National University of Singapore

Automated Temporal Verification for Real-Time Systems via Implicit Clocks and an Extended Antimirov Algorithm

Yahui Song, Wei-Ngan Chin

National University of Singapore

25th April TACAS'23 @ Paris, France

Scan Me For the Project Repository

Automated Temporal Verification for Real-Time Systems via Implicit Clocks and an Extended Antimirov Algorithm

Yahui Song, Wei-Ngan Chin

National University of Singapore

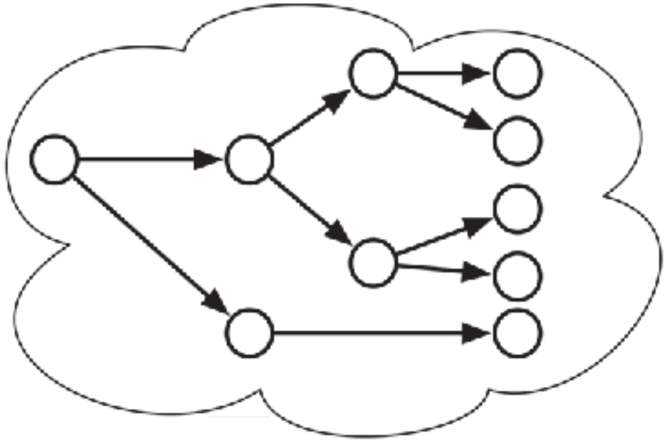
25th April TACAS’23 @ Paris, France

Scan Me For the Project Repository

Temporal Verification – Existing Framework

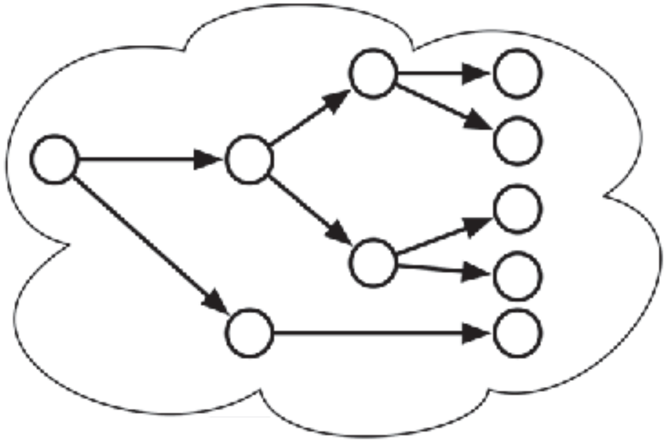
Temporal Verification – Existing Framework

Modelling →



Temporal Verification – Existing Framework

Modelling →



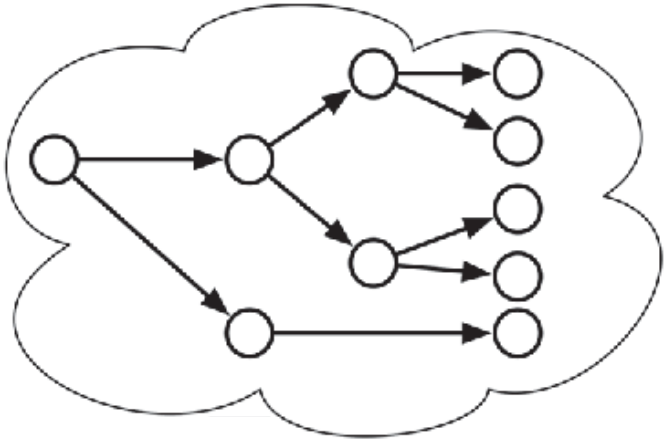
Expressiveness power is limited by the finite-state automata

Temporal Logic

Property $\boldsymbol{\Phi}$

Temporal Verification – Existing Framework

Modelling →



Temporal Logic

Property $\mathbf{\Phi}$

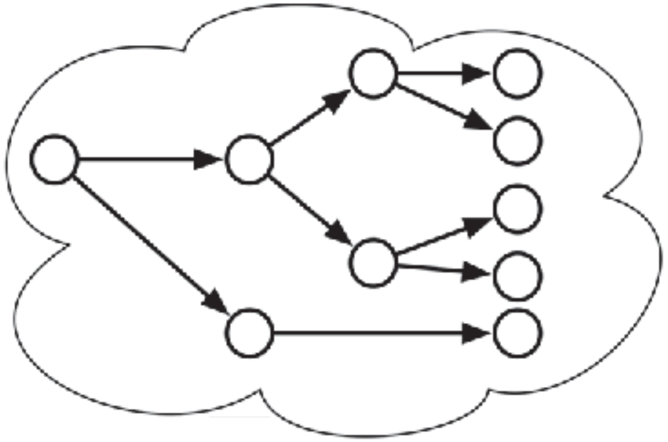
Expressiveness power is limited by the finite-state automata

MODEL CHECKER

To be bounded when non-terminating

Temporal Verification – Existing Framework

Modelling →



Expressiveness power is limited by the finite-state automata

Temporal Logic

Property Φ

MODEL CHECKER

To be bounded when non-terminating

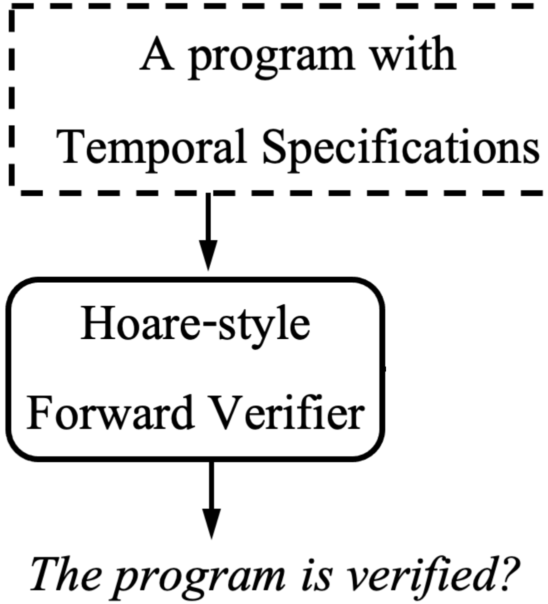
A verified logic
≠
A verified implementation

Yes, Property Φ is true

No, Property Φ is not true

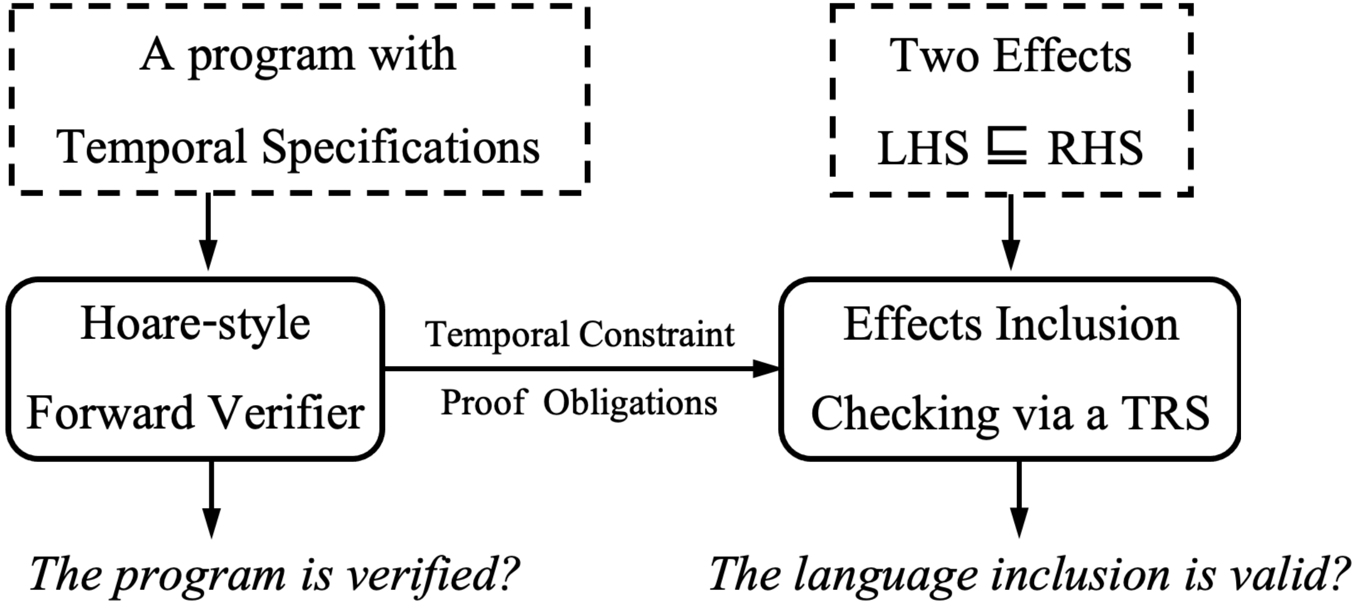
& counterexamples

A New Framework for Temporal Verification



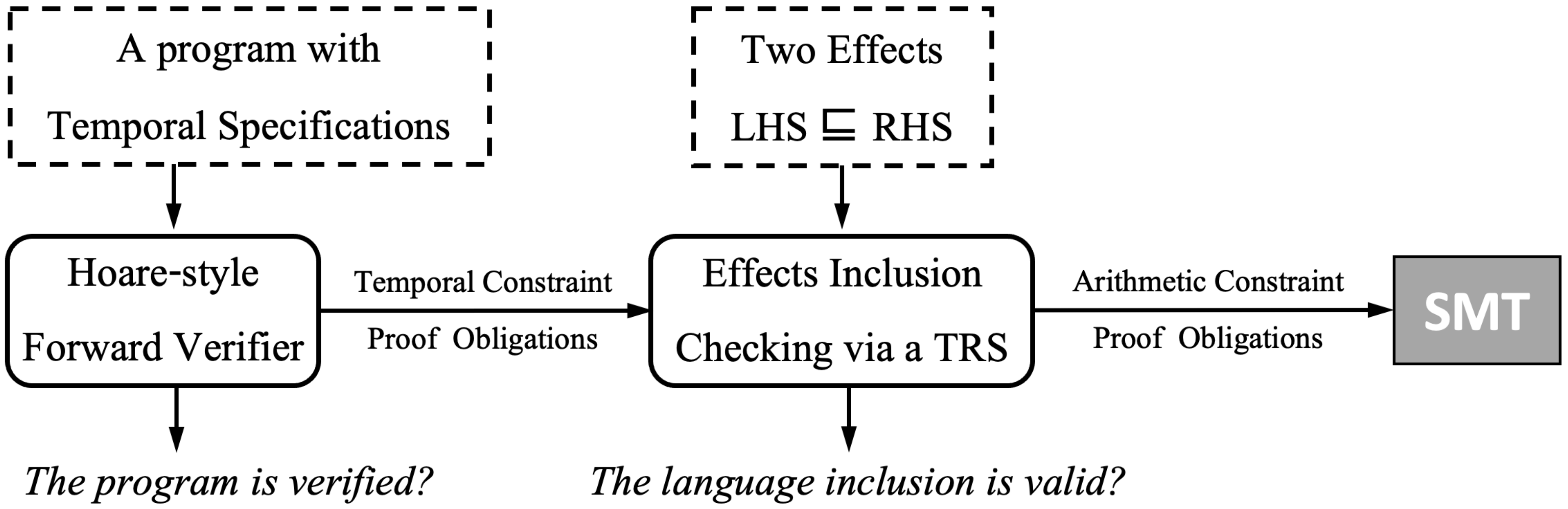
+ A verified logic = A verified implementation
+ Flexible specifications, which an be combined with other logic.

A New Framework for Temporal Verification



- \+ A verified logic = A verified implementation
- \+ Flexible specifications, which an be combined with other logic.
- \+ Symbolic entailment checker with co-inductive proofs for infinite traces.

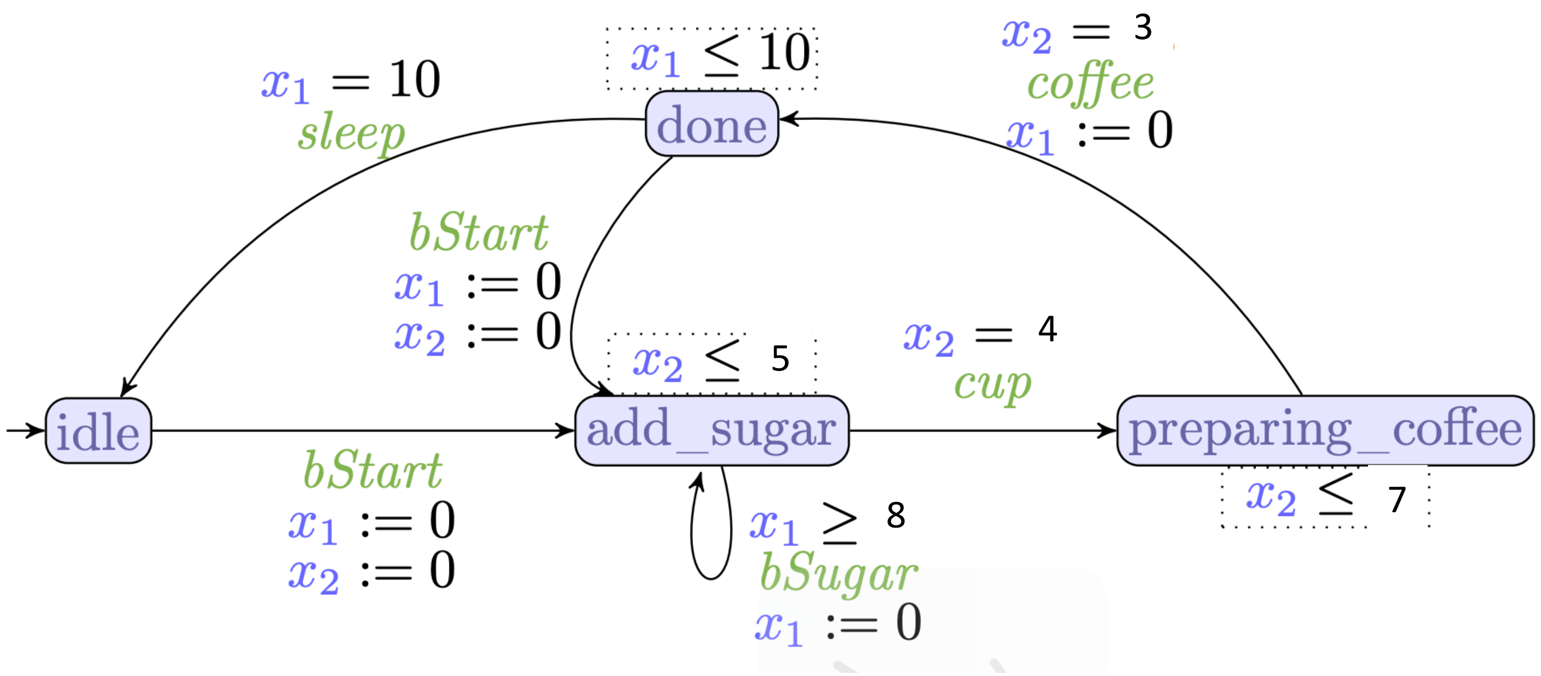
A New Framework for Temporal Verification



+ A verified logic = A verified implementation
+ Flexible specifications, which an be combined with other logic.
+ Symbolic entailment checker with co-inductive proofs for infinite traces.
- Automation/Decidability.

Timed Verification via Timed Automata

- Timed Automata lack high-level compositional patterns for hierarchical design.



Timed Verification via Timed Automata

- Timed Automata lack high-level compositional patterns for hierarchical design.
- Manually casting clocks is tedious and error-prone.

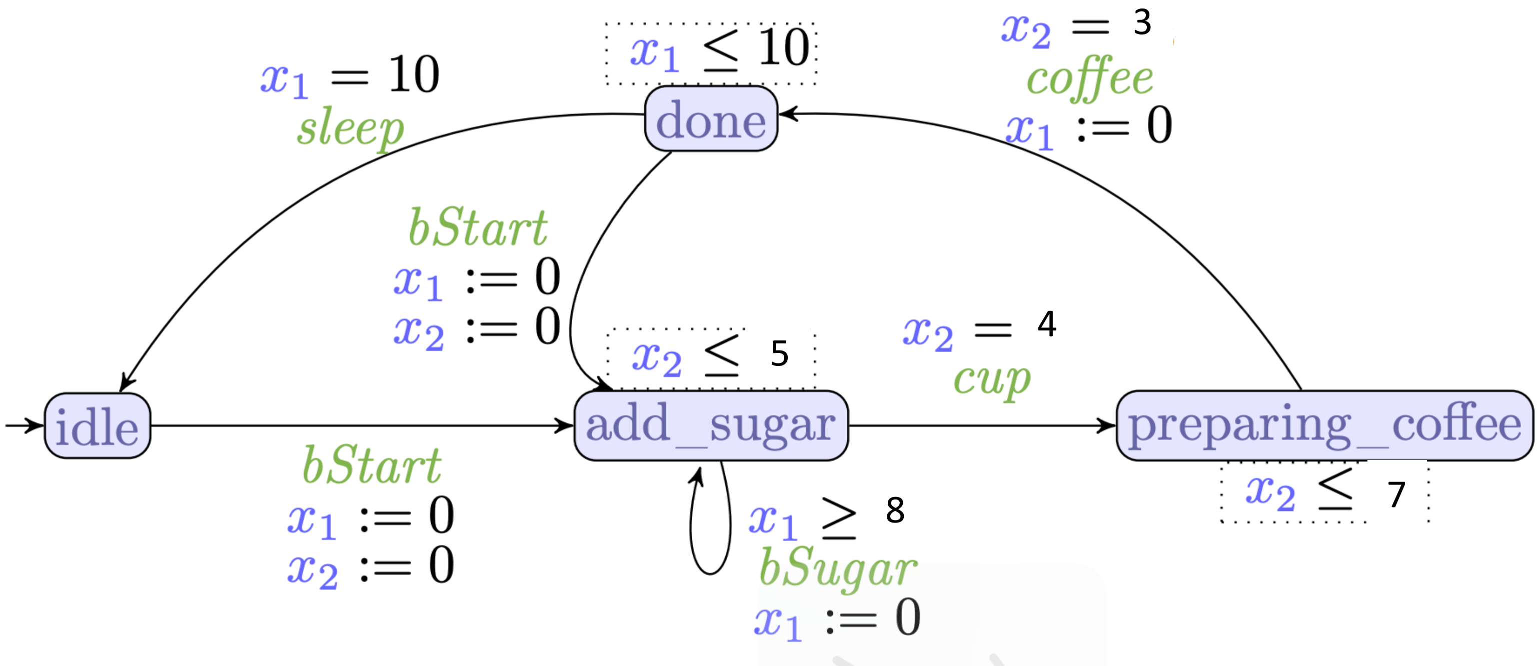


Diagram modified from "Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata" in FTSCS '22

Timed Verification via Timed Automata

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

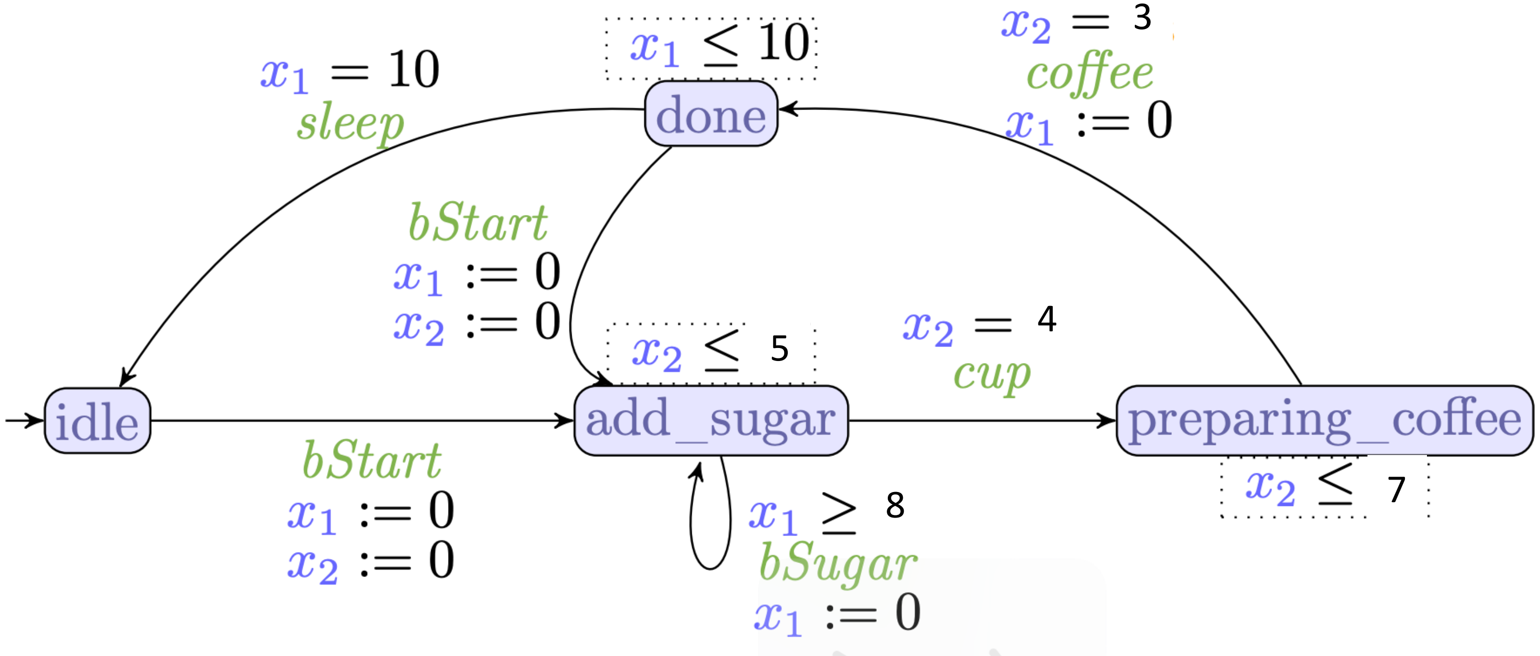


Diagram modified from "Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata" in FTSCS '22

Timed Verification via Timed Automata

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

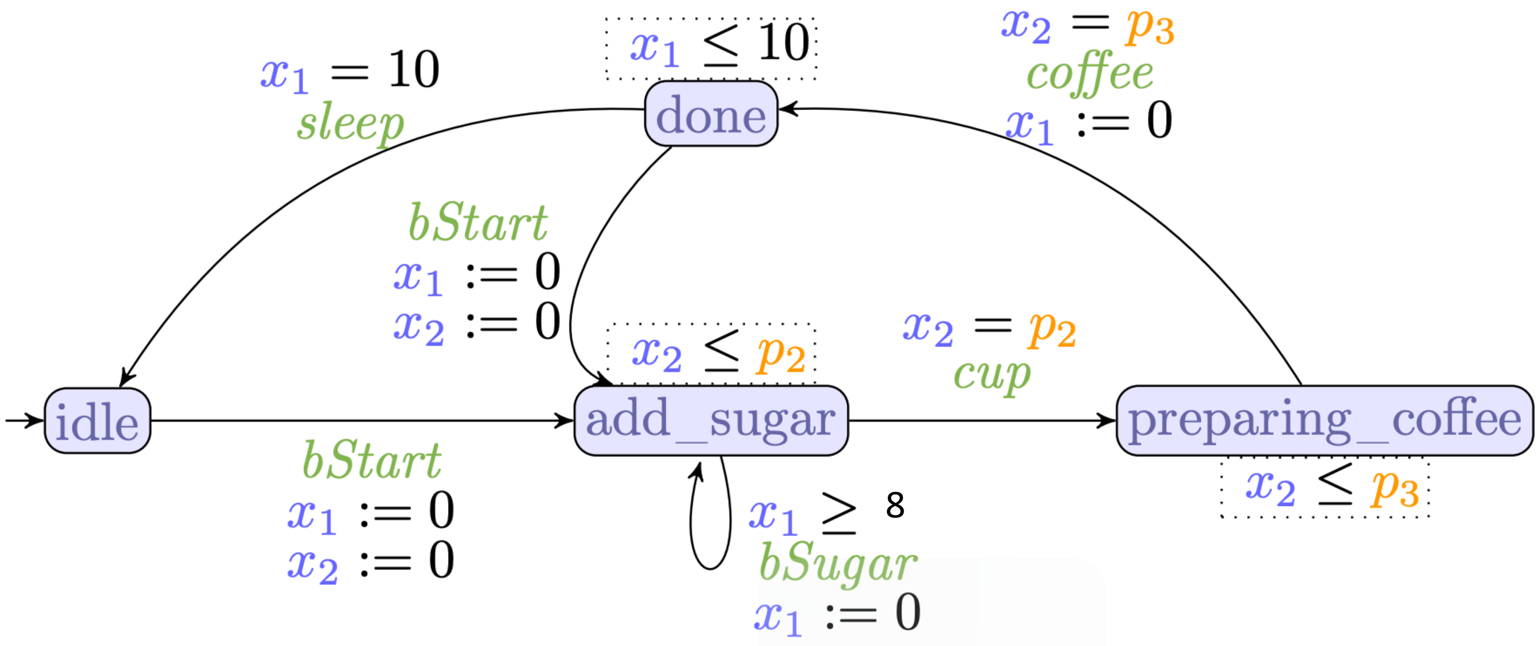


Diagram taken from "Rewriting Logic Semantics and Symbolic Analysis for Parametric Timed Automata" in FTSCS '22

We propose TimEffs - Symbolic Timed Automata

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

We propose TimEffs - Symbolic Timed Automata

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

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

We propose TimEffs - Symbolic Timed Automata

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

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

```
void makeCoffee (int n)
/* req: n≥0 ∧ _*· CupReady
   ens: n≤t≤5 ∧ t'≤4 ∧
         (EndSugar # t) · (Coffee # t') */
{ deadline (addNSugar(n), 5);
   deadline (event["Coffee"],4);}
```

We propose TimEffs - Symbolic Timed Automata

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

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

```
void makeCoffee (int n)
/* req: n≥0 ∧ _*· CupReady
   ens: n≤t≤5 ∧ t'≤4 ∧
         (EndSugar # t) · (Coffee # t') */
{ deadline (addNSugar(n), 5);
  deadline (event["Coffee"],4);}

int main ()
/* req: true ∧ ϵ
   ens: t≤9 ∧ ((!Done)* # t) · Done */
{ event["CupReady"];
  makeCoffee (3);
  event["Done"];}
```

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.

    if (n == 0){

        event ["EndSugar"];}

    else {

        addOneSugar();

        addNSugar (n-1);}}
```

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
```

$\Phi_C = \Phi_{pre}^{addNSugar(n)} = \{\mathtt{true} \wedge \_^{\star}\}$ $\quad [FV\text{-}Meth]$

```
    if (n == 0){

        event ["EndSugar"];}

    else {

        addOneSugar();

        addNSugar (n-1);}}
```

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
```

$\Phi_C = \Phi_{pre}^{addNSugar(n)} = \{\mathtt{true} \wedge \_^\star\}$ $\quad$ $[FV\text{-}Meth]$

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

$\{\mathtt{n}{=}0 \wedge \_^\star\}$ $\quad$ $[FV\text{-}Cond]$

```
            event ["EndSugar"];}

      else {
```

$\{\mathtt{n}{\neq}0 \wedge \_^\star\}$ $\quad$ $[FV\text{-}Cond]$

```
            addOneSugar();

            addNSugar (n-1);}}
```

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
```
$\Phi_C = \Phi_{pre}^{addNSugar(n)} = \{\mathtt{true} \wedge \_^{\star}\}$ $[FV\text{-}Meth]$
```
    if (n == 0){
```
$\{\mathtt{n}{=}0 \wedge \_^{\star}\}$ $[FV\text{-}Cond]$
```
        event ["EndSugar"];}
```
$\{\mathtt{n}{=}0 \wedge \_^{\star} \cdot \mathtt{EndSugar}\}$ $[FV\text{-}Event]$
```
    else {
```
$\{\mathtt{n}{\neq}0 \wedge \_^{\star}\}$ $[FV\text{-}Cond]$
```
        addOneSugar();

        addNSugar (n-1);}}
```

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
```
$\Phi_C = \Phi_{pre}^{addNSugar(n)} = \{\mathtt{true} \wedge \_^{\star}\}$ $[FV\text{-}Meth]$
```
      if (n == 0){
```
$\{\mathtt{n{=}0} \wedge \_^{\star}\}$ $[FV\text{-}Cond]$
```
            event ["EndSugar"];}
```
$\{\mathtt{n{=}0} \wedge \_^{\star} \cdot \mathtt{EndSugar}\}$ $[FV\text{-}Event]$
```
      else {
```
$\{\mathtt{n{\neq}0} \wedge \_^{\star}\}$ $[FV\text{-}Cond]$
```
            addOneSugar();
```
$\{\mathtt{n{\neq}0 \wedge t2{>}1} \wedge \_^{\star} \cdot (\epsilon \ \#\ \mathtt{t2})\}$ $[FV\text{-}Call]$
```
            addNSugar (n-1);}}
```

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
```
$\Phi_C = \Phi_{pre}^{addNSugar(n)} = \{\texttt{true} \wedge \_^{\star}\}$ $[FV\text{-}Meth]$

```
      if (n == 0){
```
$\{\texttt{n=0} \wedge \_^{\star}\}$ $[FV\text{-}Cond]$

```
            event ["EndSugar"];}
```
$\{\texttt{n=0} \wedge \_^{\star} \cdot \texttt{EndSugar}\}$ $[FV\text{-}Event]$

```
      else {
```
$\{\texttt{n}\neq\texttt{0} \wedge \_^{\star}\}$ $[FV\text{-}Cond]$

```
            addOneSugar();
```
$\{\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1} \wedge \_^{\star} \cdot (\epsilon \ \# \ \texttt{t2})\}$ $[FV\text{-}Call]$

```
            addNSugar (n-1);}}
```
$\texttt{n}\neq\texttt{0}\wedge\texttt{t2>1} \wedge \_^{\star} \cdot (\epsilon \ \# \ \texttt{t2}) \sqsubseteq \Phi_{pre}^{addNSugar(n-1)}$ // *TRS: precondition checked.*

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
```
$\Phi_C = \Phi_{pre}^{addNSugar(n)} = \{\mathtt{true} \wedge \_^\star\}$ $\quad [FV\text{-}Meth]$

```
      if (n == 0){
```
$\{\mathtt{n{=}0} \wedge \_^\star\}$ $\quad [FV\text{-}Cond]$

```
            event ["EndSugar"];}
```
$\{\mathtt{n{=}0} \wedge \_^\star \cdot \mathtt{EndSugar}\}$ $\quad [FV\text{-}Event]$

```
      else {
```
$\{\mathtt{n{\neq}0} \wedge \_^\star\}$ $\quad [FV\text{-}Cond]$

```
            addOneSugar();
```
$\{\mathtt{n{\neq}0 \wedge t2{>}1} \wedge \_^\star \cdot (\epsilon \ \# \ \mathtt{t2})\}$ $\quad [FV\text{-}Call]$

```
            addNSugar (n-1);}}
```
$\mathtt{n{\neq}0 \wedge t2{>}1} \wedge \_^\star \cdot (\epsilon \ \# \ \mathtt{t2}) \sqsubseteq \Phi_{pre}^{addNSugar(n-1)}$ $\quad$ // *TRS: precondition checked.*

$\{\mathtt{n{\neq}0 \wedge t2{>}1} \wedge \_^\star \cdot (\epsilon \ \# \ \mathtt{t2}) \cdot \Phi_{post}^{addNSugar(n-1)}\}$ $\quad [FV\text{-}Call]$

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
```
$\Phi_C{=}\Phi_{pre}^{addNSugar(n)} = \{\mathtt{true} \wedge \_^{\star}\}$ $[FV\text{-}Meth]$
```
      if (n == 0){
```
$\{\mathtt{n}{=}0 \wedge \_^{\star}\}$ $[FV\text{-}Cond]$
```
            event ["EndSugar"];}
```
$\{\mathtt{n}{=}0 \wedge \_^{\star}\cdot \mathtt{EndSugar}\}$ $[FV\text{-}Event]$
```
      else {
```
$\{\mathtt{n}{\neq}0 \wedge \_^{\star}\}$ $[FV\text{-}Cond]$
```
            addOneSugar();
```
$\{\mathtt{n}{\neq}0{\wedge}\mathtt{t2}{>}1 \wedge \_^{\star} \cdot (\epsilon \ \# \ \mathtt{t2})\}$ $[FV\text{-}Call]$
```
            addNSugar (n-1);}}
```
$\mathtt{n}{\neq}0{\wedge}\mathtt{t2}{>}1 \wedge \_^{\star} \cdot (\epsilon \ \# \ \mathtt{t2}) \sqsubseteq \Phi_{pre}^{addNSugar(n\text{-}1)}$ // TRS: precondition checked.

$\{\mathtt{n}{\neq}0{\wedge}\mathtt{t2}{>}1 \wedge \_^{\star} \cdot (\epsilon \ \# \ \mathtt{t2}) \cdot \Phi_{post}^{addNSugar(n\text{-}1)}\}$ $[FV\text{-}Call]$

$\Phi'_C{=} (\mathtt{n}{=}0 \wedge \_^{\star}{\cdot}\mathtt{EndSugar}) \vee (\mathtt{n}{\neq}0{\wedge}\mathtt{t2}{>}1 \wedge \_^{\star}{\cdot}(\epsilon\#\mathtt{t2}){\cdot}\Phi_{post}^{addNSugar(n\text{-}1)})$

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.
    if (n == 0){
        event ["EndSugar"];}
    else {
        addOneSugar();
        addNSugar (n-1);}}
```

$\Phi'_C = (n{=}0 \wedge \_^{\star}\cdot \mathtt{EndSugar}) \vee (n{\neq}0 \wedge t2{>}1 \wedge \_^{\star}\cdot(\epsilon\#t2)\cdot\Phi_{post}^{addNSugar(n\text{-}1)})$

$\Phi'_C \sqsubseteq \Phi_{pre}^{addNSugar(n)}$

Hoare-style Forward Verifier

```
void addNSugar (int n){    // initialize the state using the function precondition.

    if (n == 0){

        event ["EndSugar"];}

    else {

        addOneSugar();

        addNSugar (n-1);}}
```

$$\Phi'_C = (n{=}0 \wedge \_^{\star}\cdot \mathtt{EndSugar}) \vee (n{\neq}0 \wedge t2{>}1 \wedge \_^{\star}\cdot(\epsilon \# t2)\cdot \Phi_{post}^{addNSugar(n-1)})$$

$$\Phi'_C \sqsubseteq \Phi_{pre}^{addNSugar(n)}$$

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

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);}}
```

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

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);}}
```

$(n{=}0 \wedge ES)\ \vee$

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

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);}}
```

$$n=0 \wedge \epsilon \sqsubseteq tR \geq 0 \wedge \epsilon \; \# \; tR$$ ④ [PROVE]

$$n=0 \wedge \cancel{ES} \sqsubseteq tR \geq 0 \wedge \cancel{ES} \# tR$$ ③ [UNFOLD]

$$(n=0 \wedge ES) \vee$$

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

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);}}
```

$$(I)$$

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \; ② [LHS\text{-}OR]$$

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

$$\overline{\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad} \; ① [RENAME]$$

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

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);}}
```

$$(I)$$

② [LHS-OR]

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

① [RENAME]

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

$$(I)$$

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

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: $\mathtt{tL}=(\mathtt{tR}-\mathtt{t2})$

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

⑤ [UNFOLD]

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

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);}}
```

$$\frac{\dfrac{(I)}{\vee\ (\mathtt{n}\neq 0 \wedge \mathtt{t2}>1 \wedge \mathtt{tL}\geq(\mathtt{n}-1) \wedge\ \epsilon\#\mathtt{t2} \cdot \mathtt{ES\#tL}) \sqsubseteq \mathtt{tR}\geq\mathtt{n} \wedge \mathtt{ES\#tR}}\ ②[\mathit{LHS\text{-}OR}]}{(\mathtt{n}=0 \wedge \mathtt{ES}) \vee (\mathtt{n}\neq 0 \wedge \mathtt{t2}>1 \wedge (\epsilon\ \#\ \mathtt{t2}) \cdot \Phi^{addNSugar(n\text{-}1)}_{post}) \sqsubseteq \Phi^{addNSugar(n)}_{post}}\ ①[\mathit{RENAME}]$$

(I)

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

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);}}
```

$$(I)$$

$$\vee\ (n\neq 0 \wedge t2>1 \wedge tL\geq(n\text{-}1) \wedge\ \epsilon\#t2 \cdot ES\#tL) \sqsubseteq tR\geq n \wedge ES\#tR \quad ②[LHS\text{-}OR]$$

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

$$(I)$$

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

$$⑦[PROVE]$$

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

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

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

$$⑤[UNFOLD]$$

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

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!

$$\texttt{n=0} \wedge \epsilon \sqsubseteq \texttt{tR}\geq\texttt{0} \wedge \epsilon \;\#\; \texttt{tR}$$

④ [PROVE]

$$\texttt{n=0} \wedge \texttt{ES} \sqsubseteq \texttt{tR}\geq\texttt{0} \wedge \texttt{ES\#tR} \qquad (I)$$

③ [UNFOLD]

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

② [LHS-OR]

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

① [RENAME]

(I)

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

⑦ [PROVE]

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

⑥ [UNFOLD] π_u:tL=(tR-t2)

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

⑤ [UNFOLD]

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

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:* $u^{-1}S = \{w \in \Sigma^* \mid uw \in S\}$.

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,***

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

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,***

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

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

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

Target Language C^t, imperative with timed constructs:

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

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

Specification Language TimEffs:

$$(Timed\ Effects) \quad \Phi ::= \pi \wedge \theta \mid \Phi_1 \vee \Phi_2$$

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

$$(Pure) \quad \pi ::= True \mid False \mid bop(t_1, t_2) \mid \pi_1 \wedge \pi_2 \mid \pi_1 \vee \pi_2 \mid \neg \pi \mid \pi_1 \Rightarrow \pi_2$$

$$(Real\text{-}Time\ Terms) \quad t ::= c \mid x \mid t_1 + t_2 \mid t_1 - t_2$$

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

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.

Scan Me For the Project Repository

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); }
```

Scan Me For the Project Repository

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
  ensₐ:true ∧ (cs≤1)*   ens_b:true ∧ ((_*).Critical.Exit.(_*))* */
  { proc(0) || proc(1) || proc(2); }
```

Scan Me For the Project Repository

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	≤0.05	≤0.09
3	0.21	35	0.093	42	≤0.05	≤0.09
4	0.46	63	0.120	47	0.05	0.09
5	25.0	84	0.128	52	0.15	0.19

Scan Me For the Project Repository

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;

Scan Me For the Project Repository

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.

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.

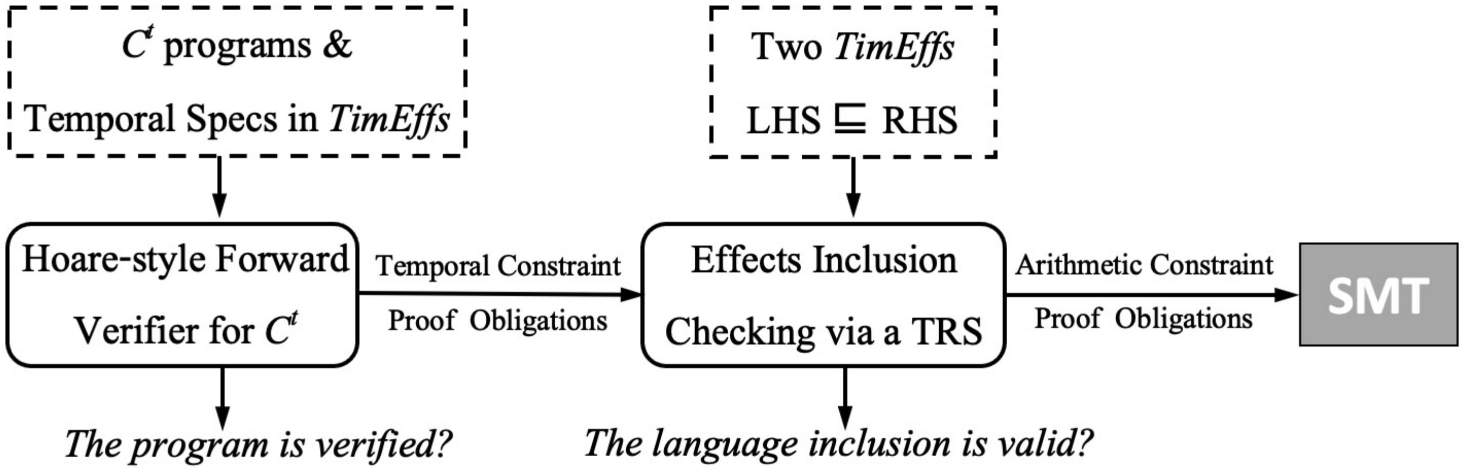
iii. our verification time largely depends on the number of querying Z3.

Scan Me For the Project Repository

Conclusion

➢New approach for verifying Real-Time Systems

- Syntax and semantics of the TimEffs

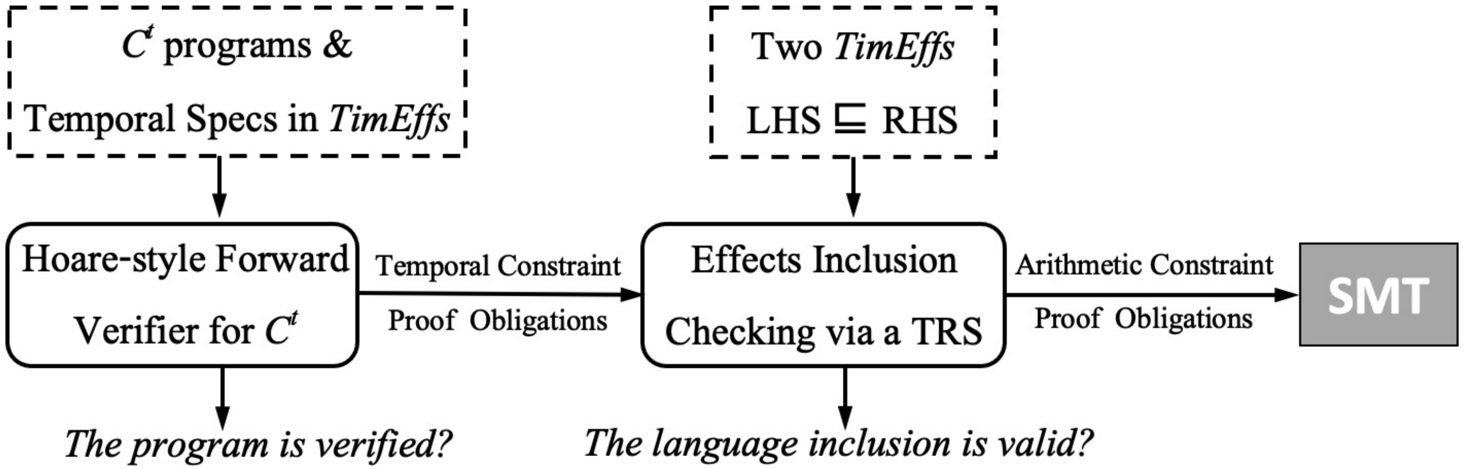


Scan Me For the Project Repository

Conclusion

➢ New approach for verifying Real-Time Systems

- Syntax and semantics of the TimEffs
- Automated Verification System: Hoare-style forward verifier + TRS

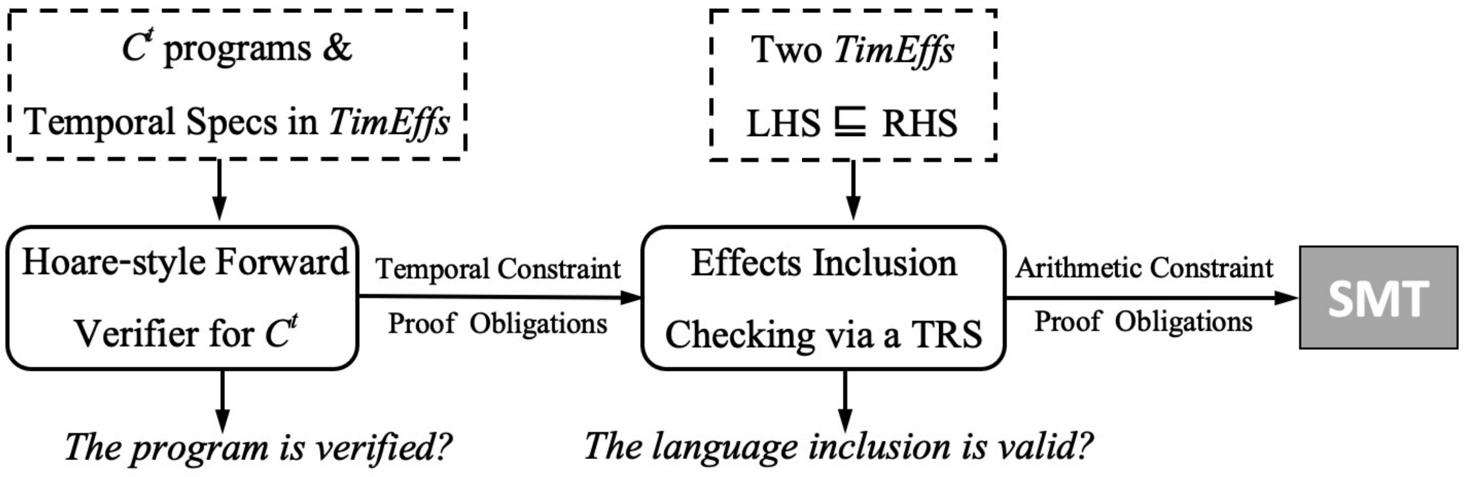


Scan Me For the
Project Repository

Conclusion

➢ New approach for verifying Real-Time Systems

- Syntax and semantics of the TimEffs
- Automated Verification System: Hoare-style forward verifier + TRS
- Prototype system (3000 LOC OCaml): experimental results and case studies



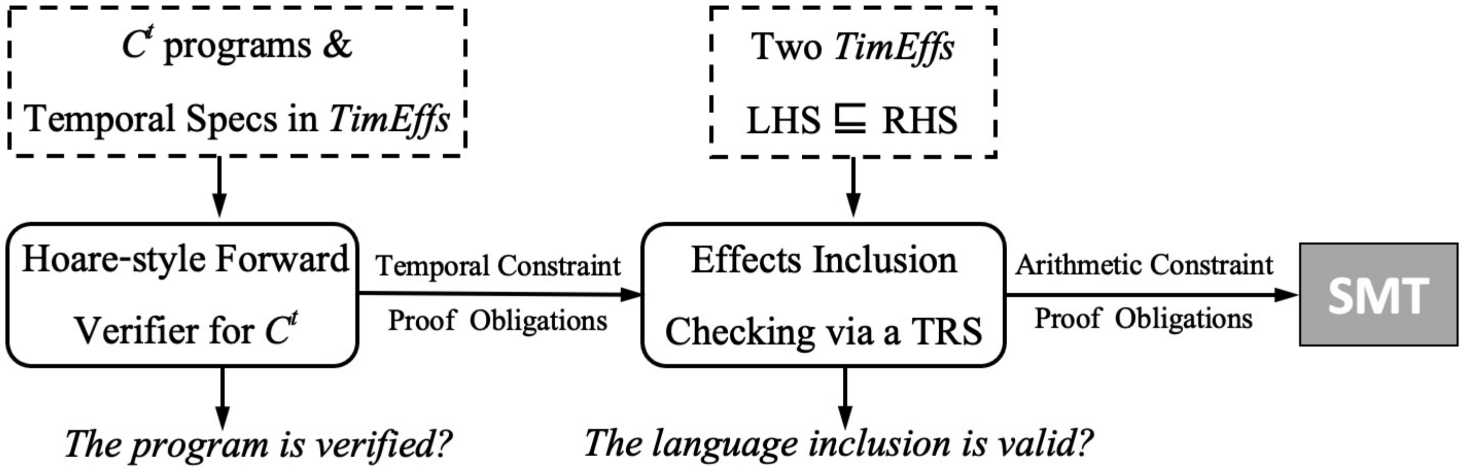
Scan Me For the
Project Repository

Conclusion

➢ New approach for verifying Real-Time Systems

- Syntax and semantics of the TimEffs
- Automated Verification System: Hoare-style forward verifier + TRS
- Prototype system (3000 LOC OCaml): experimental results and case studies

Thanks!



Scan Me For the
Project Repository