Automated Temporal Verification with Extended Regular Expressions

PhD Candidate: Yahui Song
Supervisor: 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

Expressiveness is limited by the finite-state automata
To be bounded due to the lack of symbolic proving

Manual/CSP Modelling
A verified model
≈
A verified implementation

Temporal Logic
Property $\Phi$

MODEL CHECKER

Yes, Property $\Phi$ is true
No, Property $\Phi$ is not true
& counterexamples
A New Framework for Temporal Verification

- A verified implementation;
- Flexible specifications, which can be combined with other logic;
A New Framework for Temporal Verification

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

+ A program with Temporal Specifications
+ Two Effects (LHS $\subseteq$ RHS)

The program is verified? The language inclusion is valid?

Terms and Effect Systems (Types-and-Effect Systems)

(Term Rewriting System)
A New Framework for Temporal Verification

+ A verified implementation;
+ Flexible specifications, which can 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 can be combined with other logic;
+ Efficient symbolic entailment checker with (co-)inductive proofs;
- Automation/Decidability.
Automata vs. RE: $\Sigma^* \subseteq L(A)$

Flexibility and Efficiency

- Init/Next
- Processed
- Rejecting
Flexibility and Efficiency

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

Antimirov algorithm for solving REs’ inclusions

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

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

$$r \subseteq s \iff \forall a \in \Sigma. A^{-1}(r(A^{-1}(s))$$
Automata vs. RE: $\Sigma^* \subseteq L(A)$

Flexibility and Efficiency

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

$(a \lor b)^* \subseteq (a \lor b \lor bb)^* \quad \text{(Reoccur)}$

$\varepsilon \cdot (a \lor b)^* \subseteq \varepsilon \cdot (a \lor b \lor bb)^* \quad \text{(Reoccur)}$

$a \cdot (a \lor b)^* \subseteq (a \lor b \lor bb)^*$

$b \cdot (a \lor b)^* \subseteq \ldots$

$(a \lor b)^* \subseteq (a \lor b \lor bb)^*$
# Proposals Overview

<table>
<thead>
<tr>
<th>Target Language</th>
<th>Specification Language</th>
<th>Applied Domain</th>
<th>Research Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>IntegratedEffs</td>
<td>General Effectful Programs</td>
<td>(ICFEM 2020)</td>
</tr>
<tr>
<td>\text{Imp}^a/s</td>
<td>SyncEffs</td>
<td>Synchronous Programming</td>
<td>(VMCAI 2021)</td>
</tr>
<tr>
<td>C^t</td>
<td>TimEffs</td>
<td>Time Critical Systems</td>
<td>(TACAS 2023)</td>
</tr>
<tr>
<td>\lambda_h</td>
<td>ContEffs</td>
<td>Algebraic Effects and Handlers</td>
<td>(APLAS 2022)</td>
</tr>
</tbody>
</table>

## 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.
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

\[ \Phi' = (\text{Send}^* \cdot \text{Done}, \text{Send}^\omega) \quad [\text{Hofmann, Martin, and Wei Chen. 2014}] \]

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

send n =

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

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

Φ' = (Send* · Done, Sendω) [Hofmann, Martin, and Wei Chen. 2014]
Φ'' = (Send^n · Done, Sendω) [Nanjo, Yoji, et al. 2018]
Φ_pre = True ∧ Ready · _*
Φ_post(n) = n ≥ 0 ∧ (Send^n · Done) ∨ n < 0 ∧ (Sendω)
Integrated Dependent Effects

\[
\Phi' = (\text{Send}^* \cdot \text{Done}, \text{Send}^\omega) \\
\Phi'' = (\text{Send}^n \cdot \text{Done}, \text{Send}^\omega) \\
\Phi_{\text{pre}} = \text{True} \land \text{Ready} \cdot _* \\
\Phi_{\text{post}}(n) = n \geq 0 \land (\text{Send}^n \cdot \text{Done}) \lor n < 0 \land (\text{Send}^\omega)
\]

\[
\Phi_{\text{pre}} = n \geq 0 \land \epsilon \\
\Phi_{\text{post}}(n) = n \geq 0 \land (\text{Ready} \cdot \text{Send}^n \cdot \text{Done})^\omega \\
\Phi'_{\text{pre}} = \text{True} \land \epsilon \\
\Phi'_{\text{post}}(n) = n \geq 0 \land (\text{Ready} \cdot \text{Send}^n \cdot \text{Done})^\omega \\
\lor n < 0 \land (\text{Ready} \cdot \text{Send}^\omega)
\]
Integrated Dependent Effects

1. Aware of termination (mixed definition): \((n \geq 0 \land \text{Send}^n \cdot \text{Done}) \lor (n < 0 \land \text{Send}^\omega)\)

2. Beyond the context-free grammar: \(a^n \cdot b^n \cdot c^n\)

3. Effects in precondition is new: \(\Phi_{\text{pre}} = \text{True} \land \text{Ready} \cdot _*\)

4. Undetermined termination (Kleene Star): \(\text{True} \land \text{Send}^* \cdot \text{Done}\)
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

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.

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

- An open-source 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 T.r.s respectively.
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

• 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.

```c
1 signal S1 {
2   present S1
3   then nothing
4   else emit S1
5 }
```

(a) No valid assignments (Logically incorrect).

```c
1 signal S1 {
2   present S1
3   then emit S1
4   else nothing
5 }
```

(b) Two possible assignments (Logically incorrect).

```c
1 /*@ requires S1 @*/
2 /*@ ensures S1 @*/
3 present S1
4 then emit S1
5 else nothing
```

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

[Berry G, Gonthier G. 1992]
[Jagadeesan L J, Puchol C, Von Olnhausen J E. 1995]
[Florence, Spencer P., et al. 2019]
Target Language $\lambda^{a/s}$, extending Esterel with synchronous constructs

(Program) \[ P ::= \text{module} \]

(Basic Types) \[ \tau ::= \text{IN} | \text{OUT} | \text{INOUT} \]

(Module Def.) \[ \text{module} ::= nm (\tau \xrightarrow{\delta}) \langle \text{req} \Phi_{\text{pre}} \text{ens} \Phi_{\text{post}} \rangle p \]

(Statement) \[ p, q ::= \text{nothing} | \text{pause} | \text{emit} S | p ; q | p | q | \text{loop} p \]
\[ | \text{signal} S \text{ in } p | \text{present } S \text{ then } p \text{ else } q | \text{call mn } (\delta) \]
\[ | \text{try } p \text{ with } q | \text{raise } d | \text{async } S \ p \ q | \text{await } S \]

(Signal Variables) \[ S \in \Sigma \]

(x, mn \in \text{var})

(Depth) \[ d \in \mathbb{N} \cup \{0\} \]

Specification Language SyncEffs:

(Effects) \[ \Phi ::= \bot | \epsilon | I \ | \ S? \ | \Phi_1 \cdot \Phi_2 \ | \Phi_1 \lor \Phi_2 \ | \Phi_1 || \Phi_2 \ | \Phi^* \]

(event) \[ I ::= \{\} | \{S \rightarrow \alpha\} | I_1 \cup I_2 \]

(Signal Status) \[ \alpha ::= \text{present} | \text{absent} | \text{undef} \]

(Signal Variables) \[ S \in \Sigma \]

(Blocking Waiting) \[ ? \]

(Kleene Star) \[ * \]
Logically incorrect examples, caught by SyncEffs.

Constructiveness

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

---

1. `present S1`   
   `{{S1 \mapsto \text{undef}}}`

2. `then`       
   `{{S1 \mapsto \text{undef}, S1 \mapsto \text{present}}}`

3. `nothing`    
   `{{S1 \mapsto \text{undef}, S1 \mapsto \text{present}}}`

4. `else`           
   `{{S1 \mapsto \text{undef}, S1 \mapsto \text{absent}}}`

5. `emit S1`   
   `{{S1 \mapsto \text{present}, S1 \mapsto \text{absent}}}`

(a) `\{S1 \mapsto \text{undef}, S1 \mapsto \text{present}\} \lor \{S1 \mapsto \text{present}, S1 \mapsto \text{absent}\}`

(b) `\{S1 \mapsto \text{present}\} \lor \{S1 \mapsto \text{absent}\}`

---

1. `module a_bug:`

2. `output S;`

3. `/*@`

4. `require {}`

5. `ensure {S}`

6. `@*/`

7. `signal S in`

8. `present S then emit S`

9. `else emit S`

10. `end present end signal`

11. `end module`
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.
We propose TimEffs - Symbolic Timed Automata

```c
void addOneSugar()
{
    /* req: true ∧ _ */
    en: t>1 ∧ ε # t */
    { timeout (((), 1); }
}
```
Target Language C\textsuperscript{t}, imperative with timed constructs:

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

\[
\begin{array}{c|c|c|c}
\text{c} \in \mathbb{Z} & \text{b} \in \mathbb{B} & \text{mn, x} \in \text{var} & (Action \ labels) \ A \in \Sigma
\end{array}
\]

Specification Language TimEffs:

\[
(Timed \ Effects) \quad \Phi ::= \pi \land \theta \mid \Phi_1 \lor \Phi_2
\]

\[
(Event \ Sequences) \quad \theta ::= \bot \mid \epsilon \mid ev \mid \theta_1 \cdot \theta_2 \mid \theta_1 \lor \theta_2 \mid \theta_1 \parallel \theta_2 \mid \pi?\theta \mid \theta#t \mid \theta^*
\]

\[
(Pure) \quad \pi ::= \text{True} \mid \text{False} \mid \text{bop}(t_1, t_2) \mid \pi_1 \land \pi_2 \mid \pi_1 \lor \pi_2 \mid \neg \pi \mid \pi_1 \Rightarrow \pi_2
\]

\[
(Real-Time \ Terms) \quad t ::= c \mid x \mid t_1 + t_2 \mid t_1 - t_2
\]

\[
\begin{array}{c|c|c}
c \in \mathbb{Z} & x \in \text{var} & (Real \ Time \ Bound) \ # \quad (Kleene \ Star) \ *
\end{array}
\]
Inclusion Checking – SMT based Term Rewriting

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

\[(n=0 \land ES) \lor (n \neq 0 \land t \geq 1 \land (\epsilon \ # t2) \cdot \Phi_{\text{post}}^{\text{addNSugar}(n-1)}) \subseteq \Phi_{\text{post}}^{\text{addNSugar}(n)}\]
Inclusion Checking – SMT based Term Rewriting

```c
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);}}
```
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);}}
Inclusion Checking – SMT based Term Rewriting

Succeed!

```c
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);}}
```

\[ n=0 \land \epsilon \subseteq t_R \geq 0 \land \epsilon \neq t_R \]

\[ n=0 \land ES \subseteq t_R \geq 0 \land ES \# t_R \]

\[(n=0 \land ES) \lor (n \neq 0 \land t_2 > 1 \land t_L \geq (n-1) \land \epsilon \# t_2 \cdot ES \# t_L) \subseteq t_R \geq n \land ES \# t_R \]

\[ (n=0 \land ES) \lor (n \neq 0 \land t_2 > 1 \land (\epsilon \# t_2) \cdot P_{\text{addNSugar}}(n-1) \subseteq P_{\text{addNSugar}}(n) \]

\[ t_2 > 1 \land t_L \geq (n-1) \land t_L = (t_R - t_2) \Rightarrow t_R \geq n \]

\[ n \neq 0 \land t_2 > 1 \land t_L \geq (n-1) \land \epsilon \subseteq t_R \geq n \land \epsilon \]

\[ n \neq 0 \land t_2 > 1 \land t_L \geq (n-1) \land ES \# t_L \subseteq t_R \geq n \land ES \# (t_R - t_2) \]

\[ n \neq 0 \land t_2 > 1 \land t_L \geq (n-1) \land \epsilon \# t_2 \cdot ES \# t_L \subseteq t_R \geq n \land ES \# t_R \]
Antimirov algorithm for solving REs’ inclusions

Definition 1 (Derivatives). Given any formal language \( S \) over an alphabet \( \Sigma \) 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 \( s \),

\[
r \leq s \iff \forall A \in \Sigma. A^{-1}(r) \leq A^{-1}(s).
\]

Antimirov algorithm for solving TimEffs’ inclusions

Definition 3 (TimEffs Inclusion). For TimEffs \( \Phi_1 \) and \( \Phi_2 \),

\[
\Phi_1 \subseteq \Phi_2 \iff \forall A \in \Sigma. \forall t \geq 0. (A#t)^{-1} \Phi_1 \subseteq (A#t)^{-1} \Phi_2.
\]
### Implementation and Evaluation

Table 5.3: Experimental Results for Manually Constructed Synthetic Examples.

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

**Main Observations:**
the disproving times for invalid properties are constantly lower than the proving process.
Evaluation – Fischer’s Mutual Exclusion Algorithm

```c
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);}
8. }
9.
10. void main ()
11. /* req: d<e ∧ e
12.  ensₐ:true ∧ (cs≤1)*  ensₐ:true ∧ (((*)_Critical.Exit(_*)))* */
13. { 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

<table>
<thead>
<tr>
<th>#Proc</th>
<th>Prove(s)</th>
<th>#AskZ3-u</th>
<th>Disprove(s)</th>
<th>#AskZ3-u</th>
<th>PAT(s)</th>
<th>Uppaal(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.09</td>
<td>31</td>
<td>0.110</td>
<td>37</td>
<td>≤0.05</td>
<td>≤0.09</td>
</tr>
<tr>
<td>3</td>
<td>0.21</td>
<td>35</td>
<td>0.093</td>
<td>42</td>
<td>≤0.05</td>
<td>≤0.09</td>
</tr>
<tr>
<td>4</td>
<td>0.46</td>
<td>63</td>
<td>0.120</td>
<td>47</td>
<td>0.05</td>
<td>0.09</td>
</tr>
<tr>
<td>5</td>
<td>25.0</td>
<td>84</td>
<td>0.128</td>
<td>52</td>
<td>0.15</td>
<td>0.19</td>
</tr>
</tbody>
</table>

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

```ocaml
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 "
```

[de Vilhena, Paulo Emílio, and François Pottier. 2021]
[Sivaramakrishnan, K. C., et al. 2021]
User-defined Effects and Handlers

This prints: 0 1 2 3 4

Example taken from “Effect Handlers in Multicore OCaml” slides by KC Sivaramakrishnan.
Core Language $\lambda h$: pure, higher-order, call by value

(V) $v ::= c \mid x \mid \lambda x \Rightarrow e$

(Expr) $e ::= v \mid v_1 \cdot v_2 \mid \text{let } x=v \text{ in } e \mid \text{if } v \text{ then } e_1 \text{ else } e_2 \mid \text{perform } A(v, \lambda x \Rightarrow e) \mid \text{match } e \text{ with } h \mid \text{resume } v$

Specification Language ContEffs

(ContEffs) $\Phi ::= \bigvee (\pi, \theta, v)$

(Param Label) $l ::= \Sigma (v)$

(Event Seq) $\theta ::= \perp \mid \epsilon \mid ev \mid Q \mid \theta_1 \cdot \theta_2 \mid \theta_1 \vee \theta_2 \mid \theta^* \mid \theta^\infty \mid \theta^\omega$

(Single Ev) $ev ::= \_ \mid l \mid \bar{l}$

(Placeholder) $Q ::= l! \mid l?(v)$
Examples – Zero-shot continuations (Exceptions)

```ml
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 -> ()
```

<table>
<thead>
<tr>
<th>Step</th>
<th>History</th>
<th>Current Event</th>
<th>Continuation</th>
<th>Bindings</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>emp</td>
<td>Exc!</td>
<td>Other! · Other?() · Exc?() · ‡</td>
<td>‡ = (fun x -&gt; x)</td>
</tr>
<tr>
<td>2</td>
<td>Exc</td>
<td>-</td>
<td>-</td>
<td>No “Continue”</td>
</tr>
<tr>
<td>Final</td>
<td>Exc</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
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 () -> ());
  | 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

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

• 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”

Thank you for your attention!


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).