A Formally Verified Monitor

for Metric First-Order Temporal Logic

Joshua Schneider, David Basin, Srđan Krstić, and Dmitriy Traytel

Department of Computer Science

ETH zürich

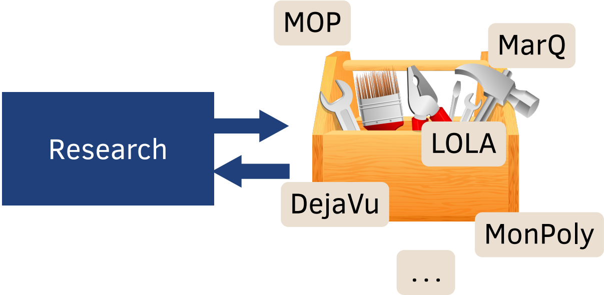
Goals of Runtime Verification

to study whether runtime application of formal methods is a viable **complement to** the traditional methods **proving** programs correct [. . .]

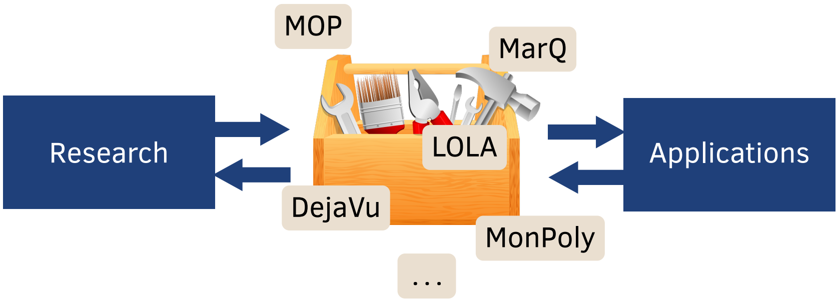
to study whether formality **improves** traditional **ad-hoc monitoring** techniques [. . .]

Source: www.runtime-verification.org (28/08/19, emphasis added)

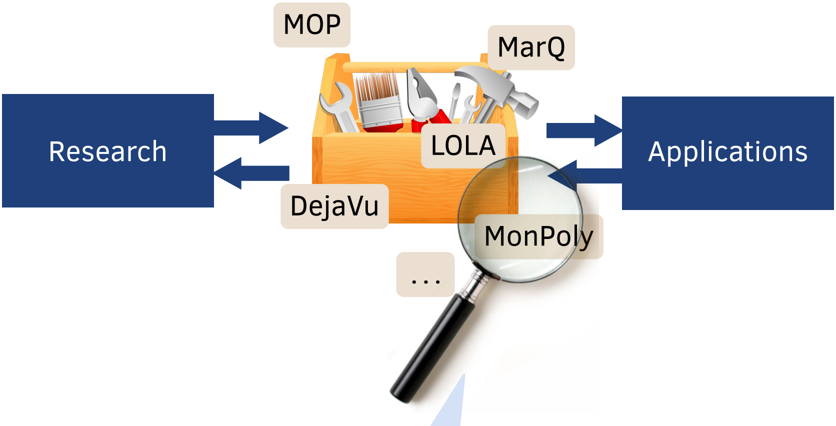
RV Tools



RV Tools



Verifying RV Tools



How can we prove that our tools are trustworthy?
Who guards the guardians?

Why Theorem Proving?

Machine-checked theorem proving is suitable for RV tools:

Criticality

Why Theorem Proving?

Machine-checked theorem proving is suitable for RV tools:

Criticality

Small size

Why Theorem Proving?

Machine-checked theorem proving is suitable for RV tools:

Criticality

Small size

$\models \varphi$

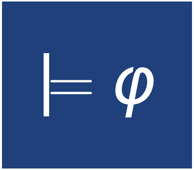
Clear specification

Why Theorem Proving?

Machine-checked theorem proving is suitable for RV tools:

Criticality

Small size



Clear specification

All RV tools and should be verified formally.

Gain understanding of assumptions and guarantees!

Related Work

	Language	Verified with	User effort
Blech et al. (2012)	Regex	Coq	manual proof
Völlinger & Akili (2017)	—	Coq	manual proof

Related Work

	Language	Verified with	User effort
Blech et al. (2012)	Regex	Coq	manual proof
Völlinger & Akili (2017)	—	Coq	manual proof
Laurent et al. (2015)	Copilot	SMT	semi-autom.

Related Work

	Language	Verified with	User effort
Blech et al. (2012)	Regex	Coq	manual proof
Völlinger & Akili (2017)	—	Coq	manual proof
Laurent et al. (2015)	Copilot	SMT	semi-autom.
Rizaldi et al. (2017)	LTL	Isabelle/HOL	none
Bohrer et al. (2018)	d$\mathcal{L}$	KeYmaera X Isabelle/HOL HOL4	none

Related Work

	Language	Verified with	User effort
Blech et al. (2012)	Regex	Coq	manual proof
Völlinger & Akili (2017)	—	Coq	manual proof
Laurent et al. (2015)	Copilot	SMT	semi-autom.
Rizaldi et al. (2017)	LTL	Isabelle/HOL	none
Bohrer et al. (2018)	d$\mathcal{L}$	KeYmaera X Isabelle/HOL HOL4	none
this work	**MFOTL**	Isabelle/HOL	none

Our Contribution

Verimon: **veri**fied **Mon**Poly (w/o optimizations)

- Formally verified monitor for metric first-order temporal logic (MFOTL)
- Expressive language with intervals and data quantification
- Proved correct for all instances of the monitor
- Explain and clarify MonPoly’s algorithm

Our Contribution

Verimon: **veri**fied **Mon**Poly (w/o optimizations)

- Formally verified monitor for metric first-order temporal logic (MFOTL)
- Expressive language with intervals and data quantification
- Proved correct for all instances of the monitor
- Explain and clarify MonPoly's algorithm

Basis for exploration:

- Monitor state manipulation [ATVA'19]
- Foundation for future extensions and optimizations

Our Contribution

Verimon: **veri**fied **Mon**Poly (w/o optimizations)

- Formally verified monitor for metric first-order temporal logic (MFOTL)
- Expressive language with intervals and data quantification
- Proved correct for all instances of the monitor
- Explain and clarify MonPoly's algorithm

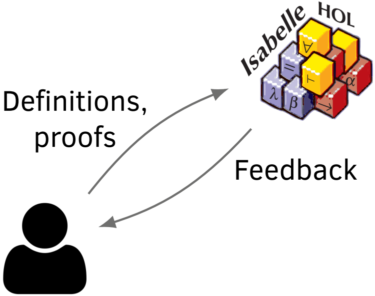
Basis for exploration:

- Monitor state manipulation [ATVA'19]
- Foundation for future extensions and optimizations

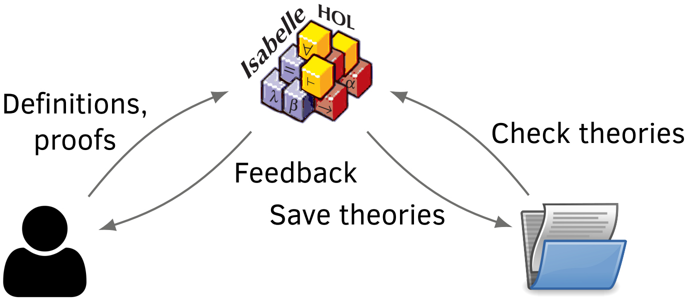
Differential testing case study:

- Used Verimon as oracle to test unverified implementations
- Tested MonPoly and DejaVu
- Found bugs!

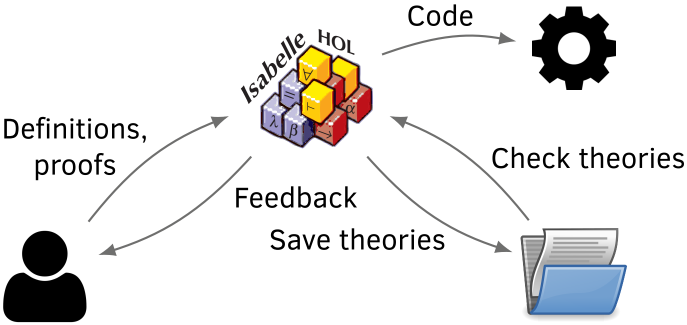
Background: Isabelle/HOL



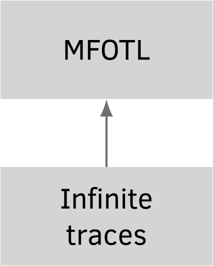
Background: Isabelle/HOL



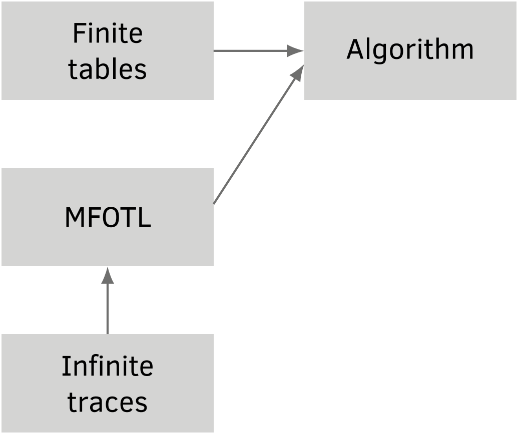
Background: Isabelle/HOL



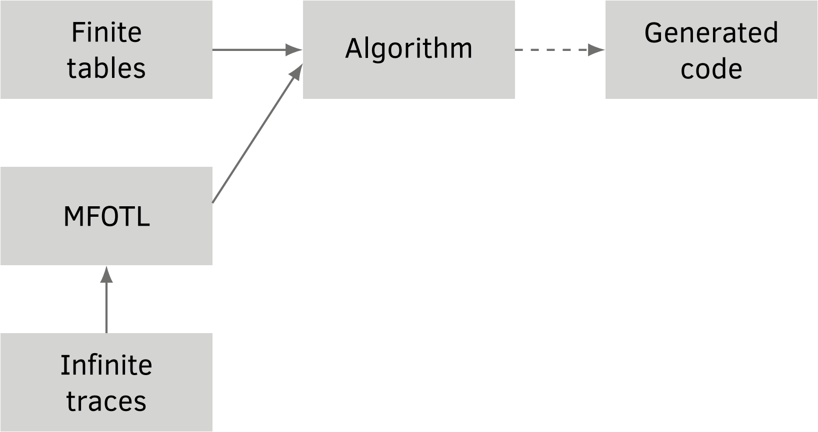
Formalization Overview



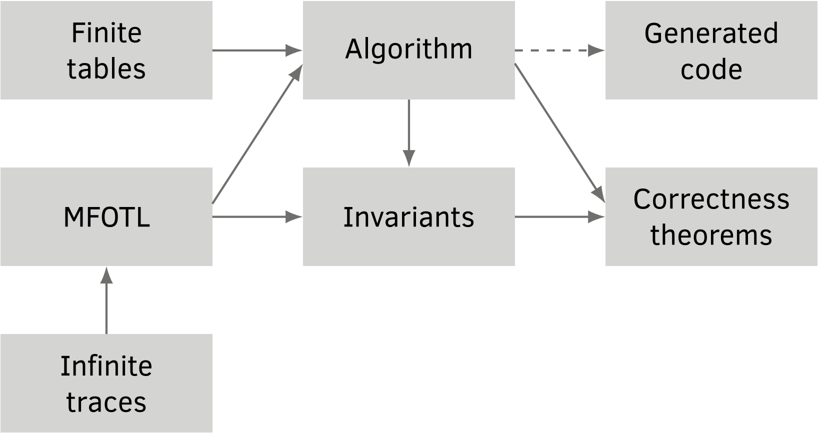
Formalization Overview



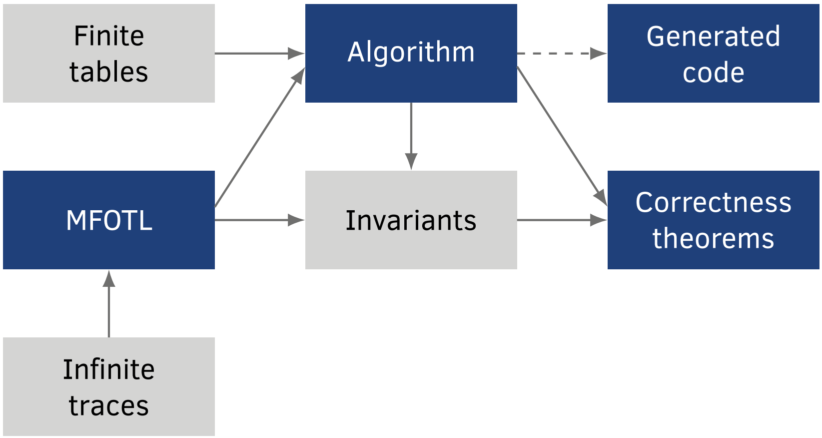
Formalization Overview



Formalization Overview



Formalization Overview



Background: MFOTL

TL

Temporal Logic: linear time ● ○ ⧫ ◊ ■ □ S U

□(access → (¬release S acquire))

Background: MFOTL

FO **First-Order**: data and quantification

$\Box\,\forall x.\ \mathrm{access}(x) \rightarrow (\neg\mathrm{release}(x)\ \mathsf{S}\ \mathrm{acquire}(x))$

TL **Temporal Logic**: linear time ● ○ ⧫ ◊ ■ □ S U

$\Box(\mathrm{access} \rightarrow (\neg\mathrm{release}\ \mathsf{S}\ \mathrm{acquire}))$

Background: MFOTL

M

Metric: time intervals

$\Box \forall x.\ \mathrm{access}(x) \rightarrow (\neg \mathrm{release}(x)\ \mathsf{S}_{[0,1s]}\ \mathrm{acquire}(x))$

FO

First-Order: data and quantification

$\Box \forall x.\ \mathrm{access}(x) \rightarrow (\neg \mathrm{release}(x)\ \mathsf{S}\ \mathrm{acquire}(x))$

TL

Temporal Logic: linear time ● ○ ⧫ ◊ ■ □ S U

$\Box (\mathrm{access} \rightarrow (\neg \mathrm{release}\ \mathsf{S}\ \mathrm{acquire}))$

Background: MFOTL

M **Metric**: time intervals

Monitorable fragment:

- Safety properties ($\square\varphi$ with bounded future)
  NOT: $\square(\mathrm{open} \rightarrow \lozenge\,\mathrm{close})$
- Finitely evaluable violations
  NOT: $\square \forall x.\ \mathrm{P}(x)$

F

T

$\square(\mathrm{access} \rightarrow (\neg\mathrm{release}\ \mathrm{S}\ \mathrm{acquire}))$

Monitor Output

Checking the specification $\square\, \forall \bar{x}.\ \varphi(\bar{x})$:

$$\text{output whether } \models_\sigma \square\, \forall \bar{x}.\ \varphi(\bar{x})$$

Monitor Output

Checking the specification $\square \forall \bar{x}.\ \varphi(\bar{x})$:

$$\text{output whether } \models_\sigma \square \forall \bar{x}.\ \varphi(\bar{x})$$

Reporting violating points of $\forall \bar{x}.\ \varphi(\bar{x})$:

$$\text{output all } i \text{ s.t. } i \not\models_\sigma \forall \bar{x}.\ \varphi(\bar{x})$$

Monitor Output

Checking the specification $\square \forall \bar{x}.\ \varphi(\bar{x})$:

$$\text{output whether } \models_\sigma \square \forall \bar{x}.\ \varphi(\bar{x})$$

Reporting violating points of $\forall \bar{x}.\ \varphi(\bar{x})$:

$$\text{output all } i \text{ s.t. } i \not\models_\sigma \forall \bar{x}.\ \varphi(\bar{x})$$

$\Longleftrightarrow$ Reporting satisfying points of the negation $\exists \bar{x}.\ \neg\varphi(\bar{x})$:

$$\text{output all } i \text{ s.t. } i \models_\sigma \exists \bar{x}.\ \neg\varphi(\bar{x})$$

Monitor Output

Checking the specification $\Box\, \forall \bar{x}.\ \varphi(\bar{x})$:

$$\text{output whether } \models_\sigma \Box\, \forall \bar{x}.\ \varphi(\bar{x})$$

Reporting violating points of $\forall \bar{x}.\ \varphi(\bar{x})$:

$$\text{output all } i \text{ s.t. } i \not\models_\sigma \forall \bar{x}.\ \varphi(\bar{x})$$

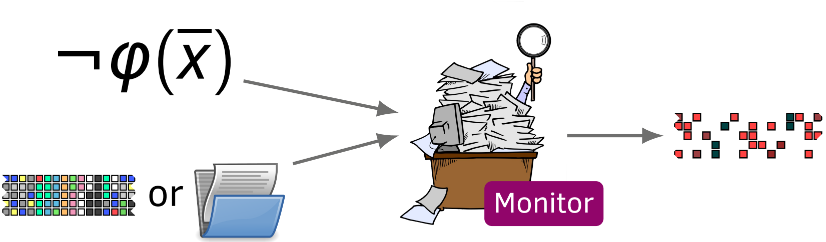
$\Longleftrightarrow$ Reporting satisfying points of the negation $\exists \bar{x}.\ \neg\varphi(\bar{x})$:

$$\text{output all } i \text{ s.t. } i \models_\sigma \exists \bar{x}.\ \neg\varphi(\bar{x})$$

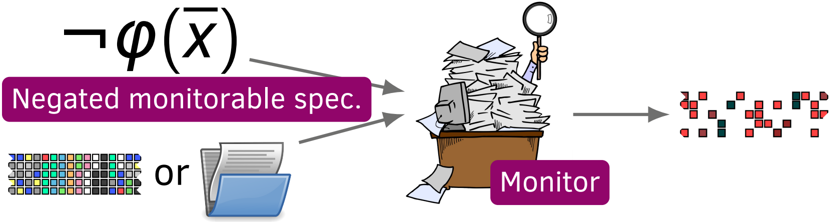
Reporting satisfying points and assignments of $\neg\varphi(\bar{x})$:

$$\text{output all } (i, \bar{x}) \text{ s.t. } i, \bar{x} \models_\sigma \neg\varphi(\bar{x})$$

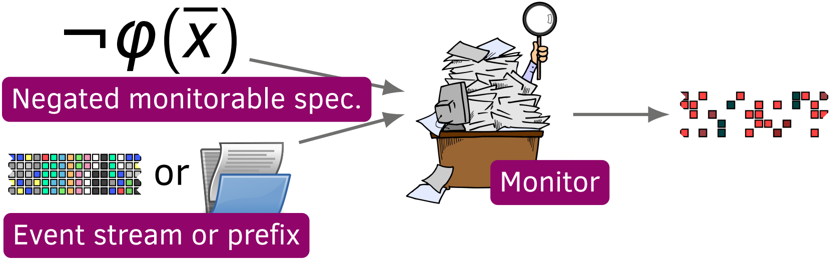
Monitor Interface



Monitor Interface



Monitor Interface



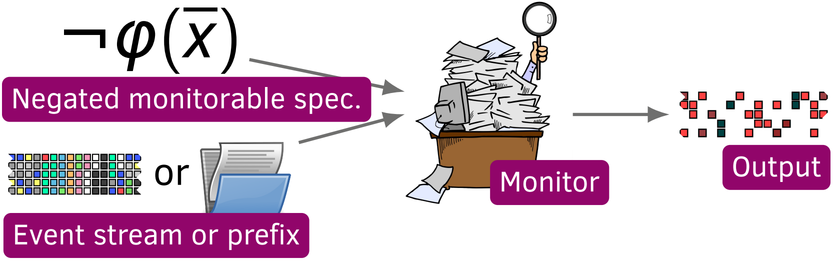
type *event* $=$ *string* $\times$ *domain list*

type *database* $=$ *event set*

type *ts* $=$ *nat*

type *prefix* $= \{p :: (\mathit{database} \times \mathit{ts})\ \mathit{list}.\ \mathrm{sorted}\ (\mathrm{map\ snd}\ p)\}$

Monitor Interface



type *output* $=$ (*nat* $\times$ *tuple*) *set*

all pairs $(i, \overline{x})$ such that $i, \overline{x} \models_\sigma \neg\varphi(\overline{x})$

type *event* $=$ *string* $\times$ *domain list*

type *database* $=$ *event set*

type *ts* $=$ *nat*

type *prefix* $=$ $\{p :: (\mathit{database} \times \mathit{ts})\ \mathit{list}.\ \text{sorted}\ (\text{map snd}\ p)\}$

Specification

Define the expected output of the monitor algorithm:

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**
  spec φ π $=$ $\{(i, t).$ wf_tuple φ $t \wedge$
    $(\forall \sigma.$ prefix_of π $\sigma \longrightarrow i <$ progress σ φ (len π) $\wedge$ sat σ t i $\varphi)\}$

Specification

Define the expected output of the monitor algorithm:

t is assignment to free variables

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**

spec $\varphi\ \pi = \{(i, t).\ \text{wf\_tuple}\ \varphi\ t \wedge$

$(\forall\sigma.\ \text{prefix\_of}\ \pi\ \sigma \longrightarrow i < \text{progress}\ \sigma\ \varphi\ (\text{len}\ \pi) \wedge \text{sat}\ \sigma\ t\ i\ \varphi)\}$

Specification

Define the expected output of the monitor algorithm:

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**

t is assignment to free variables

$\quad \text{spec}\ \varphi\ \pi = \{(i, t).\ \text{wf\_tuple}\ \varphi\ t \wedge$

$\quad\quad (\forall \sigma.\ \text{prefix\_of}\ \pi\ \sigma \longrightarrow i < \text{progress}\ \sigma\ \varphi\ (\text{len}\ \pi) \wedge \text{sat}\ \sigma\ t\ i\ \varphi)\}$

all infinite extensions σ of π

Specification

Define the expected output of the monitor algorithm:

t is assignment to free variables

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**

$\quad$ spec $\varphi\ \pi = \{(i, t).\ \text{wf\_tuple}\ \varphi\ t \wedge$

$\qquad (\forall \sigma.\ \text{prefix\_of}\ \pi\ \sigma \longrightarrow i < \text{progress}\ \sigma\ \varphi\ (\text{len}\ \pi) \wedge \text{sat}\ \sigma\ t\ i\ \varphi)\}$

all infinite extensions σ of π

MFOTL semantics

Specification

Define the expected output of the monitor algorithm:

t is assignment to free variables

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**

$\quad \text{spec}\ \varphi\ \pi = \{(i, t).\ \text{wf\_tuple}\ \varphi\ t \wedge$

$\qquad (\forall \sigma.\ \text{prefix\_of}\ \pi\ \sigma \longrightarrow i < \text{progress}\ \sigma\ \varphi\ (\text{len}\ \pi) \wedge \text{sat}\ \sigma\ t\ i\ \varphi)\}$

all infinite extensions σ of π

MFOTL semantics

delay verdicts for formulas with future modalities

Specification

Define the expected output of the monitor algorithm:

t is assignment to free variables

definition spec :: formula ⇒ prefix ⇒ output **where**

  spec φ π = {(i, t). wf_tuple φ t ∧

   (∀σ. prefix_of π σ ⟶ i < progress σ φ (len π) ∧ sat σ t i φ)}

all infinite extensions σ of π

MFOTL semantics

delay verdicts for formulas with future modalities

$\varphi \equiv P \land \Diamond_{[0,10s]}\, Q$

Specification

Define the expected output of the monitor algorithm:

t is assignment to free variables

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**

spec φ π = $\{(i, t).$ wf_tuple φ t $\wedge$

($\forall\sigma.$ prefix_of π σ $\rightarrow$ i < progress σ φ (len π) $\wedge$ sat σ t i φ)$\}$

all infinite extensions σ of π

MFOTL semantics

delay verdicts for formulas with future modalities

$\varphi \equiv P \wedge \Diamond_{[0,10s]}\, Q$

Specification

Define the expected output of the monitor algorithm:

t is assignment to free variables

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**

  spec $\varphi\ \pi = \{(i, t).\ \text{wf\_tuple}\ \varphi\ t \land$

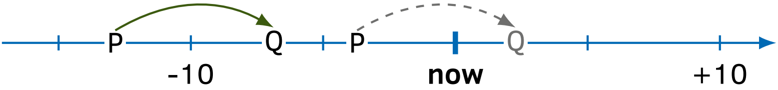
  $(\forall\sigma.\ \text{prefix\_of}\ \pi\ \sigma \longrightarrow i < \text{progress}\ \sigma\ \varphi\ (\text{len}\ \pi) \land \text{sat}\ \sigma\ t\ i\ \varphi)\}$

all infinite extensions σ of π

MFOTL semantics

delay verdicts for formulas with future modalities

$\varphi \equiv P \land \Diamond_{[0,10s]}\ Q$



Specification

Define the expected output of the monitor algorithm:

definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**

$\quad$ spec $\varphi\ \pi = \{(i, t).$ wf_tuple $\varphi\ t \wedge$

$\quad\quad (\forall \sigma.$ prefix_of $\pi\ \sigma \rightarrow i <$ progress $\sigma\ \varphi$ (len π) $\wedge$ sat $\sigma\ t\ i\ \varphi)\}$

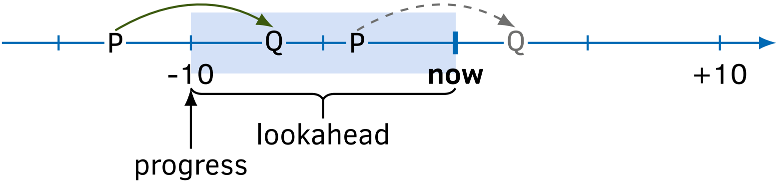
t is assignment to free variables

all infinite extensions σ of π

MFOTL semantics

delay verdicts for formulas with future modalities

$\varphi \equiv P \wedge \Diamond_{[0,10s]}\ Q$



Specification

Define the expected output of the monitor algorithm:

t is assignment to free variables

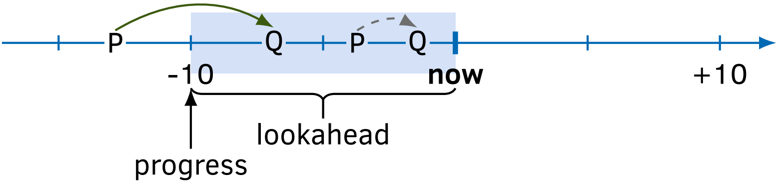
definition spec :: *formula* ⇒ *prefix* ⇒ *output* **where**
  spec $\varphi\ \pi = \{(i, t).\ \text{wf\_tuple}\ \varphi\ t \land$
   $(\forall \sigma.\ \text{prefix\_of}\ \pi\ \sigma \rightarrow i < \text{progress}\ \sigma\ \varphi\ (\text{len}\ \pi) \land \text{sat}\ \sigma\ t\ i\ \varphi)\}$

all infinite extensions σ of π

MFOTL semantics

delay verdicts for formulas with future modalities

$\varphi \equiv P \land \Diamond_{[0,10s]}\, Q$



Correctness (1)

Does the spec function characterize a reasonable monitor?

Correctness (1)

Does the spec function characterize a reasonable monitor?

Fix an event stream σ and a prefix π (i.e., prefix_of π σ is true).

Soundness:

$(i, t) \in$ spec φ π implies sat σ t i φ.

Correctness (1)

Does the spec function characterize a reasonable monitor?

Fix an event stream σ and a prefix π (i.e., prefix_of π σ is true).

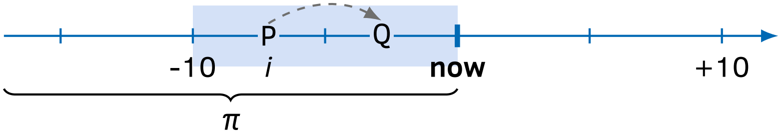
Soundness:

$(i, t) \in$ spec φ π implies sat σ t i φ.

Eventual completeness:

If $i <$ len π and wf_tuple φ t and $\forall \sigma'$. prefix_of π σ' $\rightarrow$ sat σ' t i φ, then there exists a prefix π' of σ such that $(i, t) \in$ spec φ π'.

$\varphi \equiv P \wedge \Diamond_{[0,10s]} Q$



Correctness (1)

Does the spec function characterize a reasonable monitor?

Fix an event stream σ and a prefix π (i.e., prefix_of π σ is true).

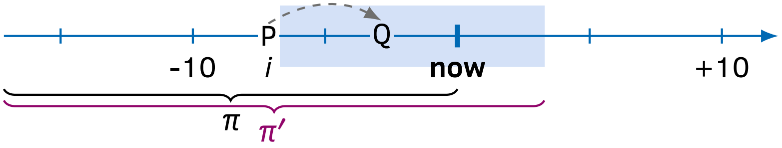
Soundness:

$(i, t) \in$ spec φ π implies sat σ t i φ.

Eventual completeness:

If $i <$ len π and wf_tuple φ t and $\forall \sigma'$. prefix_of π $\sigma' \to$ sat σ' t i φ, then there exists a prefix π' of σ such that $(i, t) \in$ spec φ π'.

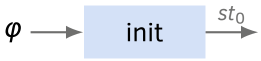
$\varphi \equiv P \wedge \Diamond_{[0,10s]}\, Q$



Implementation

Online interface (unbounded stream):

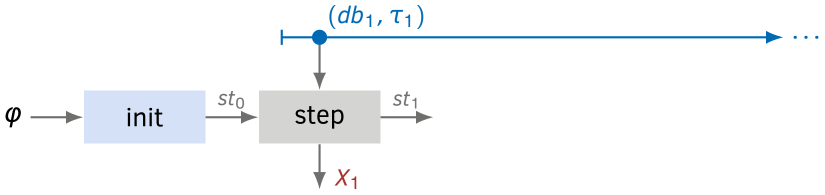
definition init :: *formula* ⇒ *state*



Implementation

Online interface (unbounded stream):

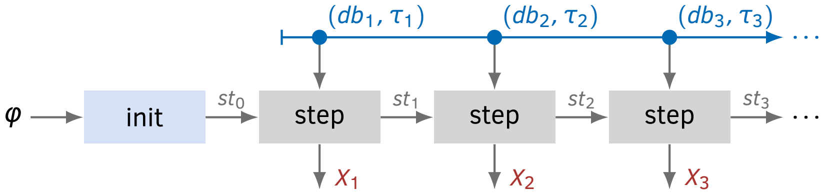
definition init :: *formula* ⇒ *state*
definition step :: *database* × *ts* ⇒ *state* ⇒ *output* × *state*



Implementation

Online interface (unbounded stream):

definition init :: *formula* ⇒ *state*
definition step :: *database* × *ts* ⇒ *state* ⇒ *output* × *state*

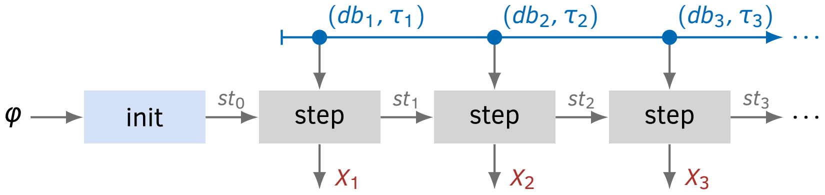


Implementation

Online interface (unbounded stream):

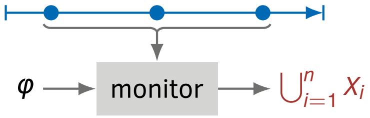
definition init :: *formula* ⇒ *state*

definition step :: *database* × *ts* ⇒ *state* ⇒ *output* × *state*



Offline interface (finite prefix):

definition monitor :: *formula* ⇒ *prefix* ⇒ *output*



Correctness (2)

Is the implementation correct?

1. init establishes the invariant wf_state:

φ is monitorable implies wf_state φ [] (init φ).

Correctness (2)

Is the implementation correct?

1. init establishes the invariant wf_state:

φ is monitorable implies wf_state φ [] (init φ).

2. step preserves the invariant:

Let step (db, τ) $st = (X, st')$. If wf_state φ π st and last_ts $\pi \leq \tau$, then wf_state φ $(\pi @ [(db, \tau)])$ st', $\ldots$

Correctness (2)

Is the implementation correct?

1. init establishes the invariant wf_state:

   φ is monitorable implies wf_state φ [] (init φ).

2. step preserves the invariant:

   Let step (db, τ) $st = (X, st')$. If wf_state φ π st and last_ts $\pi \leq \tau$, then wf_state φ $(\pi \mathbin{@} [(db, \tau)])$ st', …

3. step's output corresponds to spec:

   … and $X =$ spec $\varphi(\pi \mathbin{@} [(db, \tau)]) -$ spec φ π.

Correctness (2)

Is the implementation correct?

1. init establishes the invariant wf_state:

φ is monitorable implies wf_state φ [] (init φ).

2. step preserves the invariant:

Let step (db, τ) $st = (X, st')$. If wf_state φ π st and last_ts $\pi \leq \tau$, then wf_state φ $(\pi \mathbin{@} [(db, \tau)])$ st', …

3. step's output corresponds to spec:

… and $X =$ spec $\varphi(\pi \mathbin{@} [(db, \tau)]) -$ spec φ π.

4. monitor φ $\pi =$ spec φ π (if φ is monitorable)

Executable Monitor

Current approach:

- Extract OCaml code from formalization using Isabelle/HOL's code generator
- Reuse MonPoly's parser/data structures and add "glue"

Executable Monitor

Current approach:

- Extract OCaml code from formalization using Isabelle/HOL's code generator
- Reuse MonPoly's parser/data structures and add "glue"

Trust assumptions:

- Isabelle's kernel and code generator
- Parser and glue code
- OCaml compiler, runtime environment etc.

Executable Monitor

Current approach:

- Extract OCaml code from formalization using Isabelle/HOL's code generator
- Reuse MonPoly's parser/data structures and add "glue"

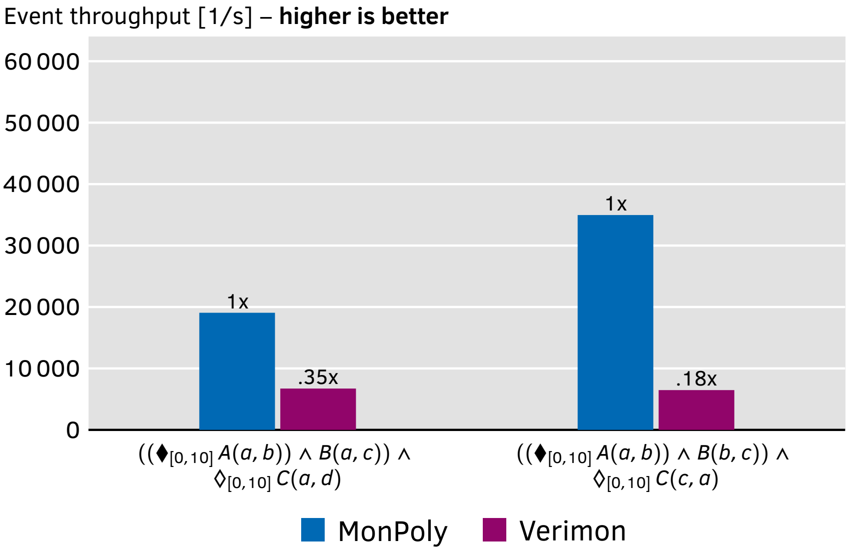
Trust assumptions:

- Isabelle's kernel and code generator
- Parser and glue code
- OCaml compiler, runtime environment etc.

Satisfactory?

- The algorithm is the challenging part
- Various techniques for full-stack verification exist, for example CakeML (used in VeriPhy)

Performance



Differential Testing

Idea: Find bugs in unverified implementations by comparing their output on random inputs with Verimon.

Two targets: **MonPoly** and **DejaVu**

Differential Testing

Idea: Find bugs in unverified implementations by comparing their output on random inputs with Verimon.

Two targets: **MonPoly** and **DejaVu**

- Random formulas parameterized by size n, free variables FV
- Generated 1000 formulas each for $2 \leq n \leq 5$, $|FV| \leq 6$

Differential Testing

Idea: Find bugs in unverified implementations by comparing their output on random inputs with Verimon.

Two targets: **MonPoly** and **DejaVu**

- Random formulas parameterized by size n, free variables FV
- Generated 1000 formulas each for $2 \leq n \leq 5$, $|FV| \leq 6$
- Random prefixes with 20, 40, 60, 100 databases
- Reuse recent event parameters with probability p

Results

Two bugs found in **MonPoly**:

1. Wrong output for class of formulas, for example $Q(x, y) \wedge \neg(P(x) \mathsf{S}\, Q(y, x))$ on prefix with only `Q(1,2)`

Results

Two bugs found in **MonPoly**:

1. Wrong output for class of formulas, for example
   $Q(x, y) \land \neg(P(x) \mathsf{S} Q(y, x))$ on prefix with only `Q(1,2)`

2. Additional violation output for finite traces

```
@0. (time point 0): true
@MaxTS (time point 1): true
```

Results

Two bugs found in **MonPoly**:

1. Wrong output for class of formulas, for example
   $Q(x, y) \land \neg(P(x) \mathsf{S} Q(y, x))$ on prefix with only `Q(1,2)`

2. Additional violation output for finite traces

```
@0. (time point 0): true
@MaxTS (time point 1): true
```

Documented differences in **DejaVu**'s semantics:

3. Arithmetic relations change semantics of quantifiers, e.g.,
   $\neg\varphi$ vs. $\neg\exists x.\ \varphi \land x = 42$

4. Active domain does not include constants in the formula, e.g.,
   $\neg\exists x.\ x = 42 \land \neg P(x)$ on `P(101)`

Ongoing and Future Work

Achieve parity with MonPoly:

- Sliding window algorithm
- Refinement to imperative data structures
- Aggregations (count, sum, max, . . .)

New and verified optimizations:

- Multi-way joins (completed by Thibault Dardinier)

New features:

- State splitting and merging [ATVA'19]
- MFO**D**L – adds regular expressions

A Formally Verified Monitor for MFOTL

	Language	Verified with	User effort
Verimon	MFOTL	Isabelle HOL	none

A Formally Verified Monitor for MFOTL

	Language	Verified with	User effort
Verimon	MFOTL	Isabelle HOL	none

A Formally Verified Monitor for MFOTL

	Language	Verified with	User effort
Verimon	MFOTL	Isabelle HOL	none

Questions?

Joshua Schneider David Basin
Srđan Krstić Dmitriy Traytel

ETH zürich