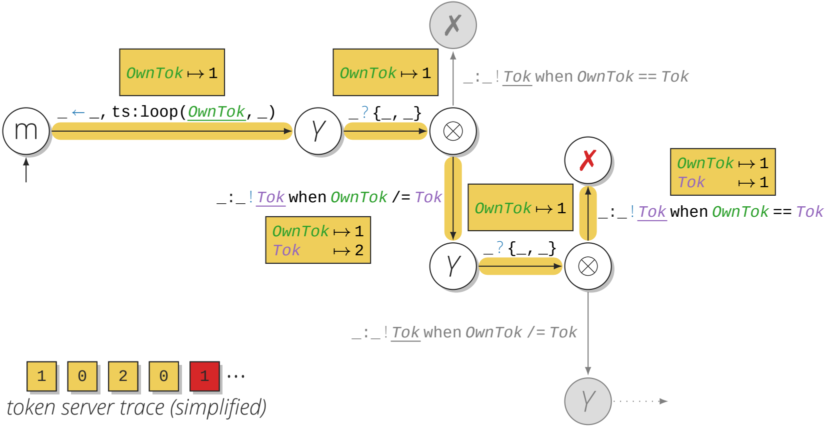


Making the theory come alive

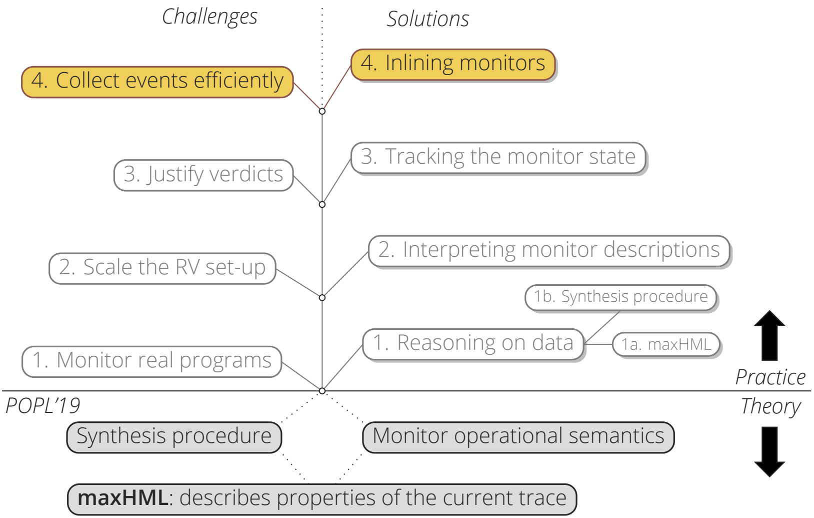


3. Tracking the monitor state

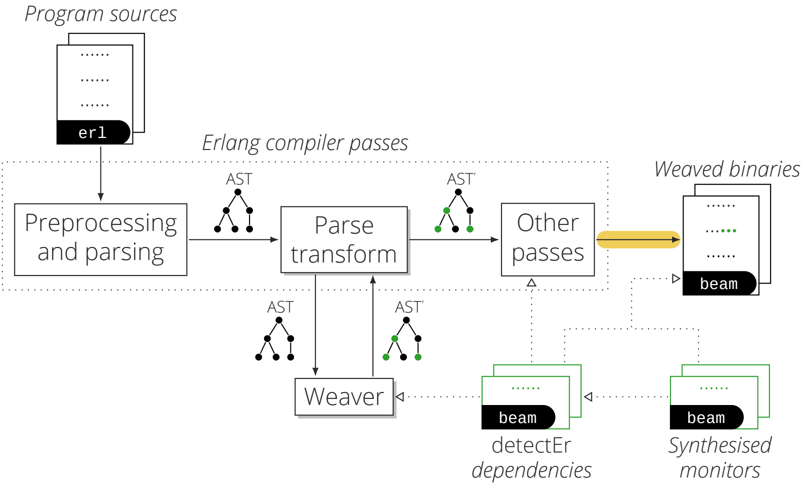
❝ *Explainability = tracking monitor state + applied rules* ❞



Making the theory come alive



4. Inlining monitors



Contributions and summary

An extended monitorable logic and monitors that handle **data**

An algorithm that follows the monitor **operational semantics**

Verdict **explainability** based on monitor reductions

One tool to monitor linear- and branching-time specifications

Future directions and improvements

- Bound on the number of states managed by the algorithm
- Leverage the outline instrumentation provided by detectEr
- Empirical study of runtime overhead

GitHub link

https://duncanatt.github.io/detecter

Thank you