Automated Temporal Verification of Integrated Dependent Effects

Yahui Song and Wei-Ngan Chin

School of Computing, NUS

@ICFEM2020, 2nd March 2021

Automated Temporal Verification of Integrated Dependent Effects

Yahui Song and Wei-Ngan Chin

School of Computing, NUS

@ICFEM2020, 2nd March 2021

Temporal Effects (Regular Expressions)

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

Temporal Effects (Regular Expressions)

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

Φ' = (Send* · Done, Send$^{\omega}$)   *[Martain 2014]*

Temporal Effects (Regular Expressions)

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

Φ' = ($Send^* \cdot Done$, $Send^\omega$) *[Martain 2014]*

Φ'' = ($Send^n \cdot Done$, $Send^\omega$) *[Yoji 2018]*

Temporal Effects (Regular Expressions)

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

Φ' = (Send* · Done, $Send^{\omega}$) *[Martain 2014]*

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

Φ_{pre} = True ∧ Ready · _*

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

Temporal Effects (Regular Expressions)

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

Φ_{then} = n = 0 ∧ Done

Φ_{else} = n /= 0 ∧ Send · Φ_{post} (n-1)

$\Phi_{if\text{-}else}$ = Φ_{then} ∨ Φ_{else}

Φ' = ($Send^*$ · Done, $Send^{\omega}$) *[Martain 2014]*

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

Φ_{pre} = True ∧ Ready · _*

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

Temporal Effects (Regular Expressions)

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

Φ_{then} = n = 0 ∧ Done

Φ_{else} = n /= 0 ∧ Send · Φ_{post} (n-1)

$\Phi_{if\text{-}else}$ = Φ_{then} ∨ Φ_{else}

Goal: $\Phi_{if\text{-}else} \sqsubseteq \Phi_{post}(n)$

Φ' = (Send* · Done, $Send^{\omega}$) *[Martain 2014]*

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

Φ_{pre} = True ∧ Ready · _*

$\Phi_{post}(n)$ = (n ≥ 0 ∧ $Send^{n}$ · Done) ∨ (n < 0 ∧ $Send^{\omega}$)

(n = 0 ∧ Done) ∨ (n > 0 ∧ Send · $Send^{n-1}$ · Done) ∨ (n < 0 ∧ $Send^{\omega}$) ⊑ (n ≥ 0 ∧ $Send^{n}$ · Done) ∨ (n < 0 ∧ $Send^{\omega}$)

Temporal Effects (Regular Expressions)

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

Φ_{then} = $n = 0 \wedge Done$

Φ_{else} = $n \,/\!= 0 \wedge Send \cdot \Phi_{post}(n-1)$

$\Phi_{if\text{-}else}$ = $\Phi_{then} \vee \Phi_{else}$

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

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

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

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

Goal: $\Phi_{if\text{-}else} \sqsubseteq \Phi_{post}(n)$

- Mix Finite & Infinite traces
- Branching Properties

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

Regular Expressions Containment Problem

A : a finite set of alphabet

E = ϕ | emp | $\underline{a}$ | E ∨ E | E · E

For r, s ∈ E, to check if r ≤ s is valid

Translation of r, s into DFA/ NFA
(gives rise to an exponential blow-up)

Symbolic decision procedure
i.e. a term rewriting system (TRS)

Regular Expressions Containment Problem

PSPACE-complete

A : a finite set of alphabet

E = ϕ | emp | <u>a</u> | E ∨ E | E · E

For r, s ∈ E, to check if r ≤ s is valid

Translation of r, s into DFA/ NFA
(gives rise to an exponential blow-up)

Symbolic decision procedure
i.e. a term rewriting system (TRS)
(The worse-case complexity is still exponential)

Regular Expressions Containment Problem

PSPACE-complete

A : a finite set of alphabet

E = ϕ | emp | $\underline{a}$ | E ∨ E | E · E

For r, s ∈ E, to check if r ≤ s is valid

Translation of r, s into DFA/ NFA
(gives rise to an exponential blow-up)

Symbolic decision procedure
i.e. a term rewriting system (TRS) ✓
(The worse-case complexity is still exponential)

Goal: $\Phi_{\text{if-else}} \sqsubseteq \Phi_{\text{post}}(n)$

- Mix Finite & Infinite traces
- Branching Properties

Overview (1)

The Effects Logic – as the specification language

- Specify the temporal properties into the pre/post condition.

(a) Source Code

```
void send (int n){
    if (n==0) {
        event["Done"];
    }else{
        event["Send"]; send (n-1);
    }}
void server (int n){
    event["Ready"];
    send(n);
    server(n);}
```

(b) Effects Specifications

$$\Phi_{pre}^{send(n)} \triangleq True \wedge \underline{Ready} \cdot \_^{\star}$$

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

$$\Phi_{pre}^{server(n)} \triangleq n \geq 0 \wedge \epsilon$$

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

Overview (1)

The Effects Logic – as the specification language

- Specify the temporal properties into the pre/post condition.

(a) Source Code

```
void send (int n){
    if (n==0) {
        event["Done"];
    }else{
        event["Send"]; send (n-1);
    }}
void server (int n){
   event["Ready"];
   send(n);
   server(n);}
```

(b) Effects Specifications

$$\Phi_{\mathtt{pre}}^{\mathtt{send(n)}} \triangleq \mathtt{True} \wedge \underline{\mathbf{Ready}} \cdot \_^{\star}$$

$$\Phi_{\mathtt{post}}^{\mathtt{send(n)}} \triangleq (\mathtt{n} \geq 0 \wedge \underline{\mathbf{Send}}^{\mathtt{n}} \cdot \underline{\mathbf{Done}}) \vee (\mathtt{n} < 0 \wedge \underline{\mathbf{Send}}^{\omega})$$

$$\Phi_{\mathtt{pre}}^{\mathtt{server(n)}} \triangleq \mathtt{n} \geq 0 \wedge \epsilon$$

$$\Phi_{\mathtt{post}}^{\mathtt{server(n)}} \triangleq \mathtt{n} \geq 0 \wedge (\underline{\mathbf{Ready}} \cdot \underline{\mathbf{Send}}^{\mathtt{n}} \cdot \underline{\mathbf{Done}})^{\omega}$$

Overview (2)

The Forward Verifier – To accumulate the effects

1) `void send (int n){` *(– initialize the current effects state –)*

$\{\Phi_C = \Phi^{send(n)}_{pre} = True \wedge \underline{\textbf{Ready}} \cdot \_^\star\}$ [FV-Meth]

2) `if(n==0){`

$\{n=0 \wedge \underline{\textbf{Ready}} \cdot \_^\star\}$ [FV-If-Else]

3) `event[Done]; }`

$\{n=0 \wedge \underline{\textbf{Ready}} \cdot \_^\star \cdot \underline{\textbf{Done}}\}$ [FV-Event]

4) `else{`

$\{n \neq 0\}$ [FV-If-Else]

5) `event[Send];`

$\{n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^\star \cdot \underline{\textbf{Send}}\}$ [FV-Event]

6) `send(n-1); }}`

$rev(n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^\star \cdot \underline{\textbf{Send}}) \sqsubseteq rev(\Phi^{send(n-1)}_{pre})$ *(-check precondition-)*

$\{n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^\star \cdot \underline{\textbf{Send}} \cdot \Phi^{send(n-1)}_{post}\}$ [FV-Call]

7) $\Phi'_C = (n=0 \wedge \underline{\textbf{Ready}} \cdot \_^\star \cdot \underline{\textbf{Done}}) \vee (n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^\star \cdot \underline{\textbf{Send}} \cdot \Phi^{send(n-1)}_{post})$

8) $\Phi'_C \sqsubseteq \Phi^{send(n)}_{pre} \cdot \Phi^{send(n)}_{post} \Leftrightarrow$ *(– check postcondition –)*

$(n=0 \wedge \underline{\textbf{Done}}) \vee (n \neq 0 \wedge \underline{\textbf{Send}} \cdot \Phi^{send(n-1)}_{post}) \sqsubseteq \Phi^{send(n)}_{post}$

Overview (2)

The Forward Verifier –
To accumulate the effects

1) $\texttt{void send (int n)\{}$ *(– initialize the current effects state –)*
$\{\Phi_C = \Phi_{pre}^{send(n)} = \texttt{True} \wedge \underline{\textbf{Ready}} \cdot \_^{\star}\}$ [FV-Meth]

2) $\texttt{if(n==0)\{}$
$\{n=0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star}\}$ [FV-If-Else]

3) $\texttt{event[Done]; \}}$
$\{n=0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Done}}\}$ [FV-Event]

Add the events into the effect state

4) $\texttt{else\{}$
$\{n \neq 0\}$ [FV-If-Else]

5) $\texttt{event[Send];}$
$\{n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}}\}$ [FV-Event]

6) $\texttt{send(n-1); \}\}}$
$\texttt{rev}(n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}}) \sqsubseteq \texttt{rev}(\Phi_{pre}^{send(n-1)})$ *(-check precondition-)*
$\{n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}} \cdot \Phi_{post}^{send(n-1)}\}$ [FV-Call]

7) $\Phi'_C = (n=0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Done}}) \vee (n \neq 0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}} \cdot \Phi_{post}^{send(n-1)})$

8) $\Phi'_C \sqsubseteq \Phi_{pre}^{send(n)} \cdot \Phi_{post}^{send(n)} \Leftrightarrow$ *(– check postcondition –)*
$(n=0 \wedge \underline{\textbf{Done}}) \vee (n \neq 0 \wedge \underline{\textbf{Send}} \cdot \Phi_{post}^{send(n-1)}) \sqsubseteq \Phi_{post}^{send(n)}$

Overview (2)

The Forward Verifier – To accumulate the effects

1) `void send (int n){` *(– initialize the current effects state –)*
$\{\Phi_C = \Phi_{pre}^{send(n)} = \mathtt{True} \wedge \underline{\mathbf{Ready}} \cdot \_^{\star}\}$ [FV-Meth]

2) `if(n==0){`
$\{n=0 \wedge \underline{\mathbf{Ready}} \cdot \_^{\star}\}$ [FV-If-Else]

3) `event[Done]; }`
$\{n=0 \wedge \underline{\mathbf{Ready}} \cdot \_^{\star} \cdot \underline{\mathbf{Done}}\}$ [FV-Event]

Add the events into the effect state

4) `else{`
$\{n \neq 0\}$ [FV-If-Else]

5) `event[Send];`
$\{n \neq 0 \wedge \underline{\mathbf{Ready}} \cdot \_^{\star} \cdot \underline{\mathbf{Send}}\}$ [FV-Event]

Check if the current effect satisfies the callee's precondition

6) `send(n-1); }}`
$rev(n \neq 0 \wedge \underline{\mathbf{Ready}} \cdot \_^{\star} \cdot \underline{\mathbf{Send}}) \sqsubseteq rev(\Phi_{pre}^{send(n-1)})$ *(-check precondition-)*
$\{n \neq 0 \wedge \underline{\mathbf{Ready}} \cdot \_^{\star} \cdot \underline{\mathbf{Send}} \cdot \Phi_{post}^{send(n-1)}\}$ [FV-Call]

7) $\Phi'_C = (n=0 \wedge \underline{\mathbf{Ready}} \cdot \_^{\star} \cdot \underline{\mathbf{Done}}) \vee (n \neq 0 \wedge \underline{\mathbf{Ready}} \cdot \_^{\star} \cdot \underline{\mathbf{Send}} \cdot \Phi_{post}^{send(n-1)})$

8) $\Phi'_C \sqsubseteq \Phi_{pre}^{send(n)} \cdot \Phi_{post}^{send(n)} \Leftrightarrow$ *(– check postcondition –)*
$(n=0 \wedge \underline{\mathbf{Done}}) \vee (n \neq 0 \wedge \underline{\mathbf{Send}} \cdot \Phi_{post}^{send(n-1)}) \sqsubseteq \Phi_{post}^{send(n)}$

Overview (2)

The Forward Verifier –
To accumulate the effects

1) $\texttt{void send (int n)\{}$ *(– initialize the current effects state –)*
$\{\Phi_C = \Phi_{pre}^{send(n)} = \texttt{True} \wedge \underline{\textbf{Ready}} \cdot \_^{\star}\}$ [FV-Meth]

2) $\texttt{if(n==0)\{}$
$\{n{=}0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star}\}$ [FV-If-Else]

3) $\texttt{event[Done];\}}$
$\{n{=}0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Done}}\}$ [FV-Event]

Add the events into the effect state

4) $\texttt{else\{}$
$\{n{\neq}0\}$ [FV-If-Else]

5) $\texttt{event[Send];}$
$\{n{\neq}0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}}\}$ [FV-Event]

Check if the current effect satisfies the callee's precondition

6) $\texttt{send(n-1);\}\}}$
$\texttt{rev}(n{\neq}0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}}) \sqsubseteq \texttt{rev}(\Phi_{pre}^{send(n-1)})$ *(-check precondition-)*
$\{n{\neq}0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}} \cdot \Phi_{post}^{send(n-1)}\}$ [FV-Call]

7) $\Phi_C' = (n{=}0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Done}}) \vee (n{\neq}0 \wedge \underline{\textbf{Ready}} \cdot \_^{\star} \cdot \underline{\textbf{Send}} \cdot \Phi_{post}^{send(n-1)})$

8) $\Phi_C' \sqsubseteq \Phi_{pre}^{send(n)} \cdot \Phi_{post}^{send(n)} \Leftrightarrow$ *(– check postcondition –)*
$(n{=}0 \wedge \underline{\textbf{Done}}) \vee (n{\neq}0 \wedge \underline{\textbf{Send}} \cdot \Phi_{post}^{send(n-1)}) \sqsubseteq \Phi_{post}^{send(n)}$

Checks if the final effects satisfy the program's postcondition

Overview (3)

Term Rewriting System – the Effects inclusion checker

Cyclic Proof Succeed!

[Prove-Reoccur]

[Prove-Frame with Residue = emp] $s > 0 \land Send^{s-1} \sqsubseteq Send^{s-1}$ [Unfold with Send]

$s = 0 \ \land \ emp \ \sqsubseteq \ emp$ $\quad$ $s > 0 \land Send^{s} \sqsubseteq Send^{s}$ [Case Split]

$s = n\text{-}1 \land s >= 0 \land Send^{s} \sqsubseteq Send^{s}$ [Substitution]

$n > 0 \land Send^{n-1} \sqsubseteq Send^{n-1}$ [Unfold with Send]

$Send \cdot Send^{n-1} \sqsubseteq Send^{n}$

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.

Imp

• An op

• Bench ... olling progra

➢de ... invalid)

➢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

Imp

• An op

• Bench ... olling

progra

➢der ... invalid)

➢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

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.

Summary

- **Integrated Dependent Effects:** We define the syntax and semantics of the logic.
- **Automated Verification System:** Targeting C programs we develop:
  1) Front-end: a Hoare-style forward verifier; and
  2) Back-end: an effects inclusion checker (the TRS).
- **A prototype system of the novel effects logic:** Proven to be sound, with experimental results and case studies to show the feasibility.

Summary

Thanks a lot for your attention!

- **Integrated Dependent Effects:** We define the syntax and semantics of the logic.
- **Automated Verification System:** Targeting C programs we develop:
  1) Front-end: a Hoare-style forward verifier; and
  2) Back-end: an effects inclusion checker (the TRS).
- **A prototype system of the novel effects logic:** Proven to be sound, with experimental results and case studies to show the feasibility.