



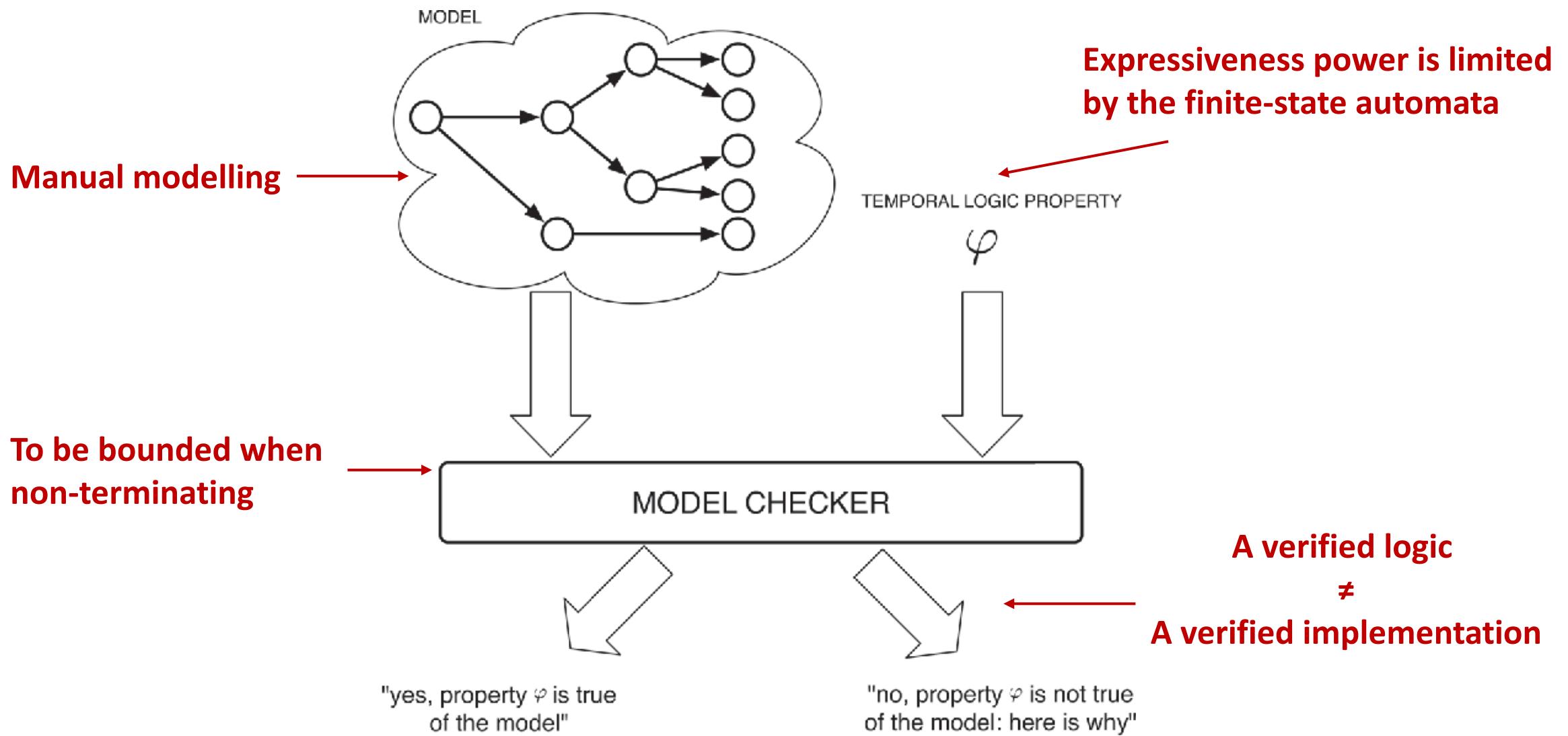
Automated Temporal Verification with Extended Regular Expressions

Thesis Presentation @ LSD, UCSC

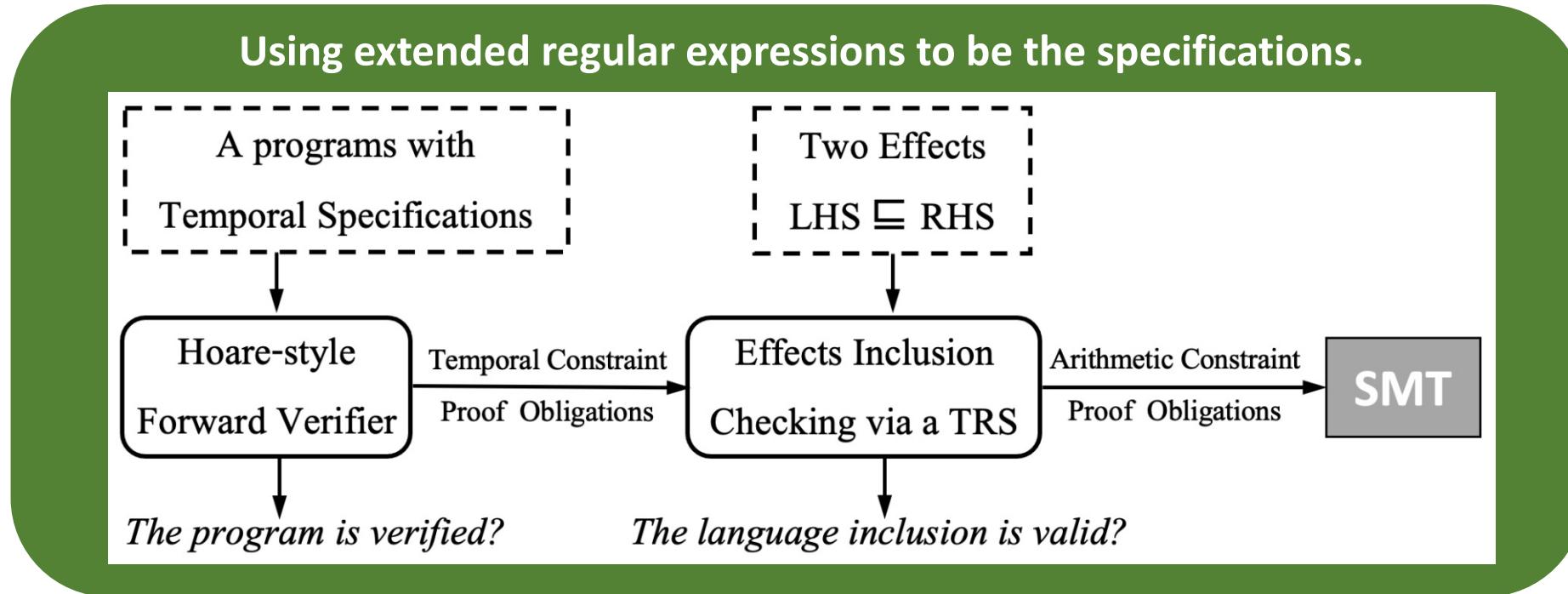
By [Yahui Song](#), 4th Nov 2022

Supervised by Prof. Chin Wei Ngan

Temporal Verification



A New Framework for Temporal Verification



- + Feeding the program with annotated specifications directly.
 - ✓ A verified logic = A verified implementation
- + Flexible specifications, which can be combined with other logic.
- + Symbolic entailment checker with co-inductive proofs for infinite traces.
- Automation/Decidability.

Proposals Overview

	Target Language	Specification Language	Applied Domain	Publication
1	C	DependentEffs	General Effectful Programs	[ICFEM 2020]
2	Imp ^{a/s}	SyncEffs	Reactive Systems	[VMCAI 2021]
3	C ^t	TimEffs	Time Critical Systems	Under submission
4	λ_h	ContEffs	Algebraic Effects and Handlers	[APLAS 2022]

A. Traditional Sequential Control Flows

Main Challenges

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

B-1. Distributed systems with shared variables

B-2. Complex non-local control flow

A. Traditional Sequential Control Flow

- 1. DependentEffs : General Effectful Programs
- 2. SyncEffs: Reactive Systems

Integrated Dependent Effects

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

$$\Phi' = (\text{Send}^* \cdot \text{Done}, \text{Send}^\omega) \quad [\text{Martain 2014}]$$

$$\Phi'' = (\text{Send}^{\textcolor{red}{n}} \cdot \text{Done}, \text{Send}^\omega) \quad [\text{Yoji 2018}]$$

Integrated Dependent Effects

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

$$\Phi' = (\text{Send}^* \cdot \text{Done}, \text{Send}^\omega) \quad [\text{Martain 2014}]$$

$$\Phi'' = (\text{Send}^n \cdot \text{Done}, \text{Send}^\omega) \quad [\text{Yoji 2018}]$$

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

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

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);
```

$\Phi' = (\text{Send}^* \cdot \text{Done}, \text{Send}^\omega)$ [Martain 2014]

$\Phi'' = (\text{Send}^n \cdot \text{Done}, \text{Send}^\omega)$ [Yoji 2018]

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

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

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

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

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);
```

$\Phi' = (\text{Send}^* \cdot \text{Done}, \text{Send}^\omega)$ [Martain 2014]

$\Phi'' = (\text{Send}^n \cdot \text{Done}, \text{Send}^\omega)$ [Yoji 2018]

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

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

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

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

$\Phi_{\text{pre}} = \text{True} \wedge \epsilon$

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

Integrated Dependent Effects – Summary

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

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

Forward Verifier

send n =

if n == 0 **then event** [Done];

else event [Send];

send (n - 1);

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

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

Forward Verifier

send n =

→ {True \wedge Ready $\cdot _^*$ }

if n == 0 then event [Done];

→ {n=0 \wedge Ready $\cdot _^* \cdot$ Done}

else event [Send];

send (n - 1);

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

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

Forward Verifier

send n =

{True \wedge Ready $\cdot _^*$ }

if n == 0 then event [Done];

{n=0 \wedge Ready $\cdot _^* \cdot$ Done}

else event [Send];

→ {n!=0 \wedge Ready $\cdot _^* \cdot$ Send}

send (n - 1);

→ {n!=0 \wedge Ready $\cdot _^* \cdot$ Send \cdot ((n > 0 \wedge Sendⁿ⁻¹ \cdot Done) \vee (n < 0 \wedge Send ^{ω}))}

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

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

$$n \neq 0 \wedge \text{Ready} \cdot _^* \cdot \text{Send}$$

\sqsubseteq

$$\text{True} \wedge \text{Ready} \cdot _^*$$

Forward Verifier

send n =

{True \wedge Ready $\cdot _^*$ }

if n == 0 **then event** [Done];

{n=0 \wedge Ready $\cdot _^* \cdot$ Done}

else event [Send];

{n!=0 \wedge Ready $\cdot _^* \cdot$ Send}

send (n - 1);

{n!=0 \wedge Ready $\cdot _^* \cdot$ Send \cdot ((n > 0 \wedge Sendⁿ⁻¹ \cdot Done) \vee (n < 0 \wedge Send ^{ω}))}

→ {Ready $\cdot _^* \cdot$ ((n=0 \wedge Done) \vee (n > 0 \wedge Send \cdot Sendⁿ⁻¹ \cdot Done) \vee (n < 0 \wedge Send \cdot Send ^{ω}))}

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

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

Goal: $\Phi_{\text{body}} \sqsubseteq \Phi_{\text{post}}(n)$

- Mix Finite & Infinite traces
- Branching Properties

Goal: $\Phi_{\text{body}} \sqsubseteq \Phi_{\text{post}}(n)$

- Mix Finite & Infinite traces
- Branching Properties

$$(n=0 \wedge \text{Done}) \vee (n > 0 \wedge \text{Send} \cdot \text{Send}^{n-1} \cdot \text{Done}) \vee (n < 0 \wedge \text{Send} \cdot \text{Send}^\omega)$$

\sqsubseteq

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

$$\frac{\Gamma \vdash \Phi_1 \sqsubseteq \Phi \rightsquigarrow \gamma_R^1 \quad \Gamma \vdash \Phi_2 \sqsubseteq \Phi \rightsquigarrow \gamma_R^2}{\Gamma \vdash \Phi_1 \vee \Phi_2 \sqsubseteq \Phi \rightsquigarrow (\gamma_R^1 \vee \gamma_R^2)} \quad \frac{[LHS-OR]}{\Gamma \vdash \Phi \sqsubseteq \Phi_i \rightsquigarrow \gamma_R^i} \quad i \in \{1, 2\}$$

Goal: $\Phi_{body} \sqsubseteq \Phi_{post}(n)$

- Mix Finite & Infinite traces
- Branching Properties

$(n=0 \wedge \text{Done}) \vee (\mathbf{n > 0 \wedge Send \cdot Send^{n-1} \cdot Done}) \vee (n < 0 \wedge \text{Send} \cdot \text{Send}^\omega)$

\sqsubseteq

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

Our TRS is an extension of Antimirov and Mosses' algorithm, which can be deployed to decide the inclusions of two regular expressions (REs) through an iterated process of checking the inclusions of their partial derivatives.

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 ,

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

Term Rewriting System - Example

$$\begin{array}{c} \xrightarrow{\hspace{1cm}} \frac{n > 0 \wedge \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}}}{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \cancel{\underline{\text{Send}}^n \cdot \underline{\text{Done}}}} \quad [\text{UNFOLD}] \\ \hline n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \Phi_{\text{post}}^{\text{send}(n)} \end{array}$$

Term Rewriting System - Example

$$\begin{array}{c} \xrightarrow{\hspace{1cm}} \frac{(n_1=n-1 \wedge n_1 \geq 0) \wedge \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} (\dagger)}{n > 0 \wedge \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}}} \quad [\text{SUBSTITUTE}] \\ \frac{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^n \cdot \underline{\text{Done}}}{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \Phi_{\text{post}}^{\text{send}(n)}} \quad [\text{UNFOLD}] \end{array}$$

Term Rewriting System - Example

$$\frac{\text{CASESPLIT}}{\frac{\text{n}_1=0 \wedge \underline{\text{Done}} \sqsubseteq \underline{\text{Done}} \quad \text{n}_1>0 \wedge \underline{\text{Send}}^{\text{n}_1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{\text{n}_1} \cdot \underline{\text{Done}}}{\frac{\text{(n}_1=\text{n}-1 \wedge \text{n}_1 \geq 0) \wedge \underline{\text{Send}}^{\text{n}_1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{\text{n}_1} \cdot \underline{\text{Done}} (\dagger)}{\frac{\text{SUBSTITUTE}}{\frac{\text{n}>0 \wedge \underline{\text{Send}}^{\text{n}-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{\text{n}-1} \cdot \underline{\text{Done}}}{\frac{\text{UNFOLD}}{\frac{\text{n}>0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{\text{n}-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{\text{n}} \cdot \underline{\text{Done}}}{\frac{\Phi_{\text{post}}^{\text{send}(\text{n})}}{\text{n}>0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{\text{n}-1} \cdot \underline{\text{Done}} \sqsubseteq \Phi_{\text{post}}^{\text{send}(\text{n})}}}}}}}}}}$$

Term Rewriting System - Example

→ $n_1=0 \wedge \epsilon \sqsubseteq \epsilon$ [FRAME]

$$\frac{}{n_1=0 \wedge \text{Done} \sqsubseteq \text{Done}}$$

[CASESPLIT]

$$\frac{n_1 > 0 \wedge \text{Send}^{n_1} \cdot \text{Done} \sqsubseteq \text{Send}^{n_1} \cdot \text{Done}}{(n_1=n-1 \wedge n_1 \geq 0) \wedge \text{Send}^{n_1} \cdot \text{Done} \sqsubseteq \text{Send}^{n_1} \cdot \text{Done} (\dagger)}$$

[SUBSTITUTE]

$$\frac{n > 0 \wedge \text{Send}^{n-1} \cdot \text{Done} \sqsubseteq \text{Send}^{n-1} \cdot \text{Done}}{n > 0 \wedge \text{Send} \cdot \text{Send}^{n-1} \cdot \text{Done} \sqsubseteq \text{Send}^n \cdot \text{Done}}$$

[UNFOLD]

$$n > 0 \wedge \text{Send} \cdot \text{Send}^{n-1} \cdot \text{Done} \sqsubseteq \Phi_{\text{post}}^{\text{send}(n)}$$

Term Rewriting System - Example

$$\frac{n_1=0 \wedge \epsilon \sqsubseteq \epsilon \quad [\text{FRAME}]}{n_1=0 \wedge \underline{\text{Done}} \sqsubseteq \underline{\text{Done}}} \quad \xrightarrow{\hspace{1cm}} \quad \frac{n_1 > 0 \wedge \underline{\text{Send}}^{n_1-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_1-1} \cdot \underline{\text{Done}}}{\begin{array}{c} n_1 > 0 \wedge \cancel{\underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}}} \sqsubseteq \cancel{\underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}}} \\ \hline n_1 > 0 \wedge \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} \end{array}} \quad [\text{UNFOLD}]$$
$$\frac{\begin{array}{c} (n_1=n-1 \wedge n_1 \geq 0) \wedge \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} (\dagger) \\ \hline n > 0 \wedge \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \end{array}}{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^n \cdot \underline{\text{Done}}} \quad [\text{CASESPLIT}]$$
$$\frac{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^n \cdot \underline{\text{Done}}}{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \Phi_{\text{post}}^{\text{send}(n)}} \quad [\text{SUBSTITUTE}]$$
$$\frac{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^n \cdot \underline{\text{Done}}}{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \Phi_{\text{post}}^{\text{send}(n)}} \quad [\text{UNFOLD}]$$

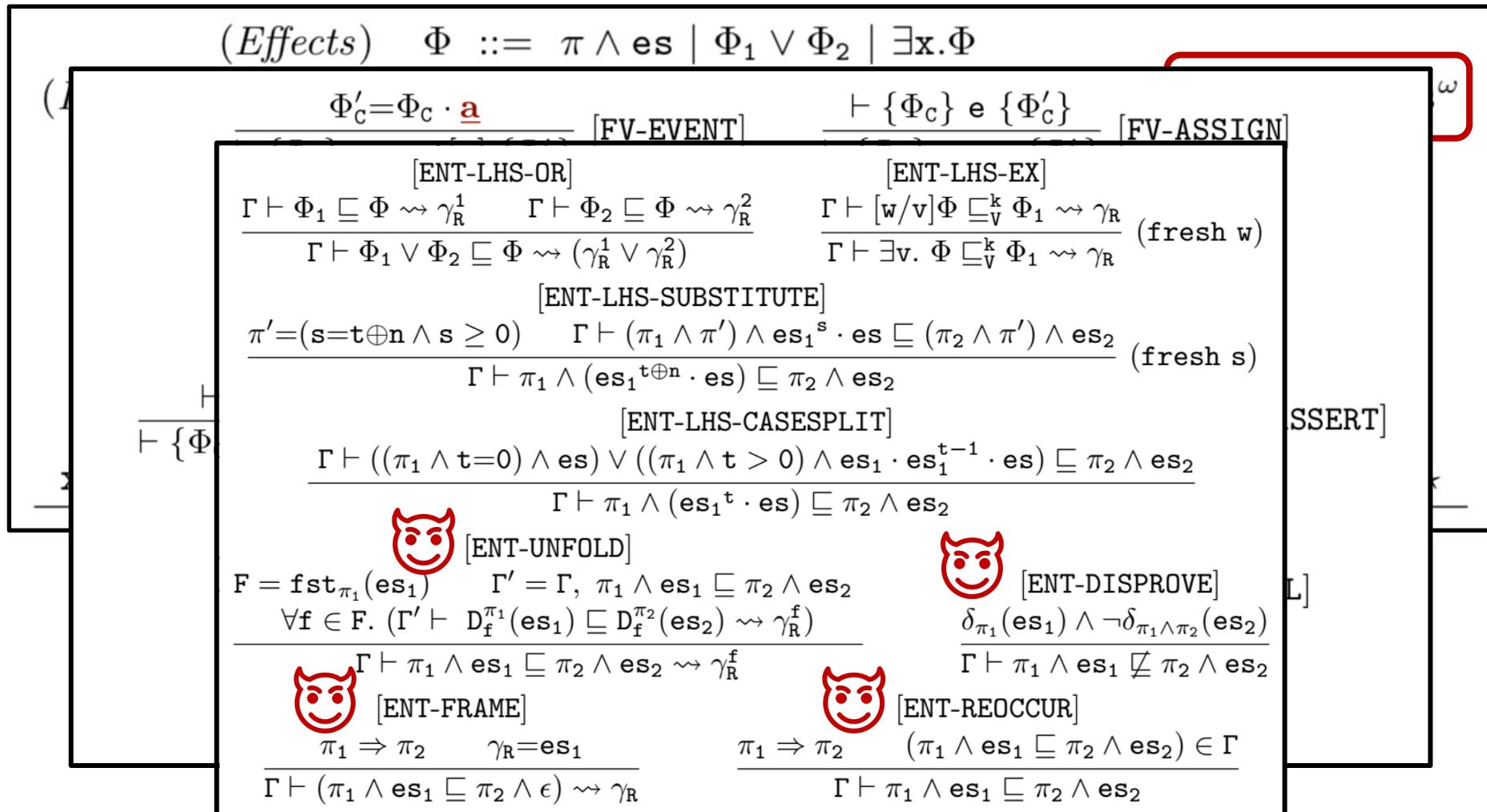
Term Rewriting System - Example

$$\begin{array}{c}
 \xrightarrow{\quad} (n_2 = n_1 - 1 \wedge n_2 \geq 0) \wedge \underline{\text{Send}}^{n_2} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_2} \cdot \underline{\text{Done}} (\ddagger) \text{ [REOCCUR]} \\
 \hline
 \frac{n_1 = 0 \wedge \epsilon \sqsubseteq \epsilon \text{ [FRAME]}}{n_1 = 0 \wedge \underline{\text{Done}} \sqsubseteq \underline{\text{Done}}} \quad \frac{n_1 > 0 \wedge \underline{\text{Send}}^{n_1-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_1-1} \cdot \underline{\text{Done}}}{n_1 > 0 \wedge \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}}} \text{ [UNFOLD]} \\
 \hline
 \frac{(n_1 = n-1 \wedge n_1 \geq 0) \wedge \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n_1} \cdot \underline{\text{Done}} (\ddagger)}{n > 0 \wedge \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}}} \text{ [CASESPLIT]} \\
 \hline
 \frac{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \underline{\text{Send}}^n \cdot \underline{\text{Done}}}{n > 0 \wedge \underline{\text{Send}} \cdot \underline{\text{Send}}^{n-1} \cdot \underline{\text{Done}} \sqsubseteq \Phi_{\text{post}}^{\text{send}(n)}} \text{ [SUBSTITUTE]}
 \end{array}$$

Formal Specification & Entailment Rules

(Effects) $\Phi ::= \pi \wedge e s \mid \Phi_1 \vee \Phi_2 \mid \exists x. \Phi$	
(1)	$\frac{\Phi'_c = \Phi_c \cdot \underline{a}}{\vdash \{\Phi_c\} \text{ event}[a] \{\Phi'_c\}} \text{ [FV-EVENT]} \quad \begin{matrix} \text{evil face icon} \\ \omega \end{matrix}$ $\frac{\vdash \{\Phi_c\} e_1 \{\Phi'_c\} \quad \vdash \{\Phi'_c\} e_2 \{\Phi''_c\}}{\vdash \{\Phi_c\} e_1; e_2 \{\Phi''_c\}} \text{ [FV-SEQ]}$ $\frac{\vdash \{v \wedge \Phi_c\} e_1 \{\Phi'_c\} \quad \vdash \{\neg v \wedge \Phi_c\} e_2 \{\Phi''_c\}}{\vdash \{\Phi_c\} \text{ if } v \text{ then } e_1 \text{ else } e_2 \{\Phi'_c \vee \Phi''_c\}} \text{ [FV-IF-ELSE]} \quad \begin{matrix} \text{evil face icon} \\ \omega \end{matrix}$ $\frac{\vdash \{\Phi_c\} e \{\Phi'_c\}}{\vdash \{\Phi_c\} \tau x; e \{\exists x. \Phi'_c\}} \text{ [FV-LOCAL]} \quad \frac{\vdash \text{rev}(\Phi_c) \sqsubseteq \text{rev}(\Phi) \rightsquigarrow \gamma_R}{\vdash \{\Phi_c\} \text{ assert } \Phi \{\Phi_c\}} \text{ [FV-ASSERT]} \quad \begin{matrix} \text{evil face icon} \\ \omega \end{matrix}$ $\tau \text{ mn } (\tau x)^* \{\text{requires } \Phi_{\text{pre}} \text{ ensures } \Phi_{\text{post}}\} \{e\} \in \mathcal{P}$ $\frac{\vdash \text{rev}(\Phi_c) \sqsubseteq \text{rev}([y^*/x^*]\Phi_{\text{pre}}) \rightsquigarrow \gamma_R \quad \Phi'_c = \Phi_c \cdot [y^*/x^*]\Phi_{\text{post}}}{\vdash \{\Phi_c\} \text{ mn}(y^*) \{\Phi'_c\}} \text{ [FV-CALL]} \quad \begin{matrix} \text{evil face icon} \\ \omega \end{matrix}$ $\frac{\Phi_c = \Phi_{\text{pre}} \quad \vdash \{\Phi_c\} e \{\Phi'_c\} \quad \vdash \Phi'_c \sqsubseteq \Phi_{\text{pre}} \cdot \Phi_{\text{post}}}{\vdash \tau \text{ mn } (\tau x)^* \{\text{requires } \Phi_{\text{pre}} \text{ ensures } \Phi_{\text{post}}\} \{e\}} \text{ [FV-METH]} \quad \begin{matrix} \text{evil face icon} \\ \omega \end{matrix}$

Formal Specification & Entailment Rules



Imp

- An op

- Bench

program

➤der

➤exp

➤we

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

olling

L invalid)

Implementation and Evaluation (Insights)

- When the transition states of the models are small, the average execution time spent by the TRS is even less than the NFAs construction time, which means it is not necessary to construct the NFAs when a TRS solves it faster;
- When the total states become larger, on average, the TRS outperforms automata-based algorithms, due to the significantly reduced search branches provided by the normalization lemmas; and
- For the invalid cases, the TRS disproves them earlier without constructing the whole NFAs.

B. Beyond Sequential Control Flow

→ 3. TimEffs: Time Critical Systems:

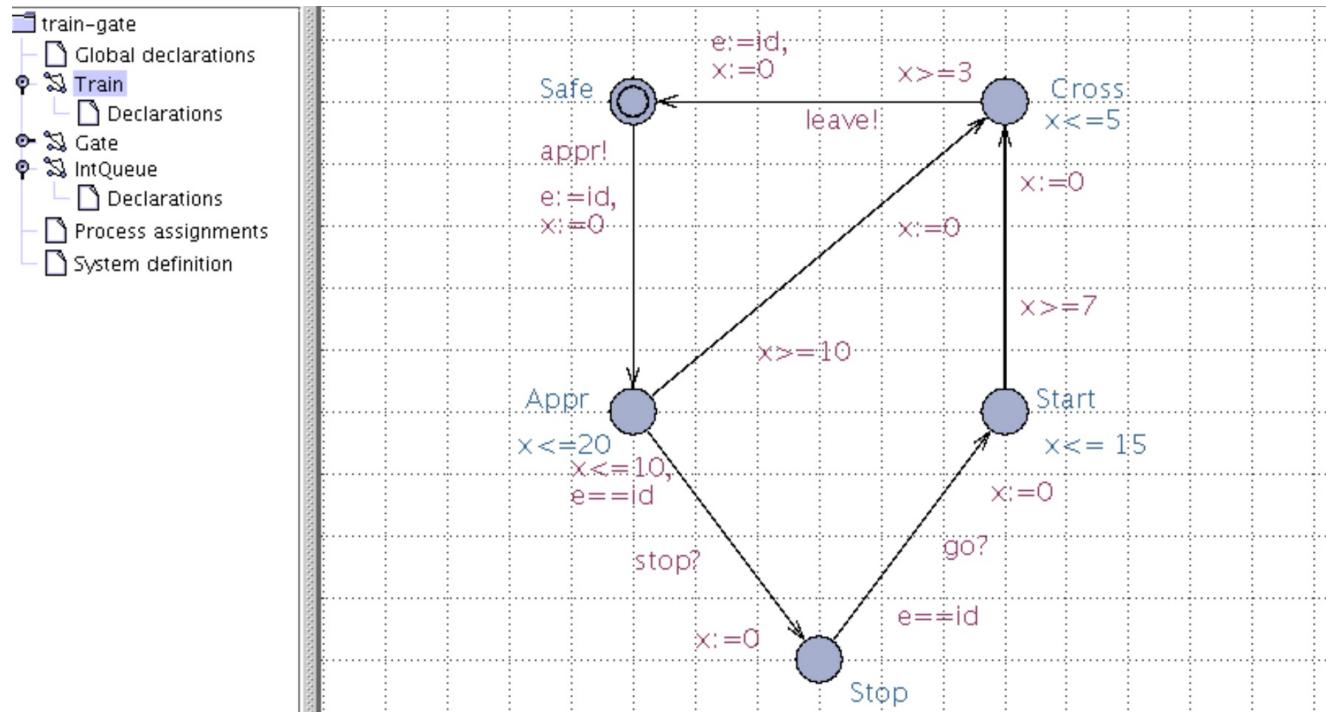
- mutable variables and concurrency
- timed behavioural patterns, such as delay, timeout, interrupt, deadline, etc.

4. ContEffs Algebraic Effects and Handlers

- The coexistence of zero-shot, one-shot and multi-shot continuations
- Non-terminating behaviours.

Timed Verification via Timed Automata

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



$$\Phi \triangleq 0 \leq t < 1 \wedge (_^\star \cdot \text{Done}) \# t$$

TimEfs - Symbolic Timed Automata

```
1 void addOneSugar()
2 /* req: true ∧ _ */
3   ens: t>1 ∧ ε # t */
4 { timeout ((), 1); }
5
6 void addNSugar (int n)
7 /* req: true ∧ _
8   ens: t≥n ∧ EndSugar#t */
9 { if (n == 0)
10   event ["EndSugar"];
11 else {
12   addOneSugar();
13   addNSugar (n-1);}}
14 void makeCoffee (int n)
15 /* req: n≥0 ∧ _ */
16   ens: n≤t≤5 ∧ t'≤4 ∧ */
17   (EndSugar # t) · (Coffee # t') */
18 { deadline (addNSugar(n), 5);
19  deadline (event ["Coffee"], 4)}
20
21 int main ()
22 /* req: true ∧ ε
23   ens: t≤9 ∧ ((!Done)* # t) · Done*/
24 { event ["CupReady"];
25  makeCoffee (3);
26  event ["Done"];}
```

Expressiveness of TimEffs

TimEffs draw similarities to Metric Temporal Logic (MTL), derived from LTL, where a set of non-negative real numbers is added to temporal modal operators. Basic operators are:

Φ_{post}	$\Box_I A \equiv (A^*)\#t$	$\Diamond_I A \equiv (_^* \cdot A)\#t$	$\bigcirc_I A \equiv (_) \#t \cdot A$	$A U_I B \equiv (A^*)\#t \cdot B$
Φ_{pre}	$\overleftarrow{\Box}_I A \equiv (A^*)\#t$	$\overleftarrow{\Diamond}_I A \equiv (A \cdot _^*)\#t$	$\ominus_I A \equiv A \cdot ((_) \#t)$	$A S_I B \equiv B \cdot ((A^*)\#t)$

\Box - “globally”; \Diamond - “finally”; \bigcirc - “next”; U – “until”, and their past time-reversed versions: $\overleftarrow{\Box}$; $\overleftarrow{\Diamond}$; and \ominus for “previous”; S for “since”.

Inclusion Checking – SMT based Term Rewriting

```

7 void addNSugar (int n)
8 /* req: true ∧ ...
9   ens: t ≥ n ∧ EndSugar # t */
10 { if (n == 0) { event ["EndSugar"];}
11   else {
12     addOneSugar();
13     addNSugar (n-1);}}

```

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

$$\begin{array}{l}
 (I) \quad t2 > 1 \wedge tL \geq (n-1) \wedge tL = (tR - t2) \Rightarrow tR \geq n \\
 \frac{}{n \neq 0 \wedge t2 > 1 \wedge tL \geq (n-1) \wedge \epsilon \sqsubseteq tR \geq n \wedge \epsilon} \text{⑦ [PROVE]} \\
 \frac{}{n \neq 0 \wedge t2 > 1 \wedge tL \geq (n-1) \wedge ES \# tL \sqsubseteq tR \geq n \wedge \cancel{ES \# (tR - t2)}} \text{⑥ [UNFOLD] } \pi_u : tL = (tR - t2) \\
 \frac{}{n \neq 0 \wedge t2 > 1 \wedge tL \geq (n-1) \wedge \epsilon \# t2 \cdot ES \# tL \sqsubseteq tR \geq n \wedge ES \# tR} \text{⑤ [UNFOLD]}
 \end{array}$$

Target Language C^t, imperative with timed constructs:

(Program)	$\mathcal{P} ::= (\alpha^*, meth^*)$			
(Types)	$\iota ::= int \mid bool \mid void$			
(Method)	$meth ::= \iota\ mn\ (\iota\ x)^*\ \{\mathbf{req}\ \Phi_{pre}\ \mathbf{ens}\ \Phi_{post}\}\ \{e\}$			
(Values)	$v ::= () \mid c \mid b \mid x$			
(Assignment)	$\alpha ::= x := v$			
(Expressions)	$e ::= v \mid \alpha \mid [v]e \mid mn(v^*) \mid e_1; e_2 \mid e_1 e_2 \mid if\ v\ e_1\ e_2 \mid \mathbf{event}[\mathbf{A}(v, \alpha^*)]$ $\mid \mathbf{delay}[v] \mid e_1\ \mathbf{timeout}[v]\ e_2 \mid e\ \mathbf{deadline}[v] \mid e_1\ \mathbf{interrupt}[v]\ e_2$			
(Terms)	$t ::= c \mid x \mid t_1 + t_2 \mid t_1 - t_2$			
	$c \in \mathbb{Z}$	$b \in \mathbb{B}$	$mn, x \in \mathbf{var}$	(Action labels) $\mathbf{A} \in \Sigma$

Specification Language TimEffs:

(Timed Effects)	$\Phi ::= \pi \wedge \theta \mid \Phi_1 \vee \Phi_2$			
(Event Sequences)	$\theta ::= \perp \mid \epsilon \mid ev \mid \theta_1 \cdot \theta_2 \mid \theta_1 \vee \theta_2 \mid \underline{\theta_1 \parallel \theta_2} \mid \pi? \theta \mid \theta \# t \mid \theta^*$			
(Events)	$ev ::= \mathbf{A}(v, \alpha^*) \mid \tau(\pi) \mid \overline{\mathbf{A}} \mid -$			
(Pure)	$\pi ::= True \mid False \mid bop(t_1, t_2) \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \neg \pi \mid \pi_1 \Rightarrow \pi_2$			
(Real-Time Terms)	$t ::= c \mid x \mid t_1 + t_2 \mid t_1 - t_2$			
	$c \in \mathbb{Z}$	$x \in \mathbf{var}$	(Real Time Bound) #	(Kleene Star) *

Implementation and Evaluation

Table 5.3: Experimental Results for Manually Constructed Synthetic Examples.

No.	LOC	Forward(ms)	#Prop(✓)	Avg-Prove(ms)	#Prop(X)	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

Observations:

- i. proving/disproving time ↗ if the effect computation time ↗
- ii. While the number of querying Z3 per property ($\#AskZ3 / (\#Prop(\checkmark) + \#Prop(X))$) ↗, the proving/disproving time ↗
- iii. the disproving times for invalid properties are constantly lower than the proving process.

Implementation and

Table 5.4: Comparison with PAT via verify

#Proc	Prove(s)	#AskZ3-u	Disprove
2	0.09	31	0.110
3	0.21	35	0.093
4	0.46	63	0.120
5	25.0	84	0.128

Observations:

- automata-based model checkers need values for constants d and e;
- our proposal can symbolically prove safety properties;
- our verification time largely depends on the number of processes.

```
1 var x := -1;
2 var cs:= 0;
3
4 void proc (int i) {
5     [x=-1] //block waiting until true
6     deadline(event["Update"](i)){x:=i},d);
7     delay (e);
8     if (x==i) {
9         event["Critical"](i){cs:=cs+1};
10        event["Exit"](i){cs:=cs-1;x:=-1};
11        proc (i);
12    } else {proc (i);}
13
14 void main ()
15 /* req:  d<e ∧ e
16    ensa:true ∧ (cs≤1)*   ensb:true ∧ ((_*) . Critical . Exit . (_*)) */
17 { proc(0) || proc(1) || proc(2); }
```

Figure 5.4: Fischer's mutually exclusion algorithm.

B. Beyond Sequential Control Flow

3. TimEffs: Time Critical Systems:

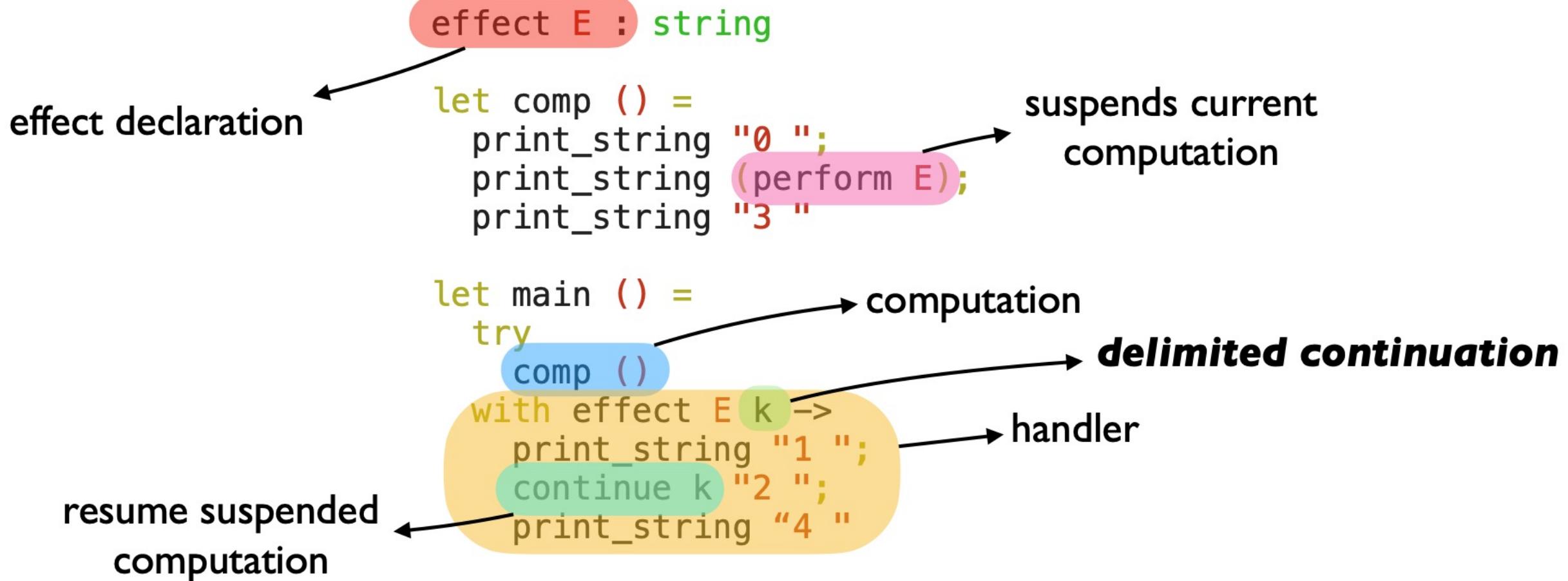
- mutable variables and concurrency
- timed behavioural patterns, such as delay, timeout, interrupt, deadline, etc.

→ 4. ContEffs Algebraic Effects and Handlers

- The coexistence of zero-shot, one-shot and multi-shot continuations
- Non-terminating behaviours.

Effect Handlers

It prints: 0 1 2 3 4



Effect Handlers – We can achieve:

- The coexistence of zero-shot, one-shot and multi-shot continuations. Should it be permitted or forbidden to invoke a captured continuation more than once?
- Non-terminating behaviours with higher-order effect signatures and deep handlers.
- Linear temporal properties. The behaviour is determined by the encompassing handlers.

```
1  effect Foo : (unit -> unit)
2
3  let f() = perform Foo ()
4
5  let loop()
6  = match f () with
7  | _ -> () (* normal return *)
8  | effect Foo k -> continue k (fun () -> perform Foo ())
```

More examples with specifications

```
1  effect Exc: unit
2  effect Other: unit
3
4  let raise ()           (*@ req _^* @*)
5  (*@ ens Exc!.Other! @*)
6
7  = perform Exc;
8    perform Other
9
10 let exHandler          (*@ req _^* @*)
11 (*@ ens Exc @*)
12
13 = match raise () with
14 | _ -> (* Abandoned *)
15 | effect Exc k -> ()
```

Fig. 10. Encoding Exceptions.

```
1  effect Goo : (unit -> unit)
2
3  let f_g ()             /*@ req _^* @*/
4
5  /*@ ens Foo!.Goo!.Foo?() @*/
6  = let f = perform Foo in
7    let g = perform Goo in
8    f () (* g is abandoned *)
9
10 let loop ()             /*@ req _^* @*/
11
12 /*@ ens _^*. (Foo.Goo)^w @*/
13 = match f_g () with
14 | _ -> ()
15 | effect Foo k -> continue k (fun () -> perform Goo ())
16 | effect Goo k -> continue k (fun () -> perform Foo ())
```

Target Language λ_h , pure, higher-order, call by value

(Program)	$\mathcal{P} ::= eff^* \ meth^*$
(Effect Declarations)	$eff ::= \textcolor{red}{A} : \tau$
(Method Definition)	$meth ::= \tau \ mn \ (\tau \ v) \ [\mathbf{req} \ \Phi_{pre} \ \mathbf{ens} \ \Phi_{post}] \ \{e\}$
(Types)	$\tau ::= \text{bool} \mid \text{int} \mid \text{unit} \mid \tau_1 \rightarrow \tau_2$
(Values)	$v ::= c \mid x \mid \lambda x \Rightarrow e$
(Handler)	$h ::= (\text{return } x \mapsto e \mid ocs)$
(Operation Cases)	$ocs ::= \emptyset \mid \{\text{effect } \textcolor{red}{A}(x, \kappa) \mapsto e\} \uplus ocs$
(Expressions)	$e ::= v \mid v_1 \ v_2 \mid \text{let } x=v \text{ in } e \mid \text{if } v \text{ then } e_1 \text{ else } e_2 \mid$ $\text{perform } \textcolor{red}{A}(v, \lambda x \Rightarrow e) \mid \text{match } e \text{ with } h \mid \text{resume } v$

Specification Language ContEffs:

(ContEffs)	$\Phi ::= \bigvee(\pi, \theta, v)$
(Parameterized Label)	$\textcolor{red}{l} ::= \Sigma(v)$
(Event Sequences)	$\theta ::= \perp \mid \epsilon \mid ev \mid \underline{Q} \mid \theta_1 \cdot \theta_2 \mid \theta_1 \vee \theta_2 \mid \theta^* \mid \theta^\infty \mid \theta^\omega$
(Single Events)	$ev ::= _ \mid \textcolor{red}{l} \mid \bar{l}$
(Placeholders)	$\underline{Q} ::= \textcolor{red}{l}! \mid \textcolor{red}{l}?(v)$
(Pure formulae)	$\pi ::= \text{True} \mid \text{False} \mid R(t_1, t_2) \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \neg \pi \mid \pi_1 \Rightarrow \pi_2$
(Terms)	$t ::= n \mid x \mid t_1 + t_2 \mid t_1 - t_2$

$x \in \mathbf{var}$

(Finite Kleene Star) \star

(Finite/Infinite) ∞

(Infinite) ω

Proposals Overview

	Target Language	Specification Language	Applied Domain	Publication
1	C	DependentEffs	General Effectful Programs	[ICFEM 2020]
2	Imp ^{a/s}	SyncEffs	Reactive Systems	[VMCAI 2021]
3	C ^t	TimEffs	Time Critical Systems	Under submission
4	λ_h	ContEffs	Algebraic Effects and Handlers	[APLAS 2022]

Main Challenges

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

Summary & Status

- New framework for temporal verification.
 - ❖ More expressiveness specifications.
 - ❖ Fine-grained, semantics oriented, forward verifiers.
 - ❖ Term-rewriting systems.
- Implementations upon possible application scenes and evaluations.
 - ❖ General Effectful Programs [ICFEM 2020]
 - ❖ Reactive Systems [VMCAI 2021]
 - ❖ Time Critical Systems (Under Submission)
 - ❖ Algebraic Effects and Handlers [APLAS 2022]

Future Work

- Temporal Verification with Spatial Information
- Temporal Verification with Incorrectness Logic
- Program Synthesis via More Expressive Temporal logics

Thank you for
your attention!

Related Links

1. DependentEffs: [PDF] [Video] <https://github.com/songyahui/EFFECTS>
2. ASyncEffs: [PDF] [Video] <https://github.com/songyahui/SyncedEffects>
https://github.com/songyahui/Semantics_HIPHOP
3. TimEffs: [PDF] https://github.com/songyahui/Timed_Verification
4. ContEffs: [PDF] <https://github.com/songyahui/AlgebraicEffect>