Unbounded-Thread Program Verification using Thread-State Equations

Konstantinos Athanasiou Peizun Liu Thomas Wahl

Northeastern University, Boston, MA

IJCAR 2016, University of Coimbra, Portugal

```c
unsigned value, m = 0;

unsigned count() {
  unsigned v = 0;

  acquire(m);
  if(value == 0u-1) {
    release(m);

    return 0;
  }
  else {
    v = value;
    value = v + 1;
    release(m);
    assert(value > v);
    return v + 1;
  }
}

int main() {
  while(...)
    thread(&count);
}
```

inc.c

```c
unsigned value, m = 0;

unsigned count() {
  unsigned v = 0;

  acquire(m);
  if(value == 0u-1) {
    release(m);

    return 0;
  }
  else {
    v = value;
    value = v + 1;
    release(m);
    assert(value > v);
    return v + 1;
  }
}

int main() {
  while(...)
    thread(&count);
}
```

inc.c

```
void main() begin
  l1:
    if 0 then goto l6; fi;
    start_thread goto l2;
    goto l5;
  l2:
    atomic_begin;
    assume(*);
    atomic_end;
    if !(*) then goto l3; fi;
    atomic_begin;
    assume(*);
    atomic_end;
    goto l4;
  l3:
    atomic_begin;
    assume(*);
    atomic_end;
    assert(!(*));
    goto l4;
  l4:
    end_thread;
  l5:
    goto l1;
  l6:
end
```

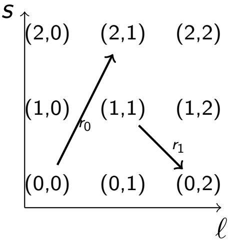
inc.bp

```
unsigned value, m = 0;

unsigned count() {
  unsigned v = 0;

  acquire(m);
  if(value == 0u−1) {
    release(m);

    return 0;
  }
  else {
    v = value;
    value = v + 1;
    release(m);
    assert(value > v);
    return v + 1;
  }
}

int main() {
  while(...)
    thread(&count);
}
```

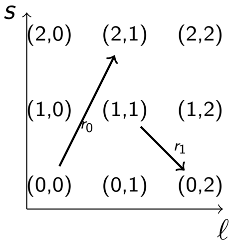
inc.c

```
void main() begin
  l1:
    if 0 then goto l6; fi;
    start_thread goto l2;
    goto l5;
  l2:
    atomic_begin;
    assume(∗);
    atomic_end;
    if !(∗) then goto l3; fi;
    atomic_begin;
    assume(∗);
    atomic_end;
    goto l4;
  l3:
    atomic_begin;
    assume(∗);
    atomic_end;
    assert(!(∗));
    goto l4;
  l4:
    end_thread;
  l5:
    goto l1;
  l6:
end
```

inc.bp

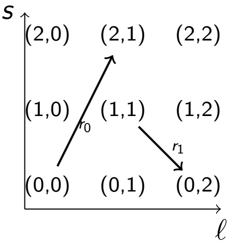


Thread-Transition Systems (TTS)



- Finite-state models extracted from recursion-free, finite-data procedures executed by threads
- (s, l): *shared s* and *local l* component.
- Configurations of the form $(s|\ell_1, \ldots, \ell_n)$
- $(0|0,0) \xrightarrow{r_0} (2|1,0)$

Thread-Transition Systems (TTS)



- Finite-state models extracted from recursion-free, finite-data procedures executed by threads
- (s, l): *shared s* and *local l* component.
- Configurations of the form $(s|\ell_1, \ldots, \ell_n)$
- $(0|0, 0) \xrightarrow{r_0} (2|1, 0)$

Problem Statement

Given a target conf. $v_F = (s_F|\ell_F)$, can the unbounded-thread system reach a configuration of the form $v = (s_F|\ell_1, \ldots, \ell_F, \ldots)$?

The coverability problem for TTS

- Coverability is decidable for well-quasi ordered systems.
  (Finkel and Schnoebelen '01, Abdulla '10)

The coverability problem for TTS

- Coverability is decidable for well-quasi ordered systems.
  (Finkel and Schnoebelen '01, Abdulla '10)

- Complexity Issues: $\mathrm{EXPSPACE}$-complete.
  (Cardoza et al. '76, Rackoff '78)

What if we over-approximate?

What if we over-approximate?

- Esparza et al. encoded the Petri net *marking equation* into symbolic expressions to over-approximate coverability (CAV'14).
- Out-performed existing approaches with high rate of success on uncoverable instances (inapplicable on coverable).

What if we over-approximate?

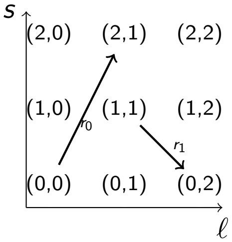
- Esparza et al. encoded the Petri net *marking equation* into symbolic expressions to over-approximate coverability (CAV'14).
- Out-performed existing approaches with high rate of success on uncoverable instances (inapplicable on coverable).

This work

- Set of equations for TTS expressed in the decidable theory of ILA, whose inconsistency implies *uncoverability* of v_F.
- Algorithm that uses the equations to often prove uncoverable *and* coverable instances and detect spurious assignments.

Thread State Equations

Thread and Transition Counting

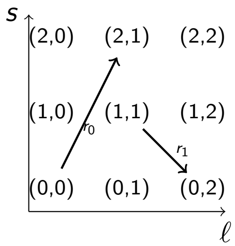


- $G = (T, R)$ where $T = S \times L$
- $\mathbf{r} \in \mathbb{N}^{|R|}$
- $\boldsymbol{\ell}_{\mathrm{I}} \in \mathbb{N}^{|L|}$
- $\boldsymbol{\ell}_F \in \mathbb{N}^{|L|}$
- $\mathbf{c} \in \{0, 1, -1\}^{|L| \times |R|}$

$$(0|0,0) \xmapsto{r_0} (2|1,0)$$

$$\mathbf{c}(\ell, r) = \begin{cases} +1 & \text{if transition } r \text{ \textit{ends} in local state } \ell \\ -1 & \text{if transition } r \text{ \textit{starts} in local state } \ell \\ 0 & \text{otherwise.} \end{cases}$$

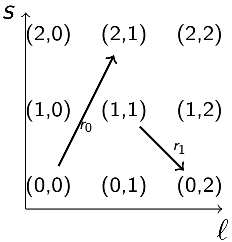
Thread and Transition Counting



- $G = (T, R)$ where $T = S \times L$
- $\mathbf{r} \in \mathbb{N}^{|R|}$
- $\boldsymbol{\ell}_{\mathrm{I}} \in \mathbb{N}^{|L|}$
- $\boldsymbol{\ell}_F \in \mathbb{N}^{|L|}$
- $\mathbf{c} \in \{0, 1, -1\}^{|L| \times |R|}$

$$\mathcal{C}_L = \bigwedge \begin{cases} \mathbf{r} \geq 0 \\ \boldsymbol{\ell}_{\mathrm{I}} \geq 0 \\ \boldsymbol{\ell}_F \geq 0 \\ \bigwedge_{\ell \notin L_{\mathrm{I}}} \boldsymbol{\ell}_{\mathrm{I}}(\ell) = 0 \\ \boldsymbol{\ell}_F = \boldsymbol{\ell}_{\mathrm{I}} + \mathbf{c} \cdot \mathbf{r} \\ \bigwedge_{\ell \in L} \boldsymbol{\ell}_F(\ell) \geq |\{i : v_F(i) = \ell\}| \end{cases}$$

An example

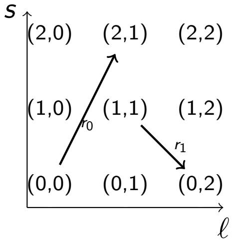


$$\mathcal{C}_L = \bigwedge \begin{cases} \dots \\ \boldsymbol{\ell}_F = \boldsymbol{\ell}_{\mathrm{I}} + \mathbf{c} \cdot \mathbf{r} \\ \bigwedge_{\ell \in L} \boldsymbol{\ell}_F(\ell) \geq |\{i : v_F(i) = \ell\}| \end{cases}$$

$$v_F = (s_F | \ell_F) = (0|2)$$

$$\boldsymbol{\ell}_F(0) = \boldsymbol{\ell}_{\mathrm{I}}(0) - \mathbf{r}(0)$$
$$\boldsymbol{\ell}_F(1) = \mathbf{r}(0) - \mathbf{r}(1)$$
$$\boldsymbol{\ell}_F(2) = \mathbf{r}(1)$$
$$\boldsymbol{\ell}_F(2) \geq 1$$

An example



$$\mathcal{C}_L \;=\; \bigwedge \left\{ \begin{array}{l} \dots \\ \boldsymbol{\ell}_F = \boldsymbol{\ell}_{\mathrm{I}} + \mathbf{c} \cdot \mathbf{r} \\ \bigwedge_{\ell \in L} \boldsymbol{\ell}_F(\ell) \geq |\{i : v_F(i) = \ell\}| \end{array} \right.$$

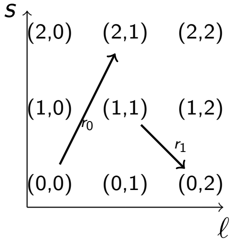
$$v_F = (s_F | \ell_F) = (0|2)$$

$$\mathbf{r} \;=\; \begin{bmatrix} 1 \\ 1 \end{bmatrix},\; \boldsymbol{\ell}_{\mathrm{I}} \;=\; \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix},\; \boldsymbol{\ell}_F \;=\; \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

sat. assignment

$$\begin{aligned} \boldsymbol{\ell}_F(0) &= \boldsymbol{\ell}_{\mathrm{I}}(0) - \mathbf{r}(0) \\ \boldsymbol{\ell}_F(1) &= \mathbf{r}(0) - \mathbf{r}(1) \\ \boldsymbol{\ell}_F(2) &= \mathbf{r}(1) \\ \boldsymbol{\ell}_F(2) &\geq 1 \end{aligned}$$

An example



Vast over-approximation

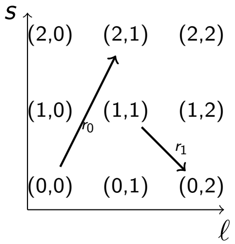
- Ordering is violated
- Shared state not utilized

$$v_F = (s_F | \ell_F) = (0|2)$$

$$\mathbf{r} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \ \boldsymbol{\ell}_{\mathrm{I}} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \ \boldsymbol{\ell}_F = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

sat. assignment

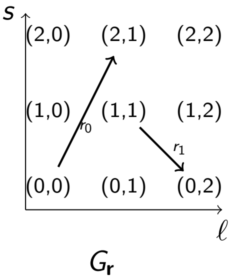
$$\begin{aligned} \boldsymbol{\ell}_F(0) &= \boldsymbol{\ell}_{\mathrm{I}}(0) - \mathbf{r}(0) \\ \boldsymbol{\ell}_F(1) &= \mathbf{r}(0) - \mathbf{r}(1) \\ \boldsymbol{\ell}_F(2) &= \mathbf{r}(1) \\ \boldsymbol{\ell}_F(2) &\geq 1 \end{aligned}$$

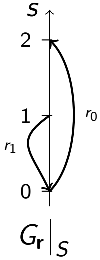


$G_{\mathbf{r}}$

A sequence of transitions forms a path p,

1. utilizing the transitions in the *multiplicity* given by $\mathbf{r}$,
2. *synchronizing* on the shared states.





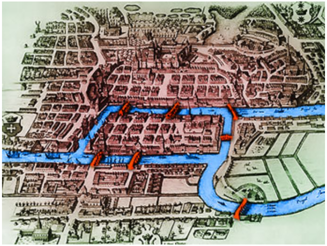
A sequence of transitions forms a path p,

1. utilizing the transitions in the *multiplicity* given by $\mathbf{r}$,
2. *synchronizing* on the shared states.

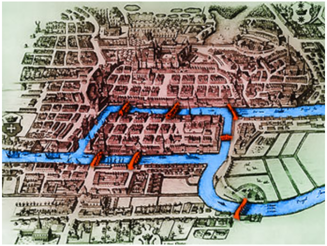
Observation

1 and 2 are satisfied *iff* p forms an **Euler path** in $G_{\mathbf{r}}\big|_S$

The Seven Bridges of Königsberg



The Seven Bridges of Königsberg



There exists an Euler path from s_{I} to s_F in $G_{\mathbf{r}}\big|_S$ iff:

flow: each shared state except s_{I} and s_F is entered and exited the same number of times, and

connectivity: the undirected subgraph of $G_{\mathbf{r}}\big|_S$ is connected.

Flow Condition

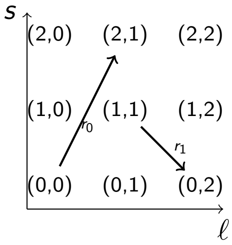
$$flow(s) \quad :: \quad \sum_{r \in in(s)} \mathbf{r}(r) \; - \; \sum_{r \in out(s)} \mathbf{r}(r) \; = \; N$$

where

$$N \; = \; \begin{cases} 0 & \text{if } s \notin \{s_I, s_F\} \textbf{ or } s = s_I = s_F \\ -1 & \text{if } s = s_{\mathrm{I}} \neq s_F \\ +1 & \text{if } s = s_F \neq s_{\mathrm{I}} \end{cases}$$

$$\mathcal{C}_F = \bigwedge_{s \in S} flow(s)$$

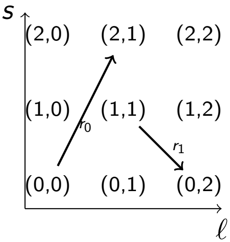
A full example

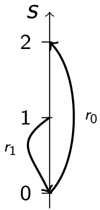


$$s_{\mathrm{I}} = 0, s_F = 0$$

$$\boldsymbol{\ell}_F(0) = \boldsymbol{\ell}_{\mathrm{I}}(0) - \mathbf{r}(0)$$
$$\boldsymbol{\ell}_F(1) = \mathbf{r}(0) - \mathbf{r}(1)$$
$$\boldsymbol{\ell}_F(2) = \mathbf{r}(1)$$
$$\ell_F(2) \geq 1$$

A full example





$$s_{\mathrm{I}} = 0, s_F = 0$$

$$\begin{aligned}
\boldsymbol{\ell}_F(0) &= \boldsymbol{\ell}_{\mathrm{I}}(0) - \mathbf{r}(0) \\
\boldsymbol{\ell}_F(1) &= \mathbf{r}(0) - \mathbf{r}(1) \\
\boldsymbol{\ell}_F(2) &= \mathbf{r}(1) \\
\ell_F(2) &\geq 1
\end{aligned}$$

$$\begin{aligned}
\mathbf{r}(1) - \mathbf{r}(0) &= 0 \\
-\mathbf{r}(1) &= 0 \\
\mathbf{r}(0) &= 0
\end{aligned}$$

Coverability via TSE

Algorithm

Input: TTS G; initial configuration v_{I}; final configuration v_F
Output: "uncoverable", or "coverable" + witness path

$\varphi := \mathcal{C}_L \wedge \mathcal{C}_F$
while $\exists m : m \models \varphi$
  $n_m := \sum_{\ell \in L} \boldsymbol{\ell}_{\mathrm{I}}(\ell)(m)$
  if $\mathrm{FSS}(G, n_m) =$ witness p
    return "coverable" + p
  $\varphi := \varphi \wedge (n > n_m)$
return "uncoverable"

Evaluation

- Compare against state-of-the-art unbounded-thread Boolean program checkers
- Investigate relation to tools targeting Petri nets
  - Conversion times from BP to Petri Nets ignored
  - Experimented with multiple translators

Experimental Setup

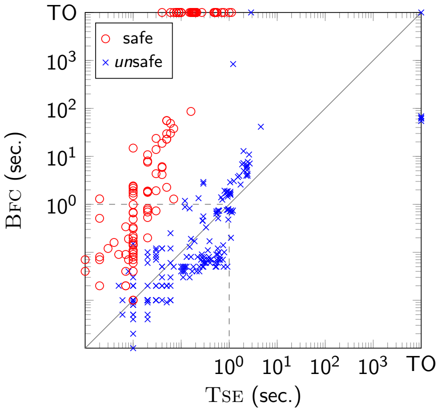
- Benchmark set consisting of 339 concurrent Boolean programs
- 135 of the Boolean programs are safe (i.e. uncoverable)
- Timeout: 30 minutes

Precision

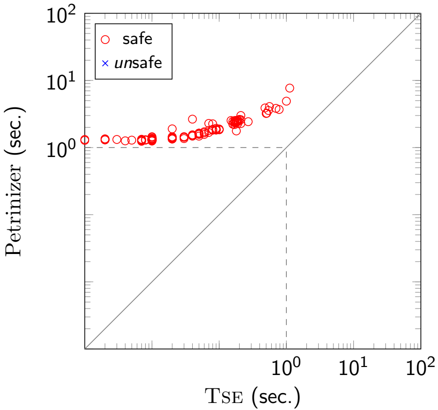
Success rate on proving Boolean programs either safe or unsafe

suite \ tools	TSE	Petrinizer	BFC	BFC-KM	IIC	MIST-AR	EEC	# instances
safe BP (%)	100	100	57.04	2.22	81.48	94.07	34.81	135
*un*safe BP (%)	97.55	–	99.02	98.04	62.75	12.75	18.63	204
total (%)	98.53	–	82.60	59.88	70.21	45.13	25.07	339

Efficiency



TO
10^3
10^2
10^1
10^0
safe
*un*safe
BFC (sec.)
10^0
10^1
10^2
10^3
TO
TSE (sec.)



10^2
10^1
10^0
safe
*un*safe
Petrinizer (sec.)
10^0
10^1
10^2
TSE (sec.)

Summary

A coverability technique that

- Can verify very often safe instances efficiently
- Can prove coverability in many unsafe instances

TSE

- An incomplete yet practical method using symbolic and explicit state techniques to verify safe and unsafe instances.
- `http://www.ccs.neu.edu/home/lpzun/tse/`