Automated Temporal Verification with Extended Regular Expressions

PhD Candidate: Yahui Song

Superviser: Prof. Wei Ngan Chin

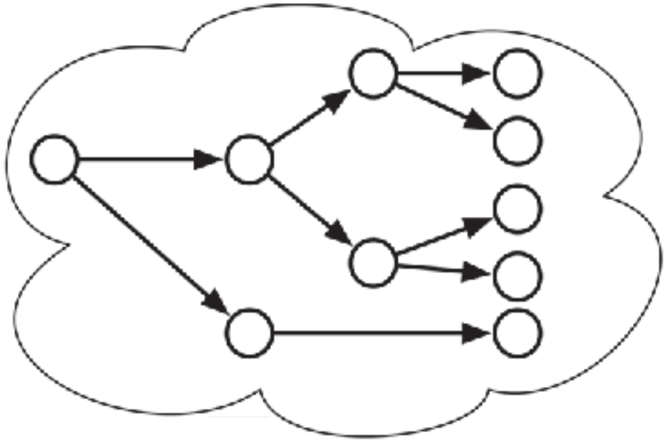
Examiners: Prof. Jin Song Dong and Prof. Joxan Jaffar

HoD Representative: Prof. Siau Cheng Khoo

15th May 2023 @ Soc, NUS

Temporal Verification – Existing Framework

Manuel/CSP
Modelling



Expressiveness is limited
by the finite-state automata

Temporal Logic

Property Φ

MODEL CHECKER

To be bounded
due to the lack of
symbolic proving

A verified model

≈

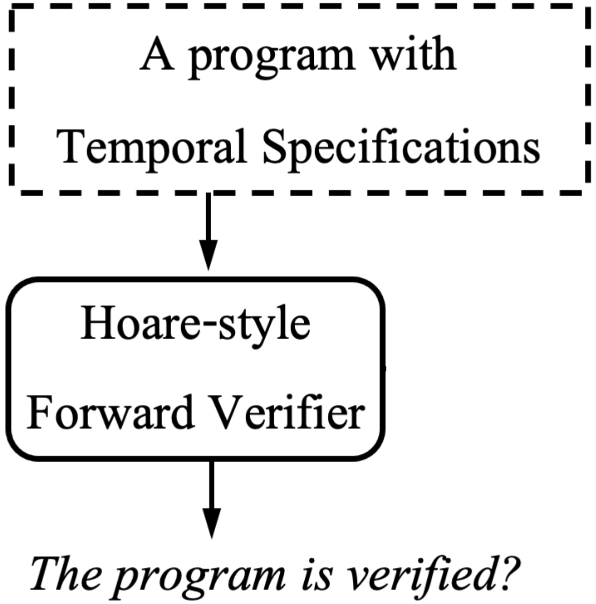
A verified implementation

Yes, Property Φ is true

No, Property Φ is not true

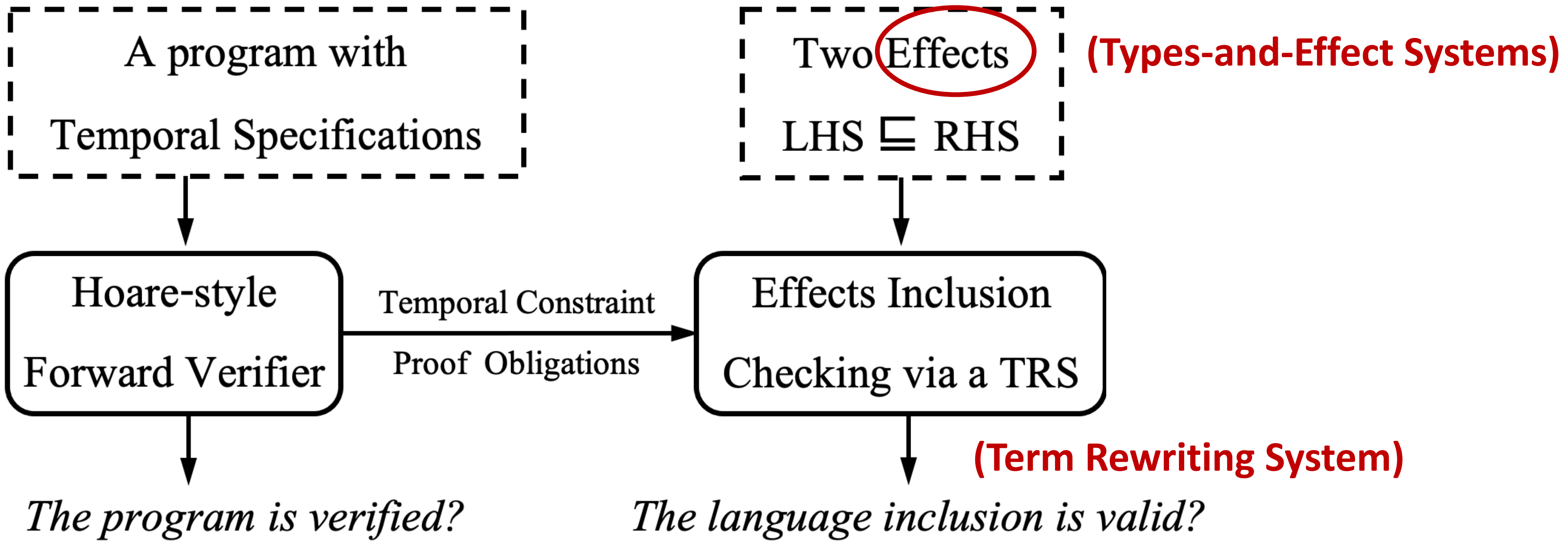
& counterexamples

A New Framework for Temporal Verification



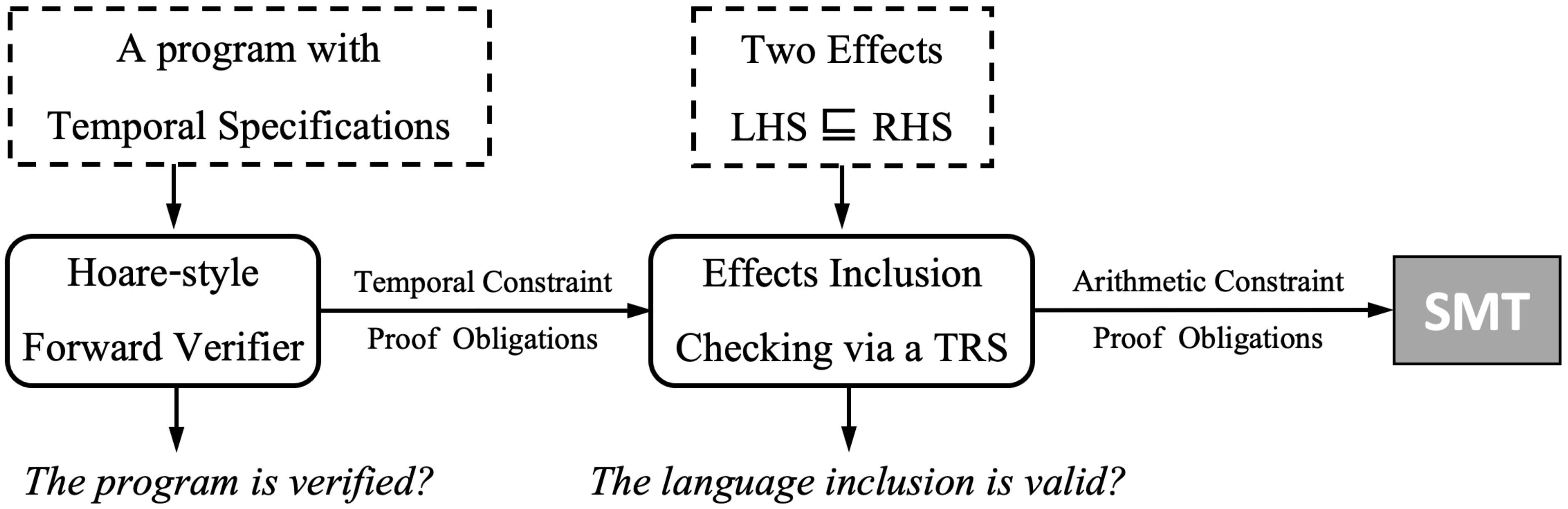
+ A verified implementation;
+ Flexible specifications, which an be combined with other logic;

A New Framework for Temporal Verification



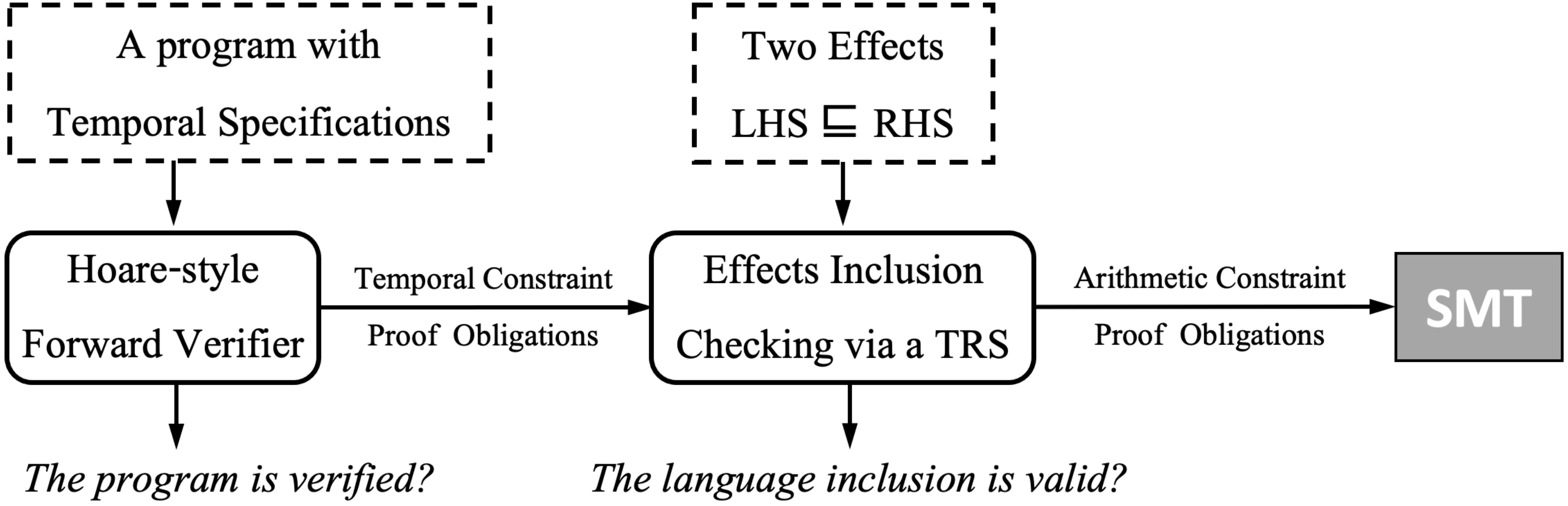
+ A verified implementation;
+ Flexible specifications, which an be combined with other logic;
+ Efficient symbolic entailment checker with (co-)inductive proofs;

A New Framework for Temporal Verification



+ A verified implementation;
+ Flexible specifications, which an be combined with other logic;
+ Efficient symbolic entailment checker with (co-)inductive proofs;
- Automation/Decidability.

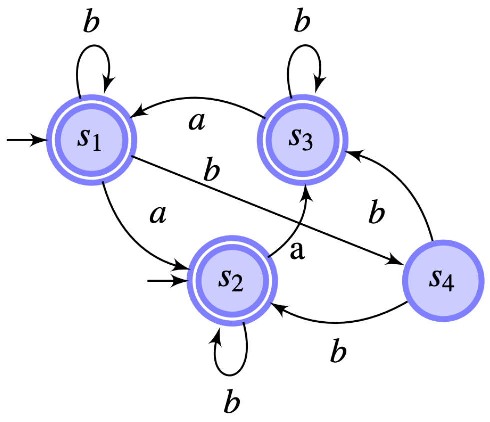
A New Framework for Temporal Verification



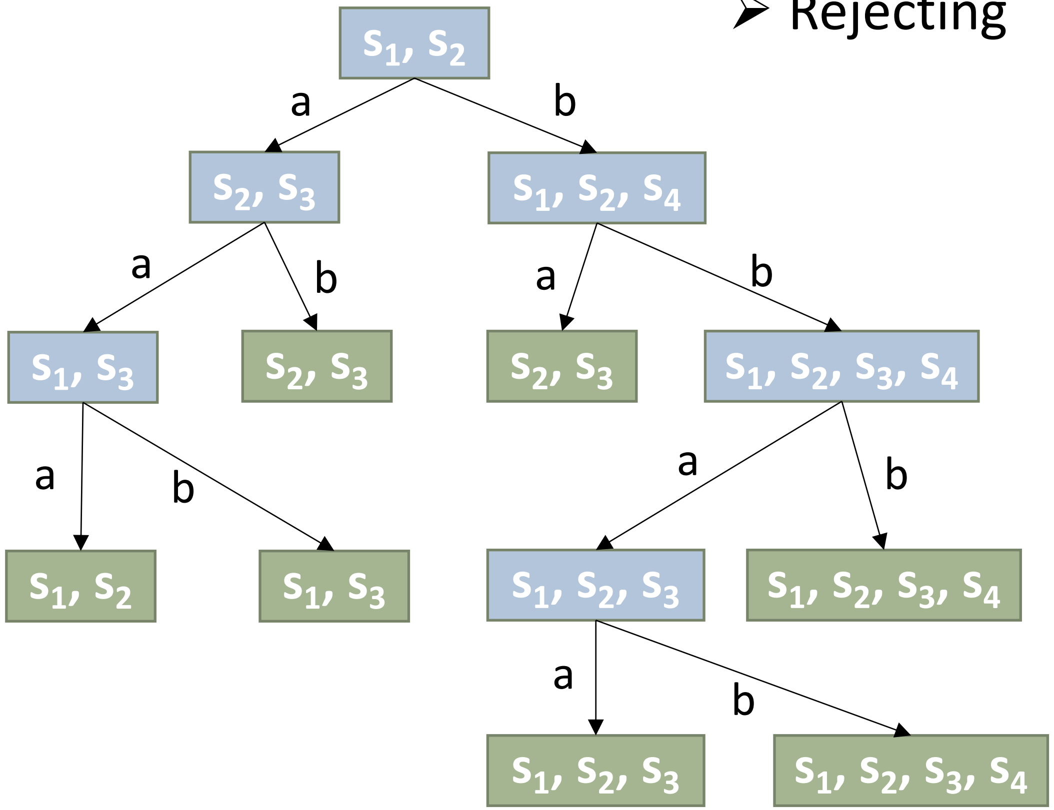
+ A verified implementation;
+ **Flexible specifications**, which an be combined with other logic;
+ **Efficient** symbolic entailment checker with (co-)inductive proofs;
- Automation/Decidability.

Automata vs. RE : Σ*⊑ L(A)

Flexibility and Efficiency

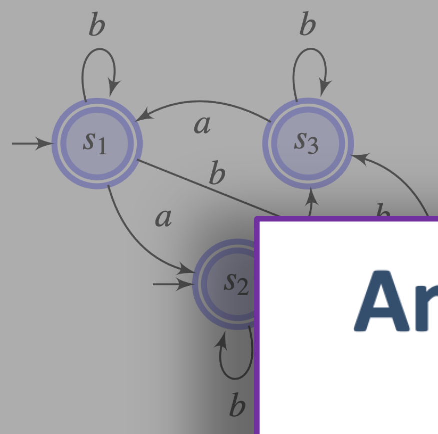


- Init/Next
- Processed
- Rejecting



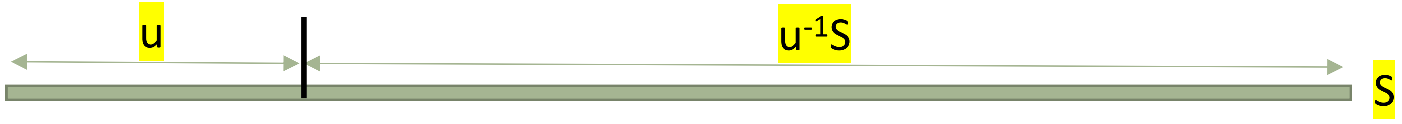
Automata vs. RE : $\Sigma^* \sqsubseteq L(A)$

Flexibility and Efficiency



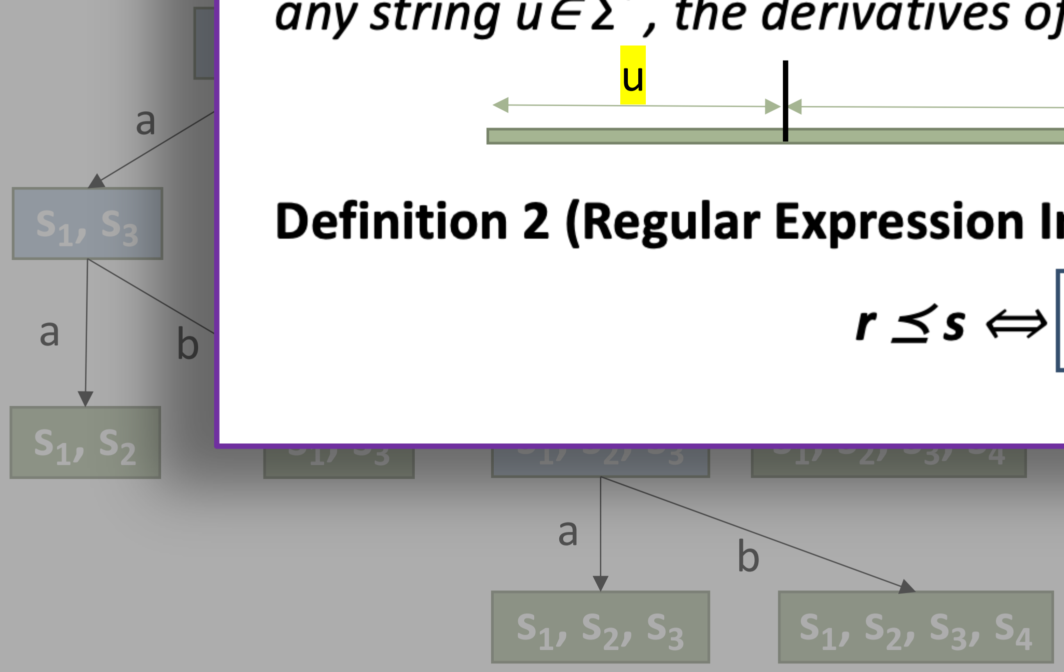
Antimirov algorithm for solving REs' inclusions

Definition 1 (Derivatives). *Given any formal language S over an alphabet Σ and any string $u \in \Sigma^*$, the derivatives of S w.r.t u is defined as: $u^{-1}S = \{w \in \Sigma^* \mid uw \in S\}$.*



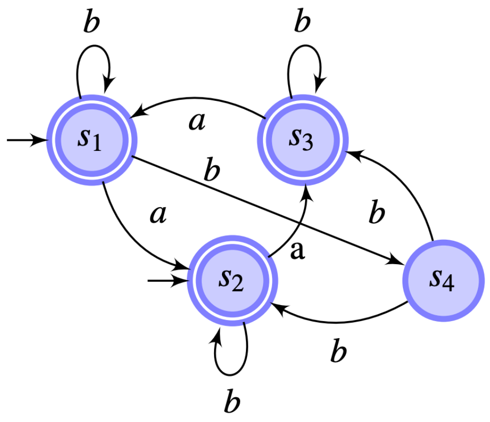
Definition 2 (Regular Expression Inclusion). ***For REs r and s,***

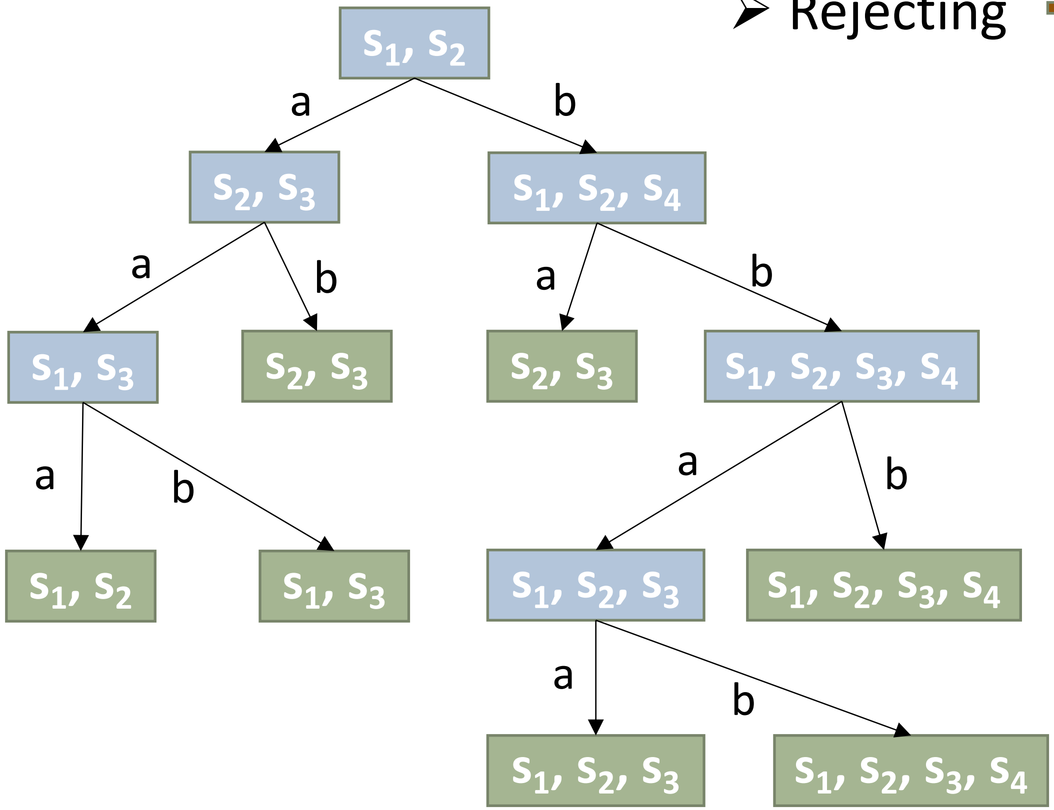
$$r \preceq s \Leftrightarrow \boxed{\forall A \in \Sigma.}\ A^{-1}(r) \preceq A^{-1}(s)\ .$$



Automata vs. RE : $\Sigma^* \sqsubseteq L(A)$

Flexibility and Efficiency





- Init/Next → First/Derivatives
- Processed → Proof Context
- Rejecting → Null-able/Infinite-able

$(a \vee b)^\star \sqsubseteq (a \vee b \vee bb)^\star$ [Reoccur]

$\varepsilon \cdot (a \vee b)^\star \sqsubseteq \varepsilon \cdot (a \vee b \vee bb)^\star$ [Reoccur]

$a \cdot (a \vee b)^\star \sqsubseteq (a \vee b \vee bb)^\star$ $b \cdot (a \vee b)^\star \sqsubseteq$ ...

$$\frac{\dfrac{\dfrac{(a \vee b)^\star \sqsubseteq (a \vee b \vee bb)^\star \ \text{[Reoccur]}}{\varepsilon \cdot (a \vee b)^\star \sqsubseteq \varepsilon \cdot (a \vee b \vee bb)^\star} \quad \dfrac{\text{[Reoccur]}}{}}{a \cdot (a \vee b)^\star \sqsubseteq (a \vee b \vee bb)^\star \quad b \cdot (a \vee b)^\star \sqsubseteq \ldots}}{(a \vee b)^\star \sqsubseteq (a \vee b \vee bb)^\star}$$

Proposals Overview

	Target Language	Specification Language	Applied Domain	Research Paper
1	C	IntegratedEffs	General Effectful Programs	(ICFEM 2020)
2	$Imp^{a/s}$	SyncEffs	Synchronous Programming	(VMCAI 2021)
3	C^{t}	TimEffs	Time Critical Systems	(TACAS 2023)
4	λ_h	ContEffs	Algebraic Effects and Handlers	(APLAS 2022)

Main Challenges

- ❖ <u>Customized</u> forward verifier: to closely capture the semantics of given program;
- ❖ <u>Customized</u> TRS: to solve specifications on different expressiveness level;
- ❖ Soundness and termination proofs for forward verifiers and TRSs.

Outline

1. DependentEffs : General Effectful Programs
   - Mixed finite (inductive) and infinite (coinductive) traces
2. SyncEffs: Synchronous Programming
3. TimEffs: Time Critical Systems
4. ContEffs Algebraic Effects and Handlers
5. Conclusion and the Future Work

Integrated Dependent Effects

```
send n =
        if n == 0 then event [Done];
        else event [Send];
            send (n - 1);
```

Φ' = ($Send^{\star}$ · Done, $Send^{\omega}$) *[Hofmann, Martin, and Wei Chen. 2014]*

Φ'' = ($Send^{n}$ · Done, $Send^{\omega}$) *[Nanjo, Yoji, et al. 2018]*

Integrated Dependent Effects

```
send n =
        if n == 0 then event [Done];
        else event [Send];
             send (n - 1);
```

Φ' = ($Send^{\star}$ · Done, $Send^{\omega}$) *[Hofmann, Martin, and Wei Chen. 2014]*

Φ'' = ($Send^{n}$ · Done, $Send^{\omega}$) *[Nanjo, Yoji, et al. 2018]*

Φ_{pre} = $True \wedge Ready \cdot \_^{*}$

$\Phi_{post}(n) = n \geq 0 \wedge (Send^{n} \cdot Done) \vee n < 0 \wedge (Send^{\omega})$

Integrated Dependent Effects

```
send n =
        if n == 0 then event [Done];
        else event [Send];
            send (n - 1);
```

```
server n =
        event [Ready];
        send (n);
        server (n);
```

$\Phi' = (Send^* \cdot Done, Send^\omega)$

$\Phi'' = (Send^n \cdot Done, Send^\omega)$

$\Phi_{pre} = True \wedge Ready \cdot \_^*$

$\Phi_{post}(n) = n \geq 0 \wedge (Send^n \cdot Done) \vee n < 0 \wedge (Send^\omega)$

$\Phi_{pre} = n \geq 0 \wedge \epsilon$

$\Phi_{post}(n) = n \geq 0 \wedge (Ready \cdot Send^n \cdot Done)^\omega$

$\Phi'_{pre} = True \wedge \epsilon$

$\Phi'_{post}(n) = n \geq 0 \wedge (Ready \cdot Send^n \cdot Done)^\omega$

$\vee\ n < 0 \wedge (Ready \cdot Send^\omega)$

Integrated Dependent Effects

Classic Regular Expressions

$$
\begin{array}{rrcl}
(\textit{Effects}) & \Phi & ::= & \pi \wedge \mathtt{es} \mid \Phi_1 \vee \Phi_2 \mid \exists \mathtt{x}.\Phi \\
(\textit{Event Sequences}) & \mathtt{es} & ::= & \boxed{\bot \mid \epsilon \mid \underline{\mathbf{a}} \mid \mathtt{es}_1 \cdot \mathtt{es}_2 \mid \mathtt{es}_1 \vee \mathtt{es}_2} \mid \_ \mid \boxed{\mathtt{es}^{\mathtt{t}} \mid \mathtt{es}^{\star} \mid \mathtt{es}^{\omega}} \\
(\textit{Pure Formulae}) & \pi & ::= & \mathtt{True} \mid \mathtt{False} \mid \mathtt{A}(\mathtt{t}_1, \mathtt{t}_2) \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \\
 & & & \mid \neg\pi \mid \pi_1 \Rightarrow \pi_2 \mid \forall \mathtt{x}.\pi \mid \exists \mathtt{x}.\pi
\end{array}
$$

Revised semantics of the repeated patterns

$$\mathbf{x} ::\in \mathbf{var} \qquad \mathbf{n} ::\in \mathbb{Z} \qquad (\textit{Event})\ \underline{\mathbf{a}} ::\in \Sigma \qquad (\textit{Infinity})\ \omega \qquad (\textit{Kleene Star})\ \star$$

1. Aware of termination (mixed definition): $(n \geq 0 \wedge Send^{n} \cdot Done) \vee (n < 0 \wedge Send^{\omega})$
2. Beyond the context-free grammar: $a^{n} \cdot b^{n} \cdot c^{n}$
3. Effects in precondition is new: $\mathbf{\Phi}_{pre} = True \wedge Ready \cdot \_^{*}$
4. Undetermined termination (Kleene Star): $True \wedge Send^{\star} \cdot Done$

Implementation and Evaluation

- An open-sourced prototype system using OCaml.
- Benchmark: 16 IOT programs implemented in C for Arduino controlling programs:
  - derive temporal properties (in total 235 properties with 124 valid and 111 invalid)
  - express these properties using both LTL formulae and our effects,
  - we record the total computation time using PAT and our TRS.

Implementation and Evaluation

- An oper
- Benchm … controlling program
  - ➢ derive … and 111 invalid)
  - ➢ expre
  - ➢ we re

Table 5. The experiments are based on 16 real world C programs, we record the lines of code (LOC), the number of testing temporal properties (#Prop.), and the (dis-) proving times (in milliseconds) using PAT and our T.r.s respectively.

Programs	**LOC**	**#Prop.**	**PAT(ms)**	**T.r.s(ms)**
1. Chrome_Dino_Game	80	12	32.09	7.66
2. Cradle_with_Joystick	89	12	31.22	9.85
3. Small_Linear_Actuator	180	12	21.65	38.68
4. Large_Linear_Actuator	155	12	17.41	14.66
5. Train_Detect	78	12	19.50	17.35
6. Motor_Control	216	15	22.89	4.71
7. Train_Demo_2	133	15	49.51	59.28
8. Fridge_Timer	292	15	17.05	9.11
9. Match_the_Light	143	15	23.34	49.65
10. Tank_Control	104	15	24.96	19.39
11. Control_a_Solenoid	120	18	36.26	19.85
12. IoT_Stepper_Motor	145	18	27.75	6.74
13. Aquariumatic_Manager	135	10	25.72	3.93
14. Auto_Train_Control	122	18	56.55	14.95
15. LED_Switch_Array	280	18	44.78	19.58
16. Washing_Machine	419	18	33.69	9.94
Total	2546	235	446.88	305.33

Outline

1. DependentEffs : General Effectful Programs
2. SyncEffs: Reactive Systems
   - Synchronous program, logical correctness, causality
3. TimEffs: Time Critical Systems
4. ContEffs Algebraic Effects and Handlers
5. Conclusion and the Future Work

Esterel – A synchronous language

[Berry G, Gonthier G. 1992]
[Jagadeesan L J, Puchol C, Von Olnhausen J E. 1995]
[Florence, Spencer P., et al. 2019]

- System-design/modelling language.
- Deterministic semantics.
- Primitive constructs execute in zero time except for the pause statement.
- The (i) correctness and (ii) safety issues are particularly critical.

```
signal S1 {
  present S1
  then nothing
  else emit S1
}
```

(a) No valid assignments (Logically incorrect).

```
signal S1 {
  present S1
  then emit S1
  else nothing
}
```

(b) Two possible assignments (Logically incorrect).

```
/*@ requires  S1  @*/
/*@ ensures   S1  @*/
present S1
then emit S1
else nothing
```

(c) One assignment under the precondition (Logically correct).

Target Language $\lambda^{a/s}$, extending Esterel with synchronous constructs

$$
\begin{array}{llcl}
(Program) & \mathcal{P} & ::= & \overrightarrow{module} \\
(Basic\ Types) & \tau & ::= & IN \mid OUT \mid INOUT \\
(Module\ Def.) & module & ::= & nm\ \ (\overrightarrow{\tau\ S})\ \langle \mathbf{req}\ \ \Phi_{pre}\ \ \mathbf{ens}\ \ \Phi_{post} \rangle\ \ p \\
(Statement) & p,\ q & ::= & nothing \mid pause \mid emit\ S \mid p; q \mid p || q \mid loop\ p \\
 & & & \mid signal\ S\ in\ p \mid present\ S\ then\ p\ else\ q \mid call\ mn\ (\overrightarrow{S}) \\
 & & & \mid try\ p\ with\ q \mid raise\ d \mid async\ S\ p\ q \mid await\ S
\end{array}
$$

$(Signal\ \ Variables)\ S \in \Sigma \qquad x, mn \in \mathbf{var} \qquad (Depth)\ d \in \mathbb{N} \cup \{0\}$

Specification Language SyncEffs:

$$
\begin{array}{rcll}
(Effects) & \Phi & ::= & \bot \mid \epsilon \mid I \mid S? \mid \Phi_1 \cdot \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid \Phi_1 || \Phi_2 \mid \Phi^\star \\
(event) & I & ::= & \{\} \mid \{S \mapsto \alpha\} \mid I_1 \cup I_2 \\
(Signal\ Status) & \alpha & ::= & present \mid absent \mid undef
\end{array}
$$

$(Signal\ \ Variables)\ S \in \Sigma \qquad (Blocking\ \ Waiting)\ ? \qquad (Kleene\ \ Star)\ \star$

Logically incorrect examples, caught by SyncEffs.

1. *present* $S1$
$\langle\{S1 \mapsto undef\}\rangle$

2. *then*
$\langle\{S1 \mapsto undef, S1 \mapsto present\}\rangle$

3. *nothing*
$\langle\{S1 \mapsto undef, S1 \mapsto present\}\rangle$

4. *else*
$\langle\{S1 \mapsto undef, S1 \mapsto absent\}\rangle$

5. *emit* $S1$
$\langle\{S1 \mapsto present, S1 \mapsto absent\}\rangle$

6. $\langle\{S1 \mapsto undef, S1 \mapsto present\} \vee \{S1 \mapsto present, S1 \mapsto absent\}\rangle$
$\langle\bot \vee \bot\rangle \Rightarrow \langle\bot\rangle$

(a)

1. *present* $S1$
$\langle\{S1 \mapsto undef\}\rangle$

2. *then*
$\langle\{S1 \mapsto undef, S1 \mapsto present\}\rangle$

3. *emit* $S1$
$\langle\{S1 \mapsto present, S1 \mapsto present\}\rangle$

4. *else*
$\langle\{S1 \mapsto undef, S1 \mapsto absent\}\rangle$

5. *nothing*
$\langle\{S1 \mapsto undef, S1 \mapsto absent\}\rangle$

6. $\langle\{S1 \mapsto present, S1 \mapsto present\} \vee \{S1 \mapsto undef, S1 \mapsto absent\}\rangle$
$\langle\{S1 \mapsto present\} \vee \{S1 \mapsto absent\}\rangle$

(b)

```
module a_bug:
output S;
/*@
require {}
ensure {S}
@*/
signal S in
  present S then emit S
  else emit S
  end present end signal
end module
```

Constructiveness

the status of the tested signal must be determined before executing the sub-expressions.

Outline

1. DependentEffs : General Effectful Programs
2. SyncEffs: Synchronous Programming
3. TimEffs: Time Critical Systems
    - mutable variables and concurrency
    - timed behavioural patterns, such as delay, timeout, interrupt, deadline, etc.
4. ContEffs Algebraic Effects and Handlers
5. Conclusion and the Future Work

Timed Verification via Timed Automata

- Timed Automata lack high-level compositional patterns for hierarchical design.
- Manually casting clocks is tedious and error-prone.
- Timed CSP, is translated to Timed Automata (TA) so that the model checker Uppaal can be applied.

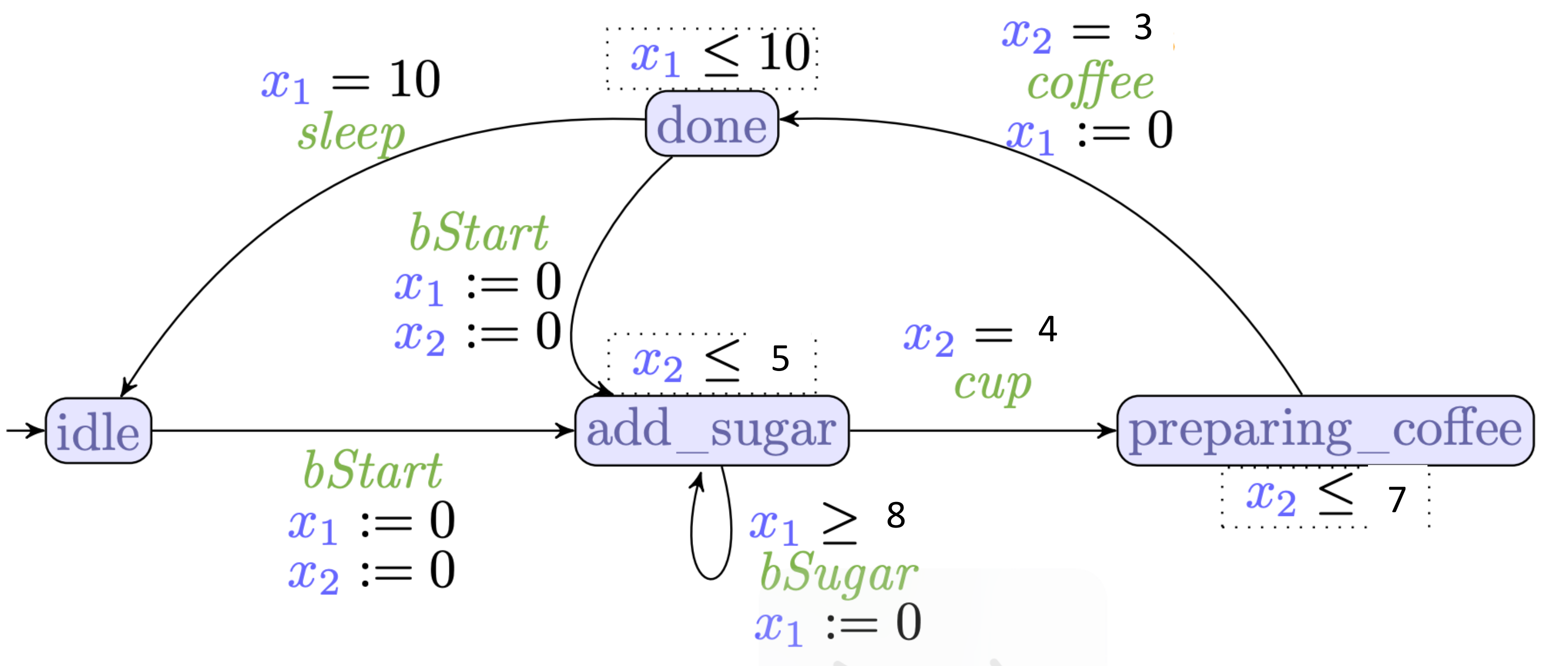


Diagram modified from “Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata” in FTSCS ’22

Timed Verification via Timed Automata

- Timed Automata lack high-level compositional patterns for hierarchical design.
- Manually casting clocks is tedious and error-prone.
- Timed CSP, is translated to Timed Automata (TA) so that the model checker Uppaal can be applied.

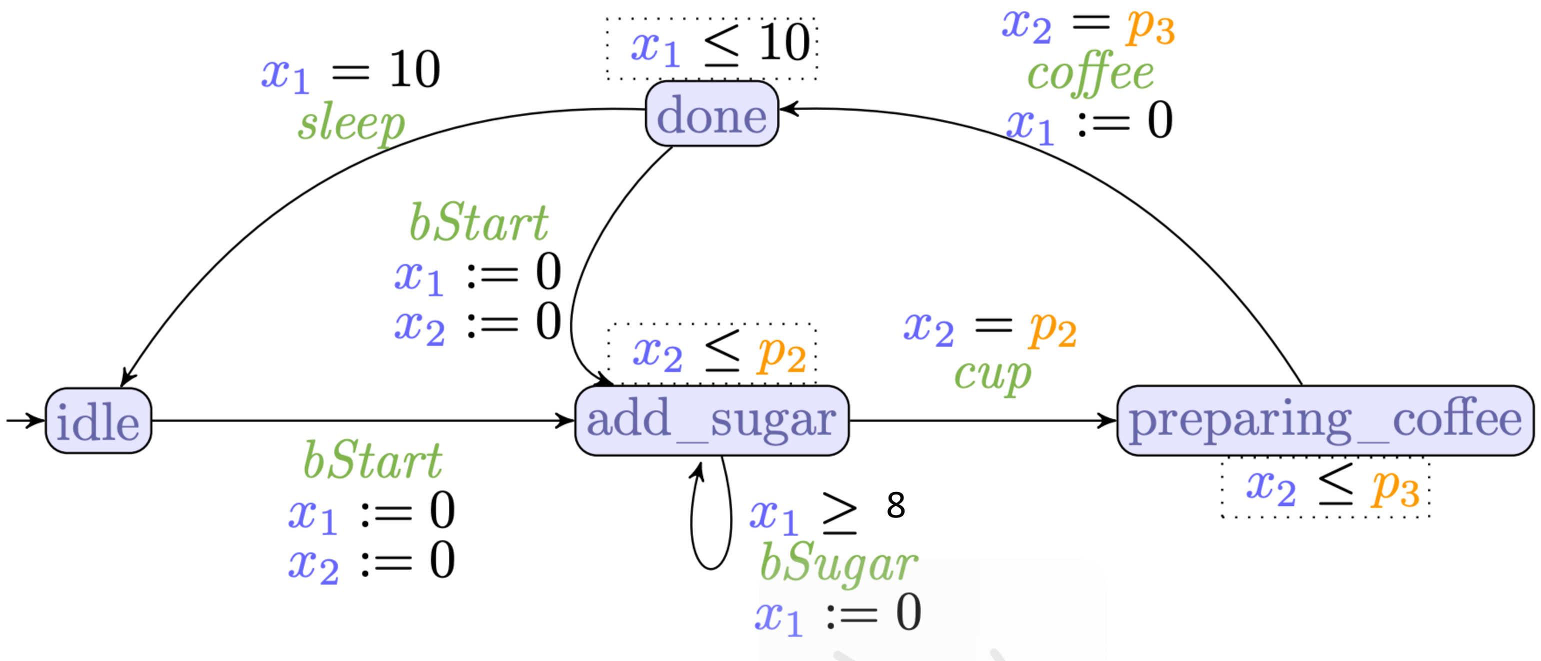


Diagram modified from "Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata" in FTSCS '22

We propose TimEffs - Symbolic Timed Automata

```
void addOneSugar()
/* req: true ∧ _*
   ens: t>1 ∧ ϵ # t */
{ timeout ((), 1); }

```

Target Language C^t, imperative with timed constructs:

$$(Expressions) \quad e ::= v \mid \alpha \mid [v]e \mid mn(v^*) \mid e_1; e_2 \mid e_1 || e_2 \mid \mathit{if}\ v\ e_1\ e_2 \mid \texttt{event}[\texttt{A}(v, \alpha^*)]$$
$$\mid \texttt{delay}[v] \mid e_1\ \texttt{timeout}[v]\ e_2 \mid e\ \texttt{deadline}[v] \mid e_1\ \texttt{interrupt}[v]\ e_2$$

$c \in \mathbb{Z}$ $\quad b \in \mathbb{B}$ $\quad mn, x \in \textbf{var}$ $\quad (Action\ labels)\ \texttt{A} \in \Sigma$

Specification Language TimEffs:

$$(Timed\ Effects) \quad \Phi ::= \pi \wedge \theta \mid \Phi_1 \vee \Phi_2$$
$$(Event\ Sequences) \quad \theta ::= \bot \mid \epsilon \mid ev \mid \theta_1 \cdot \theta_2 \mid \theta_1 \vee \theta_2 \mid \theta_1 || \theta_2 \mid \pi ? \theta \mid \theta \# t \mid \theta^\star$$

$$(Pure) \quad \pi ::= True \mid False \mid bop(t_1, t_2) \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \neg \pi \mid \pi_1 \Rightarrow \pi_2$$
$$(Real\text{-}Time\ Terms) \quad t ::= c \mid x \mid t_1 + t_2 \mid t_1 - t_2$$

$c \in \mathbb{Z}$ $\quad x \in \textbf{var}$ $\quad (Real\ Time\ Bound)\ \#$ $\quad (Kleene\ Star)\ \star$

Inclusion Checking – SMT based Term Rewriting

```
void addNSugar (int n)
/* req:  true ∧ _*
  ens:  t≥n ∧ EndSugar # t */
{ if (n == 0) { event ["EndSugar"];}
  else {
    addOneSugar();
    addNSugar (n-1);}}
```

$$(\mathtt{n{=}0} \wedge \mathtt{ES}) \vee (\mathtt{n{\neq}0{\wedge}t2{>}1} \wedge (\epsilon \ \# \ \mathtt{t2}) \cdot \Phi_{post}^{addNSugar(n\text{-}1)}) \sqsubseteq \Phi_{post}^{addNSugar(n)}$$

Inclusion Checking – SMT based Term Rewriting

```
void addNSugar (int n)
/* req:  true ∧ _*
  ens:  t≥n ∧ EndSugar # t */
{ if (n == 0) { event ["EndSugar"];}
  else {
    addOneSugar();
    addNSugar (n-1);}}
```

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\texttt{n=0} \wedge \epsilon \sqsubseteq \texttt{tR}\geq\texttt{0} \wedge \epsilon \ \#\ \texttt{tR}} \ ④\,[PROVE]}{\texttt{n=0} \wedge \cancel{\texttt{ES}} \sqsubseteq \texttt{tR}\geq\texttt{0} \wedge \cancel{\texttt{ES}}\#\texttt{tR}} \ ③[UNFOLD]}{(\texttt{n=0}\wedge\texttt{ES}) \vee}}{(\texttt{n=0} \wedge \texttt{ES}) \vee (\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1} \wedge (\epsilon \ \#\ \texttt{t2}) \cdot \Phi_{post}^{addNSugar(n\text{-}1)}) \sqsubseteq \Phi_{post}^{addNSugar(n)}}$$

Inclusion Checking – SMT based Term Rewriting

```
void addNSugar (int n)
/* req:  true ∧ _*
  ens:  t≥n ∧ EndSugar # t */
{ if (n == 0) { event ["EndSugar"];}
  else {
    addOneSugar();
    addNSugar (n-1);}}
```

$$(I)$$

$$\frac{\vee\ (\mathtt{n}\neq 0 \wedge \mathtt{t2}>1 \wedge \mathtt{tL}\geq(\mathtt{n}-1) \wedge\ \epsilon\#\mathtt{t2}\cdot \mathtt{ES}\#\mathtt{tL}) \sqsubseteq \mathtt{tR}\geq\mathtt{n} \wedge \mathtt{ES}\#\mathtt{tR}}{(\mathtt{n}=0 \wedge \mathtt{ES}) \vee (\mathtt{n}\neq 0 \wedge \mathtt{t2}>1 \wedge (\epsilon\ \#\ \mathtt{t2})\cdot\Phi_{post}^{addNSugar(n\text{-}1)}) \sqsubseteq \Phi_{post}^{addNSugar(n)}}$$

② [LHS-OR]

① [RENAME]

(I)

$$\mathtt{t2}>1 \wedge \mathtt{tL}\geq(\mathtt{n}-1) \wedge \mathtt{tL}=(\mathtt{tR}-\mathtt{t2}) \Rightarrow \mathtt{tR}\geq\mathtt{n}$$

⑦ [PROVE]

$$\mathtt{n}\neq 0 \wedge \mathtt{t2}>1 \wedge \mathtt{tL}\geq(\mathtt{n}-1) \wedge \epsilon \sqsubseteq \mathtt{tR}\geq\mathtt{n} \wedge \epsilon$$

⑥ [UNFOLD] π_u: $\mathtt{tL}=(\mathtt{tR}-\mathtt{t2})$

$$\mathtt{n}\neq 0 \wedge \mathtt{t2}>1 \wedge \mathtt{tL}\geq(\mathtt{n}-1) \wedge \cancel{\mathtt{ES}\#\mathtt{tL}} \sqsubseteq \mathtt{tR}\geq\mathtt{n} \wedge \cancel{\mathtt{ES}\#(\mathtt{tR}-\mathtt{t2})}$$

⑤ [UNFOLD]

$$\mathtt{n}\neq 0 \wedge \mathtt{t2}>1 \wedge \mathtt{tL}\geq(\mathtt{n}-1) \wedge \cancel{\epsilon\#\mathtt{t2}}\cdot \mathtt{ES}\#\mathtt{tL} \sqsubseteq \mathtt{tR}\geq\mathtt{n} \wedge \cancel{\mathtt{ES}\#\mathtt{tR}}$$

Inclusion Checking – SMT based Term Rewriting

```
void addNSugar (int n)
/* req:  true ∧ _*
  ens:  t≥n ∧ EndSugar # t */
{ if (n == 0) { event ["EndSugar"];}
  else {
    addOneSugar();
    addNSugar (n-1);}}
```

Succeed!

$$\frac{\dfrac{\dfrac{}{\texttt{n=0} \wedge \epsilon \sqsubseteq \texttt{tR}\geq\texttt{0} \wedge \epsilon \,\#\, \texttt{tR}}\;④\,[PROVE]}{\texttt{n=0} \wedge \cancel{\texttt{ES}} \sqsubseteq \texttt{tR}\geq\texttt{0} \wedge \cancel{\texttt{ES}}\texttt{\#tR}}\;③[UNFOLD] \qquad (I)}{\dfrac{(\texttt{n=0}\wedge\texttt{ES}) \vee (\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1}\wedge\texttt{tL}\geq\texttt{(n-1)}\wedge \epsilon\texttt{\#t2} \cdot \texttt{ES\#tL}) \sqsubseteq \texttt{tR}\geq\texttt{n} \wedge \texttt{ES\#tR}}{(\texttt{n=0} \wedge \texttt{ES}) \vee (\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1} \wedge (\epsilon \,\#\, \texttt{t2}) \cdot \Phi_{post}^{addNSugar(n\text{-}1)}) \sqsubseteq \Phi_{post}^{addNSugar(n)}}\;①[RENAME]}\;②[LHS\text{-}OR]$$

(I)

$$\dfrac{\dfrac{\dfrac{\texttt{t2>1}\wedge\texttt{tL}\geq\texttt{(n-1)} \wedge \texttt{tL=(tR-t2)} \Rightarrow \texttt{tR}\geq\texttt{n}}{\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1}\wedge\texttt{tL}\geq\texttt{(n-1)} \wedge \epsilon \sqsubseteq \texttt{tR}\geq\texttt{n} \wedge \epsilon}\;⑦[PROVE]}{\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1}\wedge\texttt{tL}\geq\texttt{(n-1)} \wedge \cancel{\texttt{ES\#tL}} \sqsubseteq \texttt{tR}\geq\texttt{n} \wedge \cancel{\texttt{ES\#(tR-t2)}}}\;⑥[UNFOLD]\;\pi_u\texttt{:tL=(tR-t2)}}{\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1}\wedge\texttt{tL}\geq\texttt{(n-1)} \wedge \cancel{\epsilon\texttt{\#t2}\cdot}\; \texttt{ES\#tL} \sqsubseteq \texttt{tR}\geq\texttt{n} \wedge \cancel{\texttt{ES\#tR}}}\;⑤[UNFOLD]$$

Antimirov algorithm for solving REs' inclusions

Definition 1 (Derivatives). *Given any formal language S over an alphabet Σ and any string $u \in \Sigma^*$, the derivatives of S w.r.t u is defined as: $u^{-1}S = \{w \in \Sigma^* \mid uw \in S\}$.*

Definition 2 (Regular Expression Inclusion). ***For REs r and s,***

$$r \preceq s \Leftrightarrow \boxed{\forall A \in \Sigma.}\ A^{-1}(r) \preceq A^{-1}(s)\ .$$

Antimirov algorithm for solving TimEffs' inclusions

Definition 3 (*TimEffs* Inclusion). ***For*** *TimEffs Φ_1 and Φ_2,*

$$\Phi_1 \sqsubseteq \Phi_2 \Leftrightarrow \boxed{\forall A \in \Sigma.\ \forall t \geq 0.}\ (A\#t)^{-1}\,\Phi_1 \sqsubseteq (A\#t)^{-1}\,\Phi_2\ .$$

Implementation and Evaluation

Table 5.3: Experimental Results for Manually Constructed Synthetic Examples.

No.	LOC	Forward(ms)	#Prop(✓)	Avg-Prove(ms)	#Prop(✗)	Avg-Dis(ms)	#AskZ3
1	26	0.006	5	52.379	5	21.31	77
2	37	43.955	5	83.374	5	52.165	188
3	44	32.654	5	52.524	5	33.444	104
4	72	202.181	5	82.922	5	55.971	229
5	98	42.706	7	149.345	7	60.325	396
6	134	403.617	7	160.932	7	292.304	940
7	133	51.492	7	17.901	7	47.643	118
8	173	57.114	7	40.772	7	30.977	128
9	182	872.995	9	252.123	9	113.838	1142
10	210	546.222	9	146.341	9	57.832	570
11	240	643.133	9	146.268	9	69.245	608
12	260	1032.31	9	242.699	9	123.054	928
13	265	12558.05	11	150.999	11	117.288	2465
14	286	12257.834	11	501.994	11	257.800	3090
15	287	1383.034	11	546.064	11	407.952	1489
16	337	49873.835	11	1863.901	11	954.996	15505

Main Observations:

the disproving times for invalid properties are constantly lower than the proving process.

Evaluation – Fischer's Mutual Exclusion Algorithm

Table 5. ... hm

#Proc	...)
2	
3	
4	
5	

```
var x := -1;
var cs:= 0;

void proc (int i) {
   [x=-1] //block waiting until true
   deadline(event["Update"(i)]{x:=i},d);
   delay (e);
   if (x==i) {
      event["Critical"(i)]{cs:=cs+1};
      event["Exit"(i)]{cs:=cs-1;x:=-1};
      proc (i);
   } else {proc (i);}}

void main ()
/* req:  d<e ∧ ϵ
  ens_a:true ∧ (cs≤1)*   ens_b:true ∧ ((_*).Critical.Exit.(_*))* */
  { proc(0) || proc(1) || proc(2); }
```

Evaluation – Fischer's Mutual Exclusion Algorithm

Table 5.4: Comparison with PAT via verifying Fischer's mutual exclusion algorithm

#Proc	Prove(s)	#AskZ3-u	Disprove(s)	#AskZ3-u	PAT(s)	Uppaal(s)
2	0.09	31	0.110	37	$\leq$0.05	$\leq$0.09
3	0.21	35	0.093	42	$\leq$0.05	$\leq$0.09
4	0.46	63	0.120	47	0.05	0.09
5	25.0	84	0.128	52	0.15	0.19

Observations:

i. automata-based model checkers (both PAT and Uppaal) are vastly efficient when given concrete values for constants d and e;

ii. our proposal can symbolically prove the algorithm by only providing the constraints, of d and e.

iii. our verification time largely depends on the number of querying Z3.

Outline

1. DependentEffs : General Effectful Programs
2. SyncEffs: Synchronous Programming
3. TimEffs: Time Critical Systems
4. ContEffs Algebraic Effects and Handlers
   - The coexistence of zero-shot, one-shot and multi-shot continuations
   - Non-terminating behaviours.
5. Conclusion and the Future Work

User-defined Effects and Handlers

[de Vilhena, Paulo Emílio, and François Pottier. 2021]

[Sivaramakrishnan, K. C., et al. 2021]

```
effect E : string

let comp () =
  print_string "0 ";
  print_string (perform E);
  print_string "3 "

let main () =
  try
    comp ()
  with effect E k ->
    print_string "1 ";
    continue k "2 ";
    print_string “4 "
```


Example taken from “Effect Handlers in Multicore OCaml” slides by KC Sivaramakrishnan.

User-defined Effects and Handlers

This prints: 0 1 2 3 4

```
effect E : string

let comp () =
  print_string "0 ";
  print_string (perform E);
  print_string "3 "

let main () =
  try
    comp ()
  with effect E k ->
    print_string "1 ";
    continue k "2 ";
    print_string “4 "
```

effect declaration

suspends current computation

computation

delimited continuation

handler

resume suspended computation

Example taken from “Effect Handlers in Multicore OCaml” slides by KC Sivaramakrishnan.

Core Language λh: pure, higher-order, call by value

$$
\begin{array}{ll}
(\mathit{Values}) & v ::= c \mid x \mid \lambda x \Rightarrow e \\
(\mathit{Expressions}) & e ::= v \mid v_1\ v_2 \mid \mathit{let}\ x{=}v\ \mathit{in}\ e \mid \mathit{if}\ v\ \mathit{then}\ e_1\ \mathit{else}\ e_2 \mid \\
& \quad \underline{\mathit{perform}\ A(v, \lambda x \Rightarrow e) \mid \mathit{match}\ e\ \mathit{with}\ h \mid \mathit{resume}\ v}
\end{array}
$$

Specification Language ContEffs

$$
\begin{array}{rrcl}
(\mathit{ContEffs}) & \Phi & ::= & \bigvee(\pi, \theta, v) \\
(\mathit{Parameterized\ Label}) & l & ::= & \Sigma(v) \\
(\mathit{Event\ Sequences}) & \theta & ::= & \bot \mid \epsilon \mid ev \mid Q \mid \theta_1 \cdot \theta_2 \mid \theta_1 \vee \theta_2 \mid \theta^\star \mid \theta^\infty \mid \theta^\omega \\
(\mathit{Single\ Events}) & ev & ::= & \_ \mid l \mid \overline{l} \\
(\mathit{Placeholders}) & Q & ::= & \underline{l! \mid l?(v)}
\end{array}
$$

Examples – Zero-shot continuations (Exceptions)

```
effect Exc : (unit -> unit)
effect Other : (unit -> unit)

let f ()
(*@ req emp @*)
(*@ ens Exc!.Other!.Other?().Exc?() @*)
= let x = perform Exc in
  let y = perform Other in
  y ();
  x ()

let handler
(*@ req emp @*)
(*@ ens Exc @*)
= match f () with
| x -> x
| effect Exc k -> ()
```

Step	History	Current Event	Continuation	Bindings
1	emp	Exc!	Other! · Other?() · Exc?() · ‡	‡ = (fun x -> x)
2	Exc	-	-	No “Continue”
Final	Exc	-	-	

Examples – Multi-shot continuation

```
effect Foo : (unit -> int)
effect Goo : (unit -> int)
effect Done: (unit)

let f ()
(*@ req emp @*)
(*@ ens Foo!.Goo!.Goo?().Foo?() @*)
= let x = perform Foo in
  let y = perform Goo in
  y (); x ()

let handler
(*@ req emp @*)
(*@ ens Foo.Goo.Done!.
          Goo.Done! @*)
= match f () with
| x -> perform Done;
| effect Foo k -> continue k (fun () -> ());
                  continue k (fun () -> ())
| effect Goo k -> continue k (fun () -> ())
```

Implementation and Evaluation

- Core implementation: 2500 LOC in OCaml, on top of Multicore OCaml (4.12.0)
- Validation: manually annotated synthetic test cases marked with expected outputs

No.	LOC	Infer(ms)	#Prop(✓)	Avg-Prove(ms)	#Prop(✗)	Avg-Dis(ms)
1	32	14.128	5	7.7786	5	6.2852
2	48	14.307	5	7.969	5	6.5982
3	71	15.029	5	7.922	5	6.4344
4	98	14.889	5	18.457	5	7.9562
5	156	14.677	7	10.080	7	4.819
6	197	15.471	7	8.3127	7	6.8101
7	240	18.798	7	18.559	7	7.468
8	285	20.406	7	23.3934	7	9.9086
9	343	26.514	9	16.5666	9	13.9667
10	401	26.893	9	18.3899	9	10.2169
11	583	49.931	14	17.203	15	10.4443
12	808	75.707	25	21.6795	24	16.9064

Summary & Links

- New framework for temporal verification.
  - ❖ More modular – a compositional verification strategy.
  - ❖ Finer-grained – semantics oriented, forward verifiers.
  - ❖ More efficient – term rewriting systems.
- Implementations upon possible application scenes and evaluations.
  - ❖ General Effectful Programs (ICFEM 2020) [PDF] [Video] [Code]
  - ❖ Reactive Systems (VMCAI 2021) [PDF] [Video] [Code1&Code2]
  - ❖ Time Critical Systems (TACAS 2023) [PDF] [Code]
  - ❖ Algebraic Effects and Handlers (APLAS 2022) [PDF] [Code]

Possible Future Work

Thank you for your attention!

- Symbolic verification for probabilistic programming
- Temporal verification for hyper-properties (hyper temporal logic)
- Practical analysis for mixed synchronous and asynchronous features
- Trace-based verification with spatial information
  - ❖Ongoing work: “Extending Separation Logic for Unrestricted Effect Handlers”
- Temporal verification with incorrectness logic
- Program-analyzer based repair
  - ❖Ongoing work: “Automated Program Repair guided by Temporal Properties”

Bibliography (I)

[Hofmann, Martin, and Wei Chen. 2014] "Abstract interpretation from Büchi automata." *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. 2014.

[Nanjo, Yoji, et al. 2018] "A fixpoint logic and dependent effects for temporal property verification." *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*. 2018.

[Berry G, Gonthier G. 1992] “The Esterel synchronous programming language: Design, semantics, implementation.” *Science of computer programming*, *19*(2), 87-152.

[Florence, Spencer P., et al. 2019] "A calculus for Esterel: if can, can. if no can, no can." *Proceedings of the ACM on Programming Languages* 3.POPL (2019): 1-29.

[Larsen, Kim G., Paul Pettersson, and Wang Yi. 1997] "UPPAAL in a nutshell." *International journal on software tools for technology transfer* 1 (1997): 134-152.

Bibliography (II)

[Jagadeesan L J, Puchol C, Von Olnhausen J E. 1995] “Safety property verification of Esterel programs and applications to telecommunications software.” In *Computer Aided Verification: 7th International Conference, CAV'95 Liège, Belgium, July 3–5, 1995 Proceedings 7* (pp. 127-140). Springer Berlin Heidelberg.

[Dong, Jin Song, et al. 2008] "Timed automata patterns." *IEEE Transactions on Software Engineering* 34.6 (2008): 844-859.

[Arias, Jaime, et al. 2022] "Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata." *Proceedings of the 8th ACM SIGPLAN International Workshop on Formal Techniques for Safety-Critical Systems*. 2022.

[de Vilhena, Paulo Emílio, and François Pottier. 2021] "A separation logic for effect handlers." *Proceedings of the ACM on Programming Languages* 5.POPL (2021): 1-28.

[Sivaramakrishnan, K. C., et al. 2021] "Retrofitting effect handlers onto OCaml." *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*. 2021.

Thesis Revision Plan

On the comments of Examiner 1

1. Add more details of the similarities to the types-and-effects system;
2. Enrich the introductory with background material, such as the detailed comparison between automata-based and the RE-based entailment proving;
3. Emphasize the novel departure (for each of the separated works) from the original Antimirov algorithm;
4. Expand the discussions of various experiments, and the results will be summarized rigorously against the adversaries or baselines.

Thesis Revision Plan

On the comments of Examiner 2

1. In Chapters 3 ~ 6, move the examples to later sections after technical definitions;
2. Move the essence proofs to the main text and leave the simple ones as lemmas;
3. Add Rules for precondition strengthening and postcondition weakening;
4. Gather the forward rules into a figure in each of the chapters;
5. In tables 4.4, 5.3, and 6.1, compare results with existing methods, or justify why no comparison is made (e.g., no similar tools exist).