BV-LCTRSs Obtained from Concurrent Programs

Naoki Nishida Misaki Kojima

joint work with

Ayuka Matsumi and Kentaro Miura

Nagoya University

ARI final meeting, Kira-Onsen, Japan, February 21, 2024

Contents of This Talk

Contents of This Talk

1. LCTRSs for AITSs
2. LCTRSs from SV-Benchamrks
3. Remarks
4. Conclusion

LCTRSs for Asynchronous ITSs

Example (Peterson's mutual exclusion [Baier and Katoen, 2008])

Process 1: $\to$ noncrit$_1$ $\xrightarrow{\langle b_1,x\rangle:=\langle \mathsf{true},2\rangle}$ wait$_1$ $\xrightarrow{x=1\vee\neg b_2}$ crit$_1$, $b_1:=\mathsf{false}$

Process 2: $\to$ noncrit$_2$ $\xrightarrow{\langle b_2,x\rangle:=\langle \mathsf{true},1\rangle}$ wait$_2$ $\xrightarrow{x=2\vee\neg b_1}$ crit$_2$, $b_2:=\mathsf{false}$

- b_i indicates that Process i wants to enter the critical section
- x indicates that Process x has priority for the critical section
- Asynchronous ITS for Processes 1 & 2 is represented by

$$\left\{\begin{array}{llll}
\mathsf{cnfg}(\mathsf{noncrit}_1, p_2, & b_1, b_2, x) \to \mathsf{cnfg}(\mathsf{wait}_1, & p_2, & b_1', b_2, x') \quad [\,b_1' = \mathsf{true} \wedge x' = 2\,] \\
\mathsf{cnfg}(\mathsf{wait}_1, & p_2, & b_1, b_2, x) \to \mathsf{cnfg}(\mathsf{crit}_1, & p_2, & b_1, b_2, x\ ) \quad [\,x = 1 \vee \neg b_2\,] \\
\mathsf{cnfg}(\mathsf{crit}_1, & p_2, & b_1, b_2, x) \to \mathsf{cnfg}(\mathsf{noncrit}_1, p_2, & & b_1', b_2, x\ ) \quad [\,b_1' = \mathsf{false}\,] \\
\mathsf{cnfg}(p_1, & \mathsf{noncrit}_2, & b_1, b_2, x) \to \mathsf{cnfg}(p_1, & \mathsf{wait}_2, & b_1, b_2', x') \quad [\,b_2' = \mathsf{true} \wedge x' = 1\,] \\
\mathsf{cnfg}(p_1, & \mathsf{wait}_2, & b_1, b_2, x) \to \mathsf{cnfg}(p_1, & \mathsf{crit}_2, & b_1, b_2, x\ ) \quad [\,x = 2 \vee \neg b_1\,] \\
\mathsf{cnfg}(p_1, & \mathsf{crit}_2, & b_1, b_2, x) \to \mathsf{cnfg}(p_1, & \mathsf{noncrit}_2, & b_1, b_2', x\ ) \quad [\,b_1' = \mathsf{false}\,]
\end{array}\right\}$$

- Initial states: $\langle \mathsf{cnfg}(\mathsf{noncrit}_1, \mathsf{noncrit}_2, \mathsf{false}, \mathsf{false}, x) \mid x = 1 \vee x = 2\rangle$
- This LCTRS is confluent

In ARI Format

```
(format LCTRS :smtlib 2.6)
(theory Ints)

(sort Loc)
(sort State)

(fun state (-> Loc Loc Bool Bool Int State))
(fun noncrit0 Loc)
(fun wait0 Loc)
(fun crit0 Loc)
(fun noncrit1 Loc)
(fun wait1 Loc)
(fun crit1 Loc)
(fun success State)
(fun error State)

(rule (state noncrit0 p1 b0 b1 x) (state wait0 p1 b01 b1 x1)
       :guard (and b01 (= x1 1)) :var ((x1 Int)) )
(rule (state wait0 p1 b0 b1 x) (state crit0 p1 b0 b1 x)      :guard (or (= x 0) (not b1)) )
(rule (state crit0 p1 b0 b1 x) (state noncrit0 p1 b01 b1 x)  :guard (not b01) )
(rule (state p0 noncrit1 b0 b1 x) (state p0 wait1 b0 b11 x1)
       :guard (and b11 (= x1 0)) :var ((x1 Int)) )
(rule (state p0 wait1 b0 b1 x) (state p0 crit1 b0 b1 x)      :guard (or (= x 1) (not b0)) )
(rule (state p0 crit1 b0 b1 x) (state p0 noncrit1 b0 b11 x)  :guard (not b11) )
```


In ARI Format (BV version)

```
(format LCTRS :smtlib 2.6)
(theory FixedSizeBitVectors)

(sort Loc)
(sort State)

(fun state (-> Loc Loc Bool Bool (_ BitVec 1) State))
(fun noncrit0 Loc)
(fun wait0 Loc)
(fun crit0 Loc)
(fun noncrit1 Loc)
(fun wait1 Loc)
(fun crit1 Loc)
(fun success State)
(fun error State)

(rule (state noncrit0 p1 b0 b1 x) (state wait0 p1 b01 b1 x1)
       :guard (and b01 (= x1 #b1)) :var ((x1 (_ BitVec 1))) )
(rule (state wait0 p1 b0 b1 x) (state crit0 p1 b0 b1 x)       :guard (or (= x #b0) (not b1)) )
(rule (state crit0 p1 b0 b1 x) (state noncrit0 p1 b01 b1 x)   :guard (not b01) )
(rule (state p0 noncrit1 b0 b1 x) (state p0 wait1 b0 b11 x1)
       :guard (and b11 (= x1 #b0)) :var ((x1 (_ BitVec 1))) )
(rule (state p0 wait1 b0 b1 x) (state p0 crit1 b0 b1 x)       :guard (or (= x #b1) (not b0)) )
(rule (state p0 crit1 b0 b1 x) (state p0 noncrit1 b0 b11 x)   :guard (not b11) )
```


All-Path Reachability [Ştefan Ciobâcă and Lucanu, 2018]

Execution path of transition system $(M, \rightharpoonup)$

State transition sequence that is either

- finite and ends with an irreducible state

  or
- infinite

All-path reachability (APR) problem

$$P \Rightarrow Q$$

where P and Q are sets of states in M

Demonical validity of $P \Rightarrow Q$ (w.r.t. $(M, \rightharpoonup)$)

Every finite execution path starting from a state in P includes a state in Q

All-Path Reachability Problems of LCTRSs

[Ştefan Ciobâcă and Lucanu, 2018]

Constrained term

$$\langle t \mid \phi \rangle$$

where t is a term and ϕ is a constraint

- $\langle t \mid \phi \rangle$ represents the set of ground instance $t\theta$ such that θ satisfies ϕ

APR problem of LCTRS $\mathcal{R}$

$$\langle s \mid \phi \rangle \Rightarrow \langle t \mid \psi \rangle$$

- We only deal with APR problems of the form $\langle s \mid \phi \rangle \Rightarrow \langle c \mid \text{true} \rangle$ such that
  - c is a constant normal form
- Write $\langle s \mid \phi \rangle \Rightarrow c$ for $\langle s \mid \phi \rangle \Rightarrow \langle c \mid \text{true} \rangle$
- Proof systems for demonical validity of APR problems have been proposed

  [Ştefan Ciobâcă and Lucanu, 2018, Kojima and Nishida, 2023]

From Non-Occurrence of Error States to APR Problem

[Kojima and Nishida, 2022]

- To show non-occurrence of a runtime error, we have to show that any initial execution path doesn't reach an error state

Reduction Method

- Add a rule to reduce any state to a constant success

$$\mathsf{cnfg}(\vec{x}) \to \mathsf{success}$$

- Add rules to reduce error states $\langle u \mid \psi \rangle$ to a constant error

$$u \to \mathsf{error} \quad [\psi]$$

- Reduce to APR problem $\{\text{initial states}\} \Rightarrow \mathsf{success}$

Example (cont'd)

$$\left\{\begin{array}{llll}
\mathsf{cnfg}(\mathsf{noncrit}_1, p_2, & b_1, b_2, x) \to \mathsf{cnfg}(\mathsf{wait}_1, & p_2, & b_1', b_2, x') & [\,b_1' = \mathsf{true} \wedge x' = 2\,] \\
\mathsf{cnfg}(\mathsf{wait}_1, & p_2, b_1, b_2, x) \to \mathsf{cnfg}(\mathsf{crit}_1, & p_2, & b_1, b_2, x\,) & [\,x = 1 \vee \neg b_2\,] \\
\mathsf{cnfg}(\mathsf{crit}_1, & p_2, b_1, b_2, x) \to \mathsf{cnfg}(\mathsf{noncrit}_1, & p_2, & b_1', b_2, x\,) & [\,b_1' = \mathsf{false}\,] \\
\mathsf{cnfg}(p_1, & \mathsf{noncrit}_2, b_1, b_2, x) \to \mathsf{cnfg}(p_1, & \mathsf{wait}_2, & b_1, b_2', x') & [\,b_2' = \mathsf{true} \wedge x' = 1\,] \\
\mathsf{cnfg}(p_1, & \mathsf{wait}_2, & b_1, b_2, x) \to \mathsf{cnfg}(p_1, & \mathsf{crit}_2, & b_1, b_2, x\,) & [\,x = 2 \vee \neg b_1\,] \\
\mathsf{cnfg}(p_1, & \mathsf{crit}_2, & b_1, b_2, x) \to \mathsf{cnfg}(p_1, & \mathsf{noncrit}_2, & b_1, b_2', x\,) & [\,b_1' = \mathsf{false}\,] \\
\mathsf{cnfg}(p_1, & p_2, & b_1, b_2, x) \to \mathsf{success} & & & \\
\mathsf{cnfg}(\mathsf{crit}_1, & \mathsf{crit}_2, & b_1, b_2, x) \to \mathsf{error} & & &
\end{array}\right.$$

- APR problem for race-freedom is

$$\langle \mathsf{cnfg}(\mathsf{noncrit}_1, \mathsf{noncrit}_2, \mathsf{false}, \mathsf{false}, x) \mid x = 1 \vee x = 2 \rangle \Rightarrow \mathsf{success}$$

- This LCTRS is not confluent

In ARI Format

```
(format LCTRS :smtlib 2.6)
(theory Ints)

(sort Loc)
(sort State)

(fun state (-> Loc Loc Bool Bool Int State))
(fun noncrit0 Loc)
(fun wait0 Loc)
(fun crit0 Loc)
(fun noncrit1 Loc)
(fun wait1 Loc)
(fun crit1 Loc)
(fun success State)
(fun error State)

(rule (state noncrit0 p1 b0 b1 x) (state wait0 p1 b01 b1 x1)
       :guard (and b01 (= x1 1)) :var ((x1 Int)) )
(rule (state wait0 p1 b0 b1 x) (state crit0 p1 b0 b1 x)      :guard (or (= x 0) (not b1)) )
(rule (state crit0 p1 b0 b1 x) (state noncrit0 p1 b01 b1 x)  :guard (not b01) )
(rule (state p0 noncrit1 b0 b1 x) (state p0 wait1 b0 b11 x1)
       :guard (and b11 (= x1 0)) :var ((x1 Int)) )
(rule (state p0 wait1 b0 b1 x) (state p0 crit1 b0 b1 x)      :guard (or (= x 1) (not b0)) )
(rule (state p0 crit1 b0 b1 x) (state p0 noncrit1 b0 b11 x)  :guard (not b11) )

(rule (state p0 p1 b0 b1 x) success )
(rule (state crit0 crit1 b0 b1 x) error )
```


In ARI Format (BV version)

```
(format LCTRS :smtlib 2.6)
(theory FixedSizeBitVectors)

(sort Loc)
(sort State)

(fun state (-> Loc Loc Bool Bool (_ BitVec 1) State))
(fun noncrit0 Loc)
(fun wait0 Loc)
(fun crit0 Loc)
(fun noncrit1 Loc)
(fun wait1 Loc)
(fun crit1 Loc)
(fun success State)
(fun error State)

(rule (state noncrit0 p1 b0 b1 x) (state wait0 p1 b01 b1 x1)
       :guard (and b01 (= x1 #b1)) :var ((x1 (_ BitVec 1))) )
(rule (state wait0 p1 b0 b1 x) (state crit0 p1 b0 b1 x)      :guard (or (= x #b0) (not b1)) )
(rule (state crit0 p1 b0 b1 x) (state noncrit0 p1 b01 b1 x)  :guard (not b01) )
(rule (state p0 noncrit1 b0 b1 x) (state p0 wait1 b0 b11 x1)
       :guard (and b11 (= x1 #b0)) :var ((x1 (_ BitVec 1))) )
(rule (state p0 wait1 b0 b1 x) (state p0 crit1 b0 b1 x)      :guard (or (= x #b1) (not b0)) )
(rule (state p0 crit1 b0 b1 x) (state p0 noncrit1 b0 b11 x)  :guard (not b11) )

(rule (state p0 p1 b0 b1 x) success )
(rule (state crit0 crit1 b0 b1 x) error )
```

Contents of This Talk

LCTRSs from SV-Benchamrks

Example (c/recursive-simple/id_b3_o2-1.c)

```
extern int __VERIFIER_nondet_int();
extern void abort(void);
extern void __assert_fail(const char *, const char *, unsigned int, const char *) __
void reach_error() { __assert_fail("0", "id_b3_o2-1.c", 4, "reach_error"); }

int id(int x) {
  if (x==0) return 0;
  int ret = id(x-1) + 1;
  if (ret > 3) return 3;
  return ret;
}

int main(void) {
  int input = __VERIFIER_nondet_int();
  int result = id(input);
  if (result == 2) {
    ERROR: {reach_error();abort();}
  }
}
```


In ARI Format

```
(format LCTRS :smtlib 2.6)
(theory FixedSizeBitVectors)
(sort Cstack)
(sort Cnfg)
(sort Frame)
(fun cnfg (-> Cstack Cnfg))
(fun cstack (-> Frame Cstack Cstack))
(fun empty Cstack)
(fun id (-> (_ BitVec 32) Frame))
(fun id_7 (-> (_ BitVec 32) Frame))
(fun id_8 (-> (_ BitVec 32) Frame))
(fun id_8call (-> (_ BitVec 32) Frame))
(fun id_9 (-> (_ BitVec 32) (_ BitVec 32) Frame))
(fun id_return (-> (_ BitVec 32) Frame))
(fun main Frame)
(fun main_14 Frame)
(fun main_15 (-> (_ BitVec 32) Frame))
(fun main_15call (-> (_ BitVec 32) Frame))
(fun main_16 (-> (_ BitVec 32) (_ BitVec 32) Frame))
(fun main_19 (-> (_ BitVec 32) (_ BitVec 32) Frame))
(fun success Cnfg)
(fun error Cnfg)
(fun overflow Cnfg)
(fun underflow Cnfg)
(fun reach Cnfg)
(rule (cnfg (cstack (id x) s)) (cnfg (cstack (id_7 x) s)) )
(rule (cnfg (cstack (id_7 x) s)) (cnfg (cstack (id_return #x00000000) s))
        :guard (= x #x00000000) )
(rule (cnfg (cstack (id_7 x) s)) (cnfg (cstack (id_8 x) s))
        :guard (distinct x #x00000000) )
```


Demo

-

Contents of This Talk

1. LCTRSs for AITSs
2. LCTRSs from SV-Benchamrks
3. Remarks
4. Conclusion

Remarks

- `bvsub` is not included in `FixedSizeBitVectors`
- Should `x1` in the following rule be typable?

```
(rule (state noncrit0 p1 b0 b1 x) (state wait0 p1 b01 b1 x1)
        :guard (and b01 (= x1 1)) )
```

  - NOTE: My checker succeeds in typing the following `x1`:

```
(rule (state noncrit0 p1 b0 b1 x) (state wait0 p1 b01 b1 x1)
        :guard (and b01 (= 1 x1)) )
```

- 2,570 BV-LCTRSs are automatically obtained from C programs in SV-Benchmarks [Miura, 2024]
  - Not in ARI format yet

Contents of This Talk

1. LCTRSs for AITSs
2. LCTRSs from SV-Benchamrks
3. Remarks
4. Conclusion

Conclusion

Summary

- LCTRSs obtained from C programs

Future Work

- Implementation

References

Baier, C. and Katoen, J. (2008).
Principles of model checking.
MIT Press.

Ștefan Ciobâcă and Lucanu, D. (2018).
A coinductive approach to proving reachability properties in logically constrained term rewriting systems.
In Galmiche, D., Schulz, S., and Sebastiani, R., editors, *Proceedings of the 9th International Joint Conference on Automated Reasoning*, volume 10900 of *Lecture Notes in Computer Science*, pages 295–311. Springer.

Kojima, M. and Nishida, N. (2022).
On reducing non-occurrence of specified runtime errors to all-path reachability problems of constrained rewriting.
In Ciobaca, S. and Nakano, K., editors, *Informal Proceedings of the 9th International Workshop on Rewriting Techniques for Program Transformations and Evaluation*, pages 1–16.

Kojima, M. and Nishida, N. (2023).
Reducing non-occurrence of specified runtime errors to all-path reachability problems of constrained rewriting.
Journal of Logical and Algebraic Methods in Programming.
to appear.