Automated Timed Temporal Verification for a Mixed Sync-Async Concurrency Paradigm

Yahui Song and Wei-Ngan Chin

School of Computing, NUS

@Computing Research Week 2021

Automated Timed Temporal Verification for a Mixed Sync-Async Concurrency Paradigm

Hiphop.js = Esterel + JS

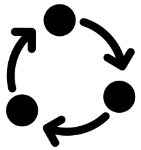
Yahui Song and Wei-Ngan Chin

School of Computing, NUS

@Computing Research Week 2021

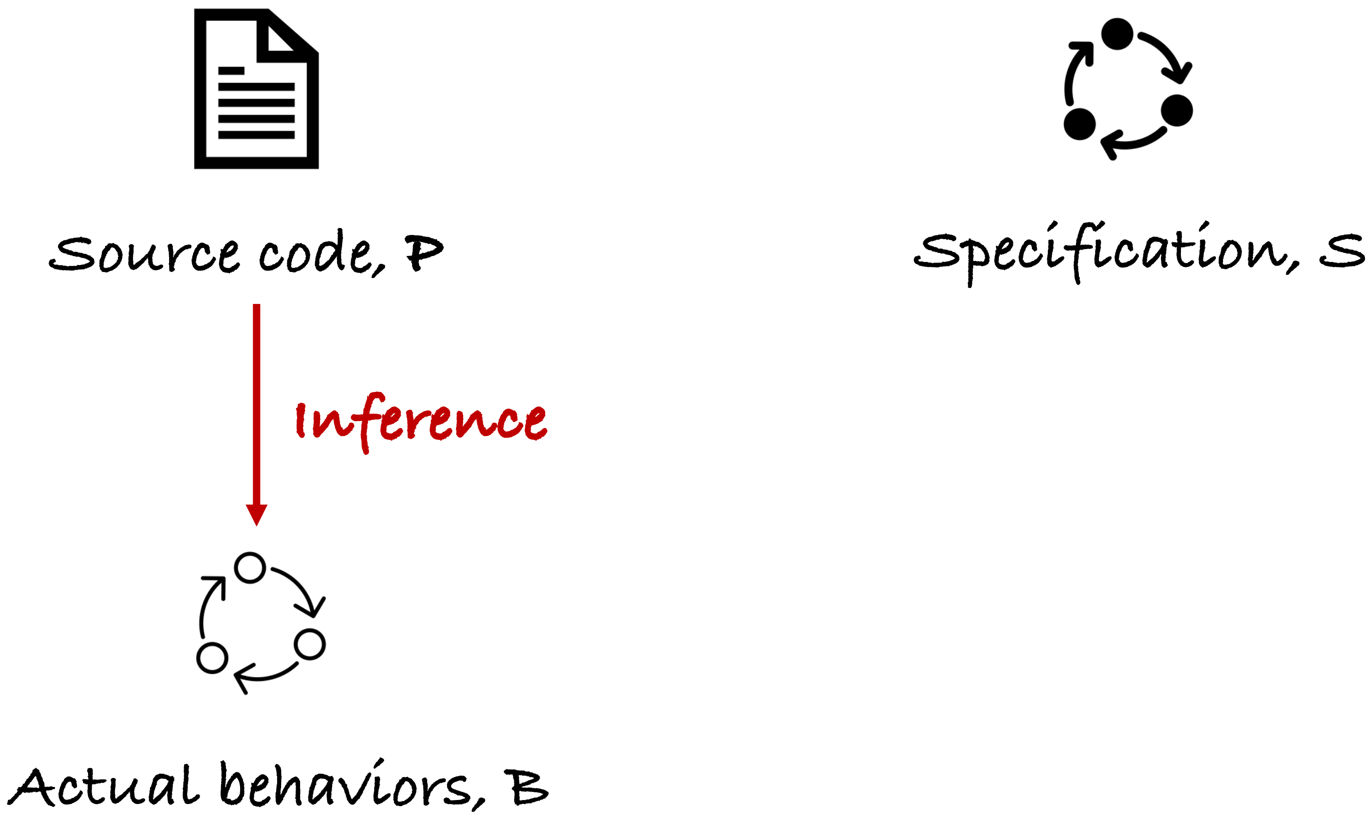
Verification Overview

Source code, P

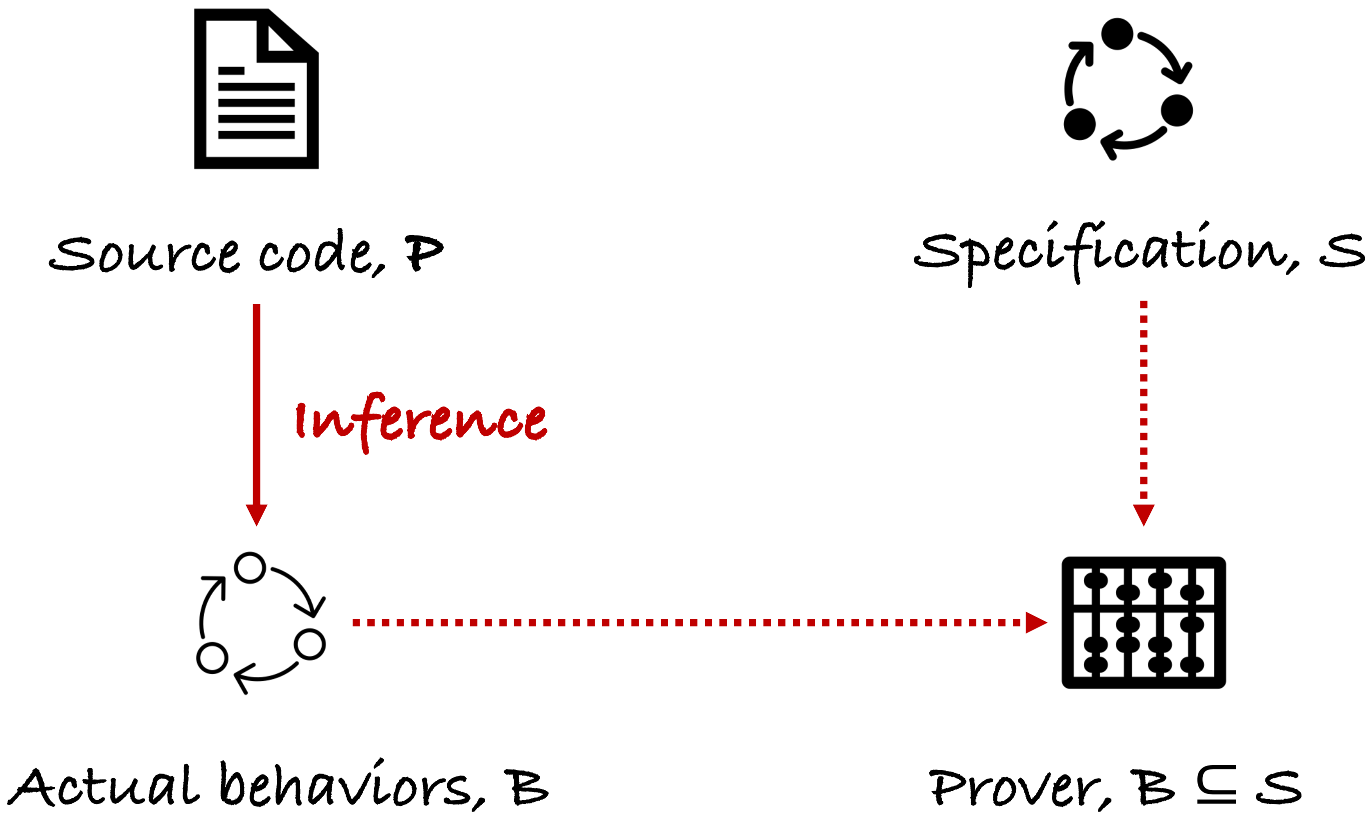


Specification, S

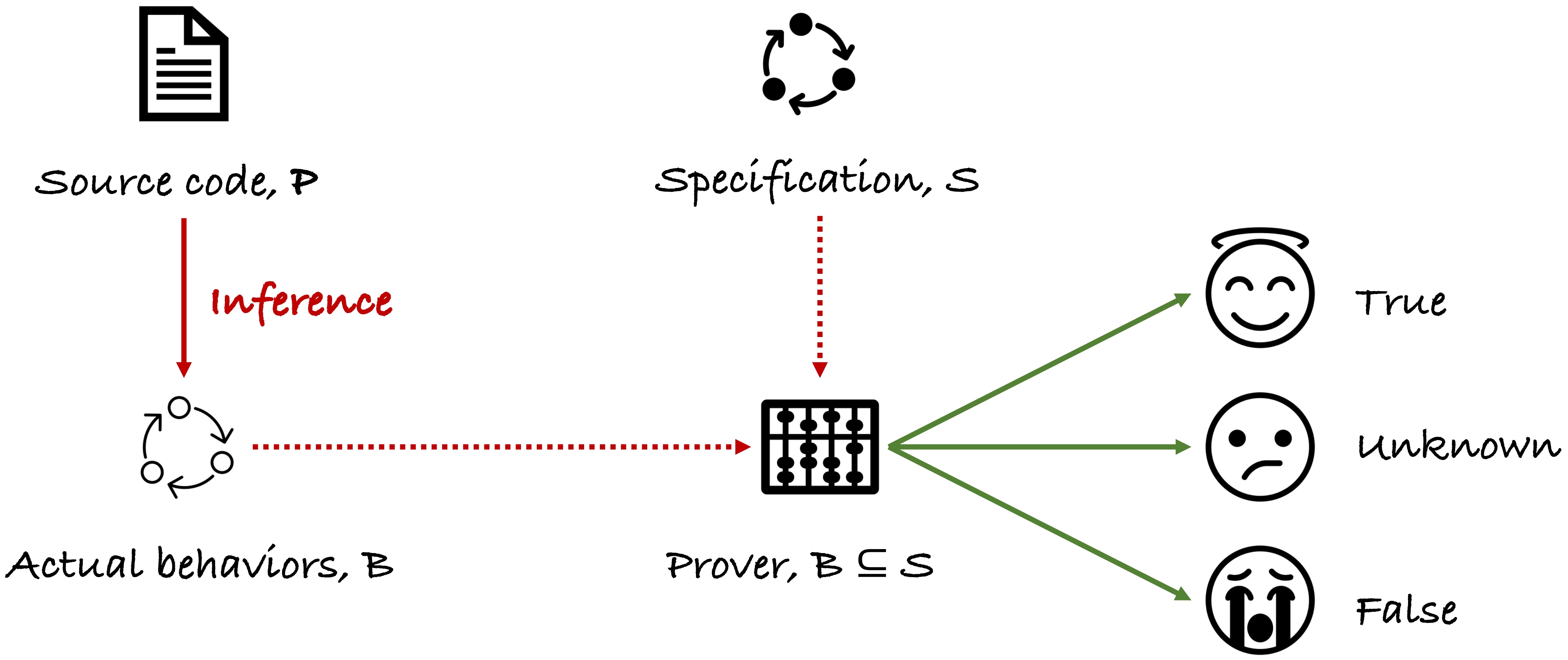
Verification Overview



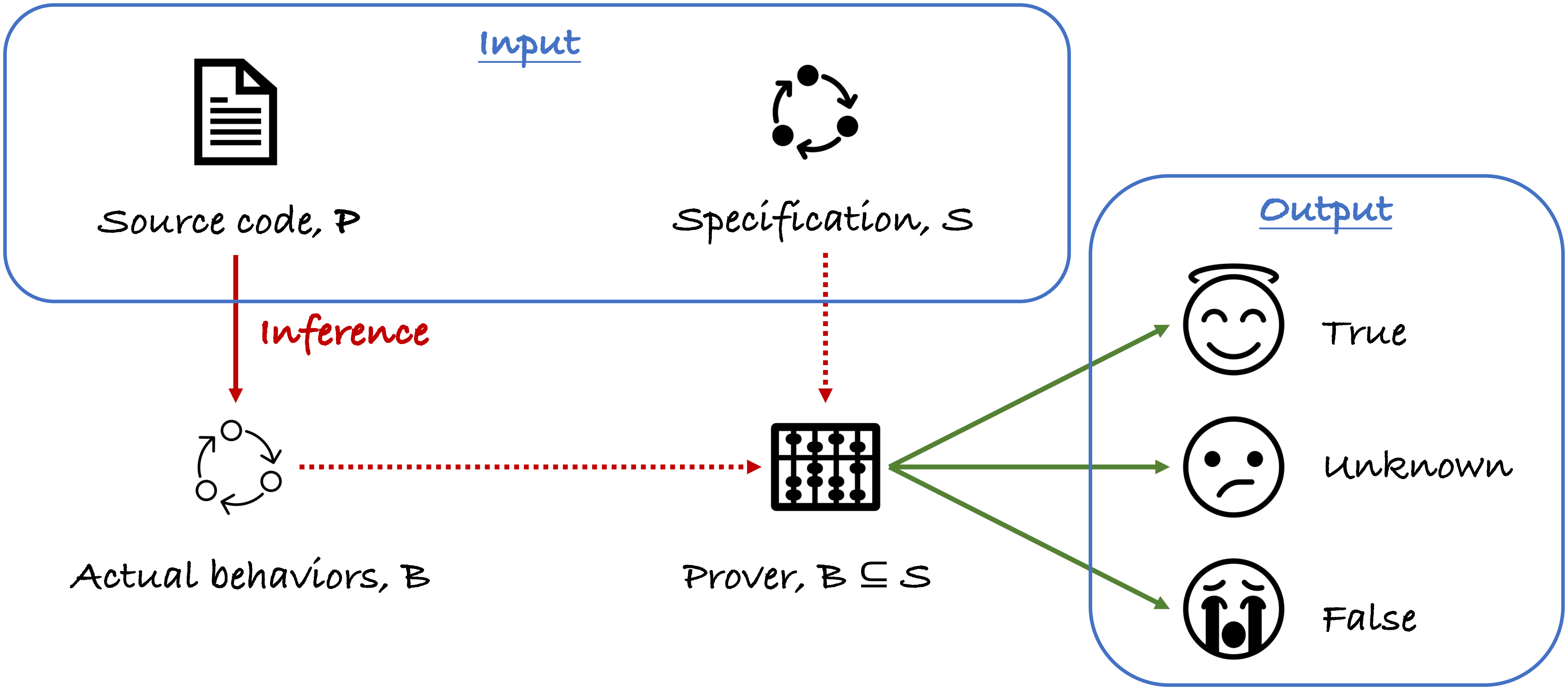
Verification Overview



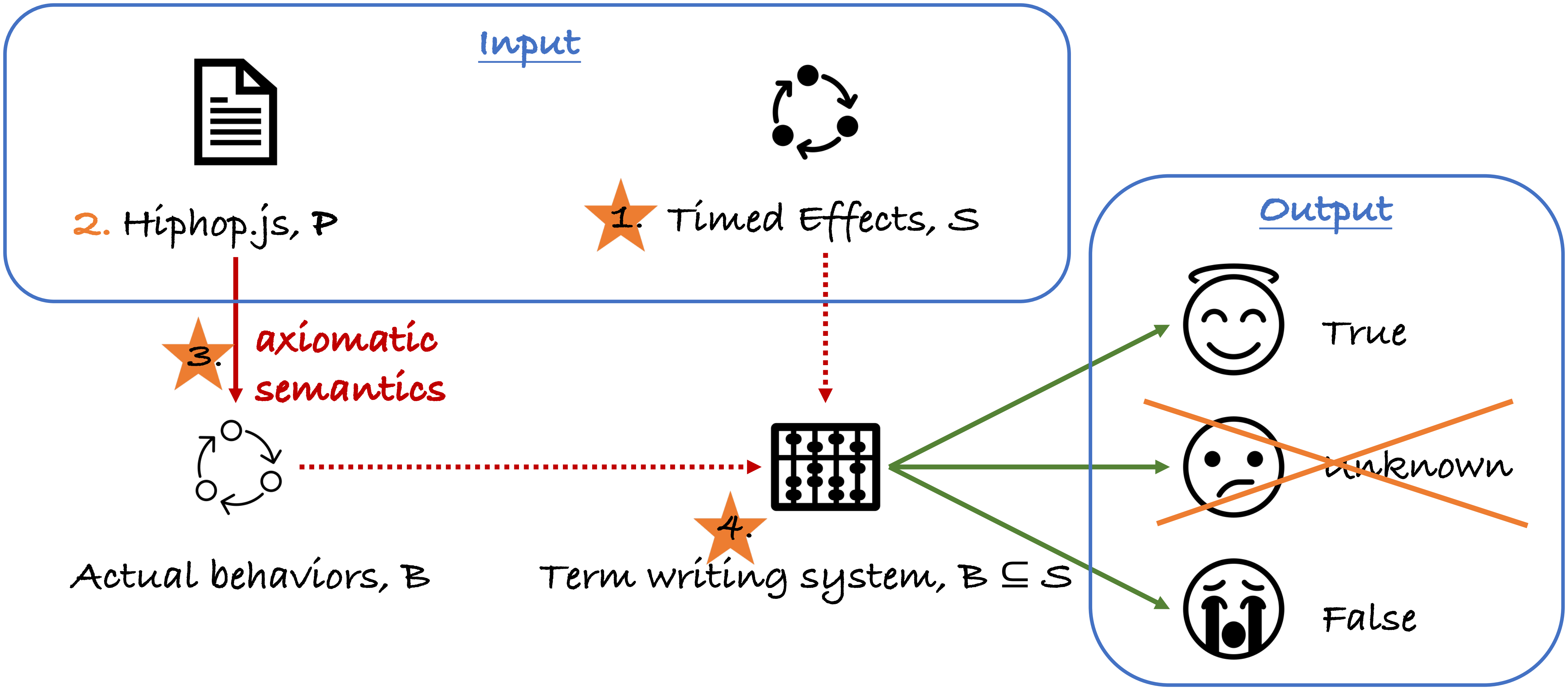
Verification Overview



Automated Verification Overview



Our Work & Contributions



1. Timed Synchronous Effects

"The event will be triggered no later than 1000 milliseconds"

$$\Phi \triangleq 0 \leq t < 1000 : (\{\}^\star \cdot \{\underline{\mathbf{Done}}\})\#t.$$

1. Timed Synchronous Effects

"The event will be triggered no later than 1000 or t milliseconds"

$$\Phi \triangleq 0{\leq}t{<}1000 : (\{\}^{\star} \cdot \{\underline{\mathbf{Done}}\})\#t.$$

$$\Phi^{send(d)} \triangleq (0{<}d{\leq}5000 \wedge 0{\leq}t{<}d) : (\{\underline{\mathbf{Send}}\}\#t) \cdot \{\underline{\mathbf{Done}}\}.$$

1. Timed Synchronous Effects

"The event will be triggered no later than 1000 or t milliseconds"

$$\Phi \triangleq 0{\leq}t{<}1000 : (\{\}^{\star} \cdot \{\underline{\mathbf{Done}}\})\#t.$$

$$\Phi^{send(d)} \triangleq (0{<}d{\leq}5000 \wedge 0{\leq}t{<}d) : (\{\underline{\mathbf{Send}}\}\#t) \cdot \{\underline{\mathbf{Done}}\}.$$

- Extends Synchronous Kleene Algebra with the operator #.
- Defines a set of exact timed transition systems.

2. Computation Models

i. **Transformational programs** compute output values from input values. This is the domain of classical sequential programming languages.

ii. **Asynchronous concurrent programs** perform interactions between their components using typically network-based communication.

iii. **Synchronous reactive programs** react to external events in a conceptually instantaneous and deterministic way.

2. Hiphop.js = Esterel + JS

i. **Transformational programs** compute output values from input values. This is the domain of classical sequential programming languages.

ii. **Asynchronous concurrent programs** perform interactions between their components using typically network-based communication. **(JS)**

iii. **Synchronous reactive programs** react to external events in a conceptually instantaneous and deterministic way. **(Esterel)**

↓

Hiphop.js

2. Hiphop.js = Esterel + JS

"Mixed Sync-Async Concurrency Paradigm"

{A} · {B} · {C} || {W} · {X} · {Y} · {Z} -> {A,W} · {B,X} · {C,Y} · {Z}

{A} · {B} · {C} · {D} || {E} · C? · {F} -> {A, E} · {B} · {**C**} · {D, F}

{A} · {B} · {D} || {E} · C? · {F} -> {A, E} · {B} · {D} · C? · {F}

2. Hiphop.js = Esterel + JS

"Mixed Sync-Async Concurrency Paradigm"

{A} · {B} · {C} || {W} · {X} · {Y} · {Z} -> {A,W} · {B,X} · {C,Y} · {Z}

{A} · {B} · {C} · {D} || {E} · C? · {F} -> {A, E} · {B} · {**C**} · {D, F}

{A} · {B} · {D} || {E} · C? · {F} -> {A, E} · {B} · {D} · C? · {F}

JS: Broken chain promises.

3. Effects Inference

```
hiphop module main (out Prep, in Tick, out Ready, out Go, out Cook)
/*@ requires true : emp @*/
/*@ ensures 0<=t/\t<3000 : {}.({Cook})#t.{}^* @*/
{
    fork{
        await Ready; emit Go;
    }par{
        emit Prep;
        async Ready { run cook (3000, Tick, Cook); }}}


hiphop module cook (var d, in Tick, out Cook)
/*@ requires d>2000 : {}^*.{Prep} @*/
/*@ ensures 0<=t/\t<d : ({}.{Cook})#t @*/
{ abort count(d, Tick) { yield; emit Cook; }}
```

(1) *fork*{ (– *initialize the current effects state using the module precondition* –)

$\langle true : emp \rangle$ (– *emp indicates an empty trace* –)

(2) *await* *Ready*;

$\langle true : Ready? \cdot \{\} \rangle$ [*FV-Await*]

(3) *emit* *Go*;

$\langle true : Ready? \cdot \{Go\} \rangle$ [*FV-Emit*]

Add the events into the effect state

fork{ (– *initialize the current effects state using the module precondition* –)

$\langle true : emp \rangle$ (– *emp indicates an empty trace* –)

await Ready;

$\langle true : Ready? \cdot \{\} \rangle$ [FV-Await]

emit Go;

$\langle true : Ready? \cdot \{Go\} \rangle$ [FV-Emit]

}*par*{ (– *initialize the current effects state using the module precondition* –)

$\langle true : emp \rangle$

emit Prep;

$\langle true : \{Prep\} \rangle$ [FV-Emit]

async Ready{

run cook (3000, Cook)}}}

(-TRS: check the precondition of module cook-)

$d{=}3000 : \{Prep\} \sqsubseteq d{>}2000 \wedge \{Prep\}$

(-TRS: succeed-)

$\langle 0 \leq t < 3000 : (\{Prep\} \cdot \{Cook\})\#t \rangle$ [FV-Call]

Add the events into the effect state

Check if the current effect satisfies the callee's precondition

(1) *fork*{ (– *initialize the current effects state using the module precondition* –)

$\langle true : emp \rangle$ (– *emp indicates an empty trace* –)

(2) *await* *Ready*;

$\langle true : Ready? \cdot \{\} \rangle$ [FV-Await]

(3) *emit* *Go*;

$\langle true : Ready? \cdot \{Go\} \rangle$ [FV-Emit]

(4) }*par*{ (– *initialize the current effects state using the module precondition* –)

$\langle true : emp \rangle$

(5) *emit* *Prep*;

$\langle true : \{Prep\} \rangle$ [FV-Emit]

(6) *async* *Ready*{

(7) *run* *cook* (3000, *Cook*)}}}

(-TRS: check the precondition of module cook-)

$d{=}3000 : \{Prep\} \sqsubseteq d{>}2000 \wedge \{Prep\}$

(-TRS: succeed-)

$\langle 0{\leq}t{<}3000 : (\{Prep\} \cdot \{Cook\})\#t \rangle$ [FV-Call]

$\langle 0{\leq}t{<}3000 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \rangle$ [FV-Async]

(8) $\langle (true \wedge 0{\leq}t{<}3000) : Ready? \cdot \{Go\} \parallel (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \rangle$ [FV-Fork]

$\langle 0{\leq}t{<}3000 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \cdot \{Go\} \rangle$ [Effects-Normalization]

(9) (-TRS: check the postcondition of module main; Succeed, cf. Table 1.-)

$0{\leq}t{<}3000 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \cdot \{Go\} \sqsubseteq 0{\leq}t{<}3000 : \{\} \cdot (\{Cook\})\#t \cdot \{\}^{\star}$

Add the events
into the effect state

Check if the current effect
satisfies the callee's precondition

Checks if the final effects satisfy the
Program's postcondition

4. Language Inclusion (TRS)

$$\frac{t_L{<}3 : (\{Prep\} \cdot \{Cook\})\#t_L \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R{<}3 : \{\} \cdot \{Cook\}\#t_R \cdot \{\}^\star}{t{<}3 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t{<}3 : \{\} \cdot \{Cook\}\#t \cdot \{\}^\star} \text{①}[RENAME]$$

4. Language Inclusion (TRS)

$$t_L<3 \wedge t_L^1+t_L^2=t_L : \{Prep\}\#t_L^1 \cdot \{Cook\}\#t_L^2 \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \{\} \cdot \{Cook\}\#t_R \cdot \{\}^{\star}$$

②[SPLIT]

$$t_L<3 : (\{Prep\} \cdot \{Cook\})\#t_L \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \{\} \cdot \{Cook\}\#t_R \cdot \{\}^{\star}$$

①[RENAME]

$$t<3 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t<3 : \{\} \cdot \{Cook\}\#t \cdot \{\}^{\star}$$

4. Language Inclusion (TRS)

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : \cancel{\{Ready\}} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{emp} \vee \cancel{\{\}} \cdot \{\}^\star$$

④[UNFOLD−UNIFY]

$$t_L<3 \wedge t_L^1+t_L^2=t_L : \cancel{\{Cook\}\#t_L^2} \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{\{Cook\}\#t_R} \cdot \{\}^\star$$

③[UNFOLD]

$$t_L<3 \wedge t_L^1+t_L^2=t_L : \cancel{\{Prep\}\#t_L^1} \cdot \{Cook\}\#t_L^2 \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{\{\}} \cdot \{Cook\}\#t_R \cdot \{\}^\star$$

②[SPLIT]

$$t_L<3 : (\{Prep\} \cdot \{Cook\})\#t_L \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \{\} \cdot \{Cook\}\#t_R \cdot \{\}^\star$$

①[RENAME]

$$t<3 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t<3 : \{\} \cdot \{Cook\}\#t \cdot \{\}^\star$$

4. Language Inclusion (TRS)

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R \Rightarrow t_R<3 \qquad emp \sqsubseteq \{\}^\star$$

⑧[PROVE]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : emp \sqsubseteq t_R<3 : \bot \vee \{\}^\star$$

⑦[UNFOLD]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : \cancel{\{Go\}} \sqsubseteq t_R<3 : \cancel{emp} \vee \cancel{\{\}} \cdot \{\}^\star$$

⑥[Normalization]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : \{Go\} \sqsubseteq t_R<3 : \bot \vee \{\}^\star$$

⑤[UNFOLD]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : \cancel{\{Ready\}} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{emp} \vee \cancel{\{\}} \cdot \{\}^\star$$

④[UNFOLD−UNIFY]

$$t_L<3 \wedge t_L^1+t_L^2=t_L : \cancel{\{Cook\}\#t_L^2} \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{\{Cook\}\#t_R} \cdot \{\}^\star$$

③[UNFOLD]

$$t_L<3 \wedge t_L^1+t_L^2=t_L : \cancel{\{Prep\}\#t_L^1} \cdot \{Cook\}\#t_L^2 \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{\{\}} \cdot \{Cook\}\#t_R \cdot \{\}^\star$$

②[SPLIT]

$$t_L<3 : (\{Prep\} \cdot \{Cook\})\#t_L \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \{\} \cdot \{Cook\}\#t_R \cdot \{\}^\star$$

①[RENAME]

$$t<3 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t<3 : \{\} \cdot \{Cook\}\#t \cdot \{\}^\star$$

4. Language Inclusion (TRS)

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R \Rightarrow t_R<3 \qquad emp \sqsubseteq \{\}^\star$$

⑧[*PROVE*]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : emp \sqsubseteq t_R<3 : \bot \vee \{\}^\star$$

⑦[*UNFOLD*]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : \cancel{\{Go\}} \sqsubseteq t_R<3 : \cancel{emp} \vee \cancel{\{\}} \cdot \{\}^\star$$

⑥[*Normalization*]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : \{Go\} \sqsubseteq t_R<3 : \bot \vee \{\}^\star$$

⑤[*UNFOLD*]

$$t_L<3 \wedge t_L^1+t_L^2=t_L \wedge t_L^2=t_R : \cancel{\{Ready\}} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{emp} \vee \cancel{\{\}} \cdot \{\}^\star$$

④[*UNFOLD−UNIFY*]

$$t_L<3 \wedge t_L^1+t_L^2=t_L : \cancel{\{Cook\}\#t_L^2} \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{\{Cook\}\#t_R} \cdot \{\}^\star$$

③[*UNFOLD*]

$$t_L<3 \wedge t_L^1+t_L^2=t_L : \cancel{\{Prep\}\#t_L^1} \cdot \{Cook\}\#t_L^2 \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \cancel{\{\}} \cdot \{Cook\}\#t_R \cdot \{\}^\star$$

②[*SPLIT*]

$$t_L<3 : (\{Prep\} \cdot \{Cook\})\#t_L \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t_R<3 : \{\} \cdot \{Cook\}\#t_R \cdot \{\}^\star$$

①[*RENAME*]

$$t<3 : (\{Prep\} \cdot \{Cook\})\#t \cdot \{Ready\} \cdot \{Go\} \sqsubseteq t<3 : \{\} \cdot \{Cook\}\#t \cdot \{\}^\star$$

Implementation and Evaluation

- An open-sourced prototype system using Ocaml.
- Benchmarks, 155 programs (10~300 lines) with manually annotated specs:
  1. CEC: It is an open-source compiler which provides Esterel programs for testing.
  2. Hiphop.js: It is a DSL for JavaScript.
- Proven the back-end solver (inclusion checker) sound and complete.

Summary

- **Timed Synchronous Effects (TSE):** goes beyond timed automata;
- **Automated Forward Verifier:** an axiomatic semantics for HipHop.js;
- **An Efficient Term Rewriting System (TRS):** the back-end prover for TSE language inclusions, proven sound and complete;
- Implementation and Evaluation;

Thanks a lot for your attention!