15th International Conference on Information Systems Security

ICISS 2019 | Hyderabad, India | December 16-20, 2019

Secure Information Flow Analysis Using the PRISM Model Checker

Ali A. Noroozi https://alianoroozi.github.io/

Khayyam Salehi, Jaber Karimpour, Ayaz Isazadeh

University of Tabriz, Iran

December 2019

Contents

Introduction

Secure information flow

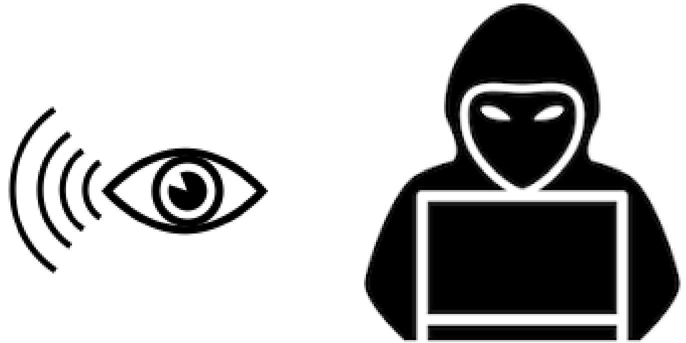
Introduction

Information flow

secret variables

↓

public variables



Information flow

`l:=h` 100% leakage

Introduction

Information flow

```
if h>0 then l:=-5 else l:=5 fi
```

1 bit leakage

Introduction

Secure information flow analysis

Program model

Security property

Verification method

Introduction

Security property for concurrent programs

Observational determinism

Introduction

Challenges of existing definitions of observational determinism

Scheduler-independent

Imprecise

Introduction

Verifying observational determinism

Type systems

Logics

Algorithmic verification

Introduction

Challenges of OD verification methods

Restrictive

Not extensible

Non-automated

Not scalable

Contributions

Proposing an approach for analyzing

secure information flow of concurrent programs

Contributions

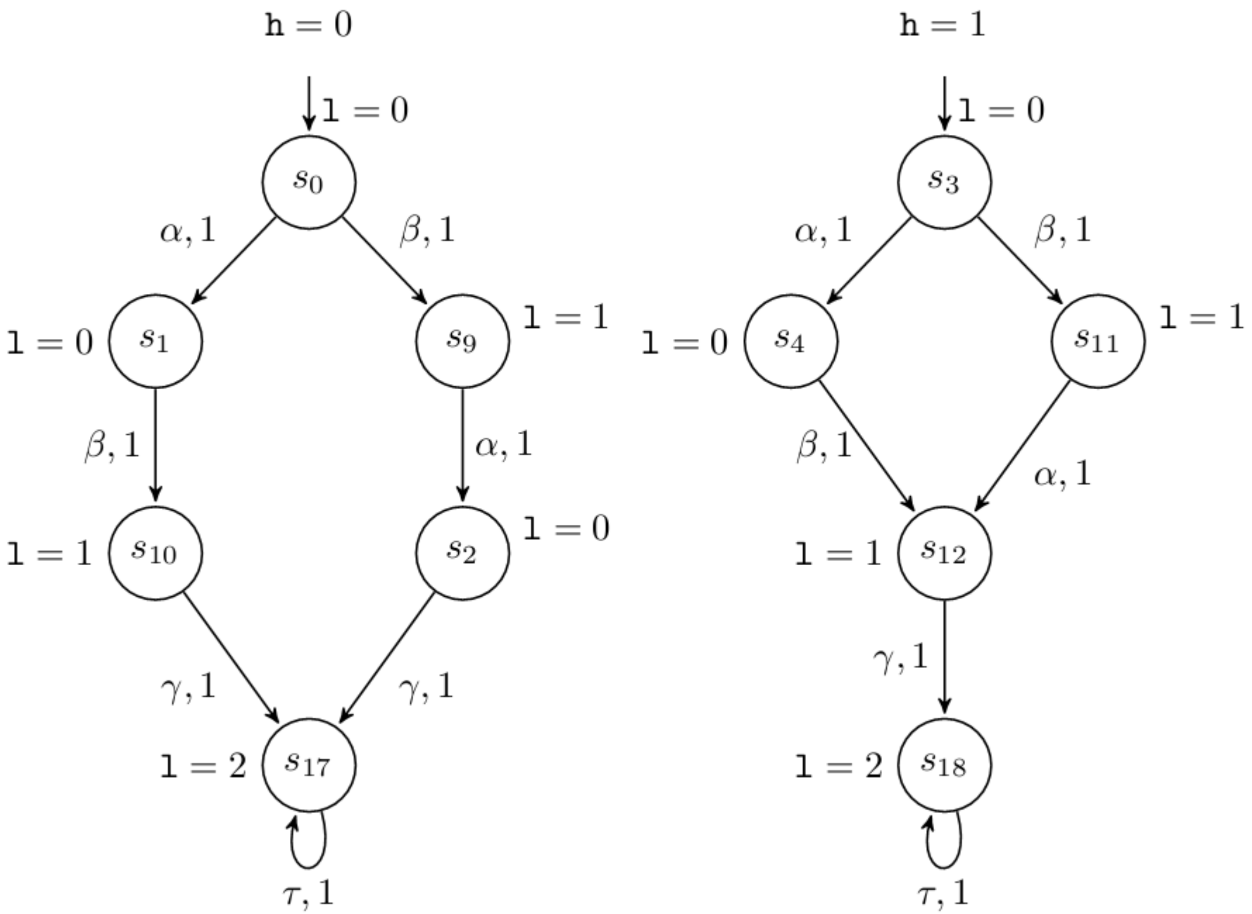
- Formally modeling concurrent programs
  by Markov processes and probabilistic schedulers
- A formal definition of observational determinism
- Algorithms to verify observational determinism
- An automated tool PRISM-Leak

Contents

Background

Markov Decision Process

$$\mathcal{M}^P = (S, Act, \mathbf{P}, \zeta, Val_L, V_L)$$



Background

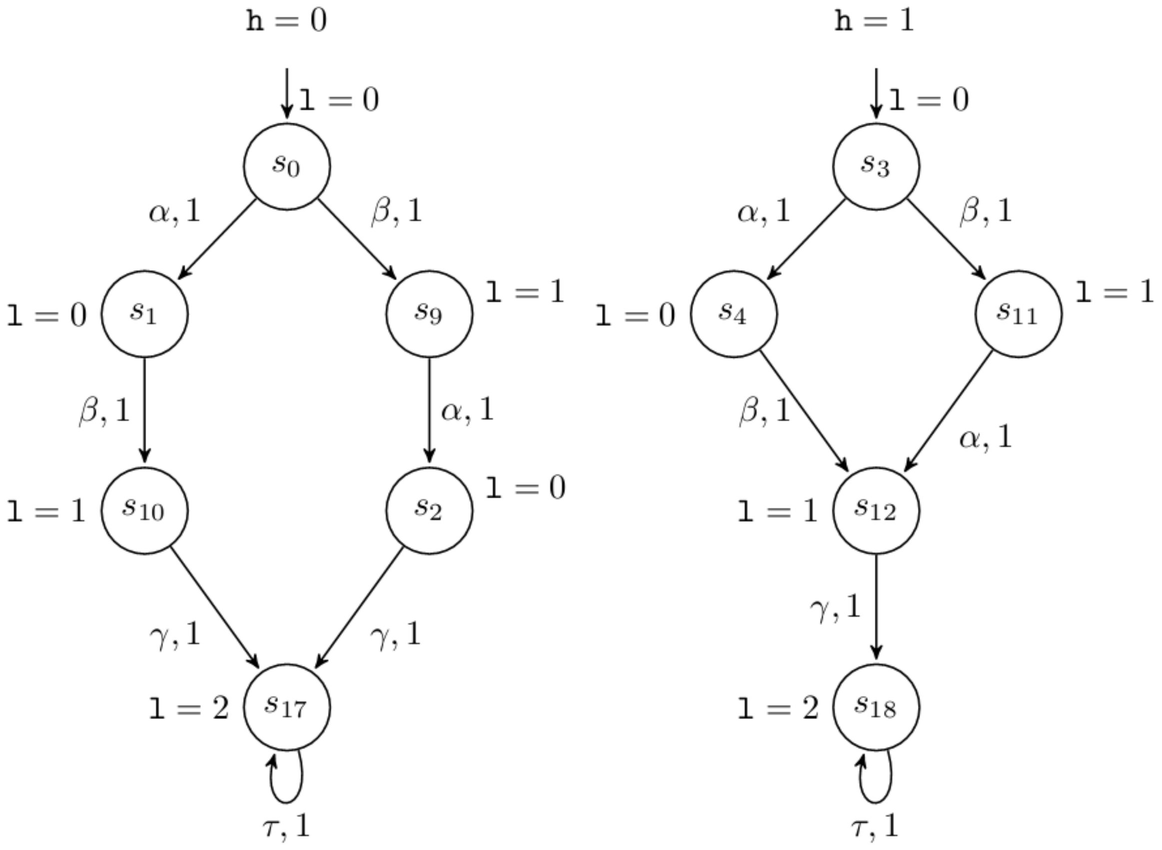
Memoryless probabilistic scheduler

$$\delta : S \rightarrow \mathcal{D}(Act)$$

$$\delta(s) \in \mathcal{D}(Act(s)) \text{ for all } s \in S$$

Background

Memoryless probabilistic scheduler δ



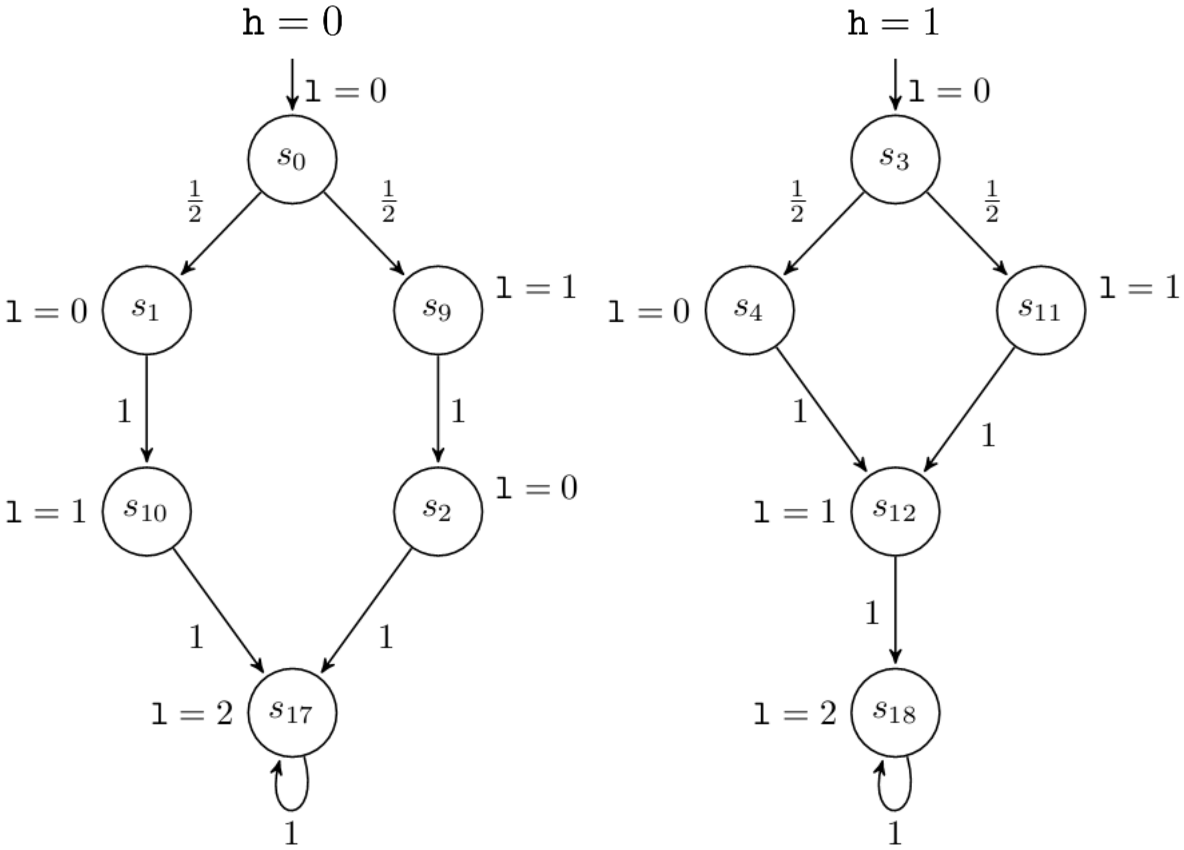
$$\delta(s_0) = \{\alpha \mapsto \tfrac{1}{2}, \beta \mapsto \tfrac{1}{2}\}$$

$$\delta(s_1) = \{\beta \mapsto 1\}$$

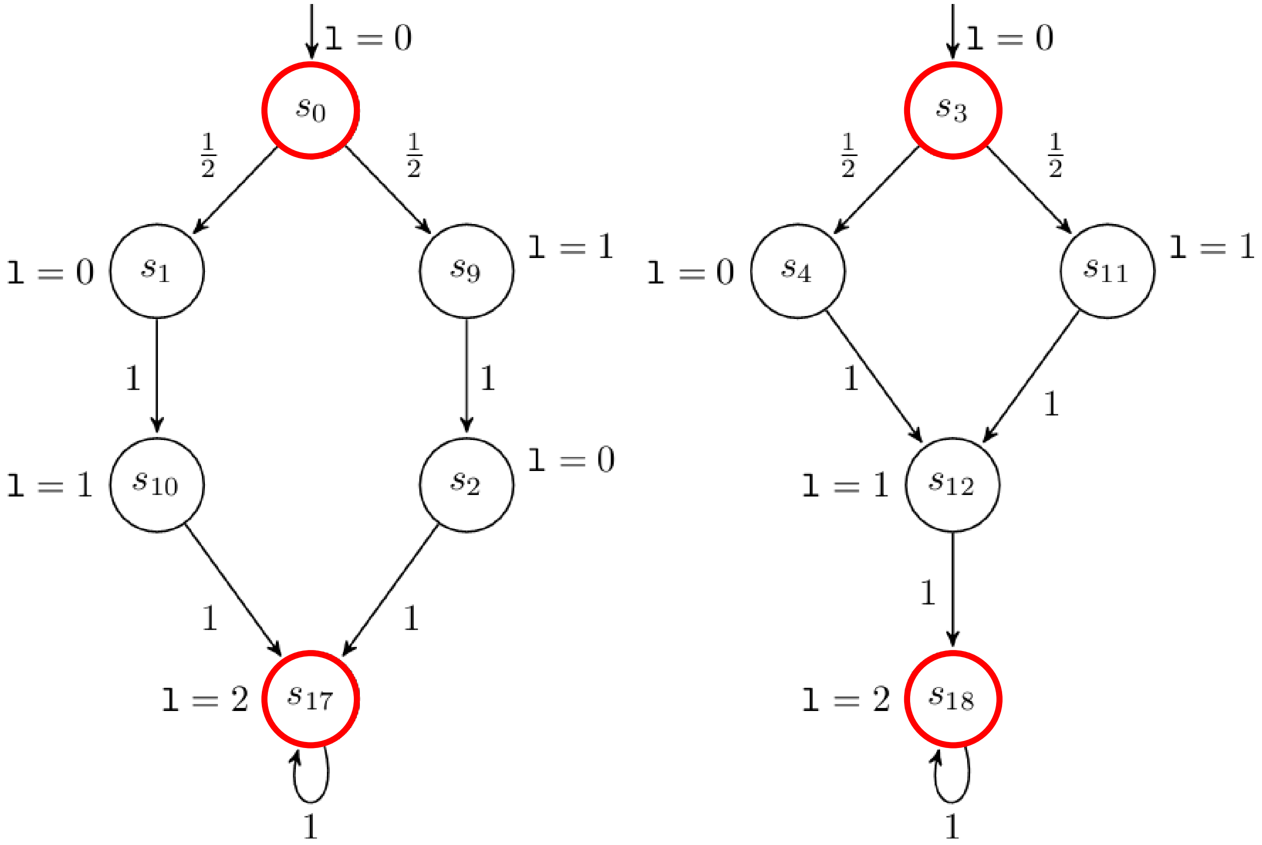
Background

Markov Chain

$$\mathcal{M}_\delta^P = (S, \mathbf{P}_\delta, \zeta, Val_L, V_L)$$



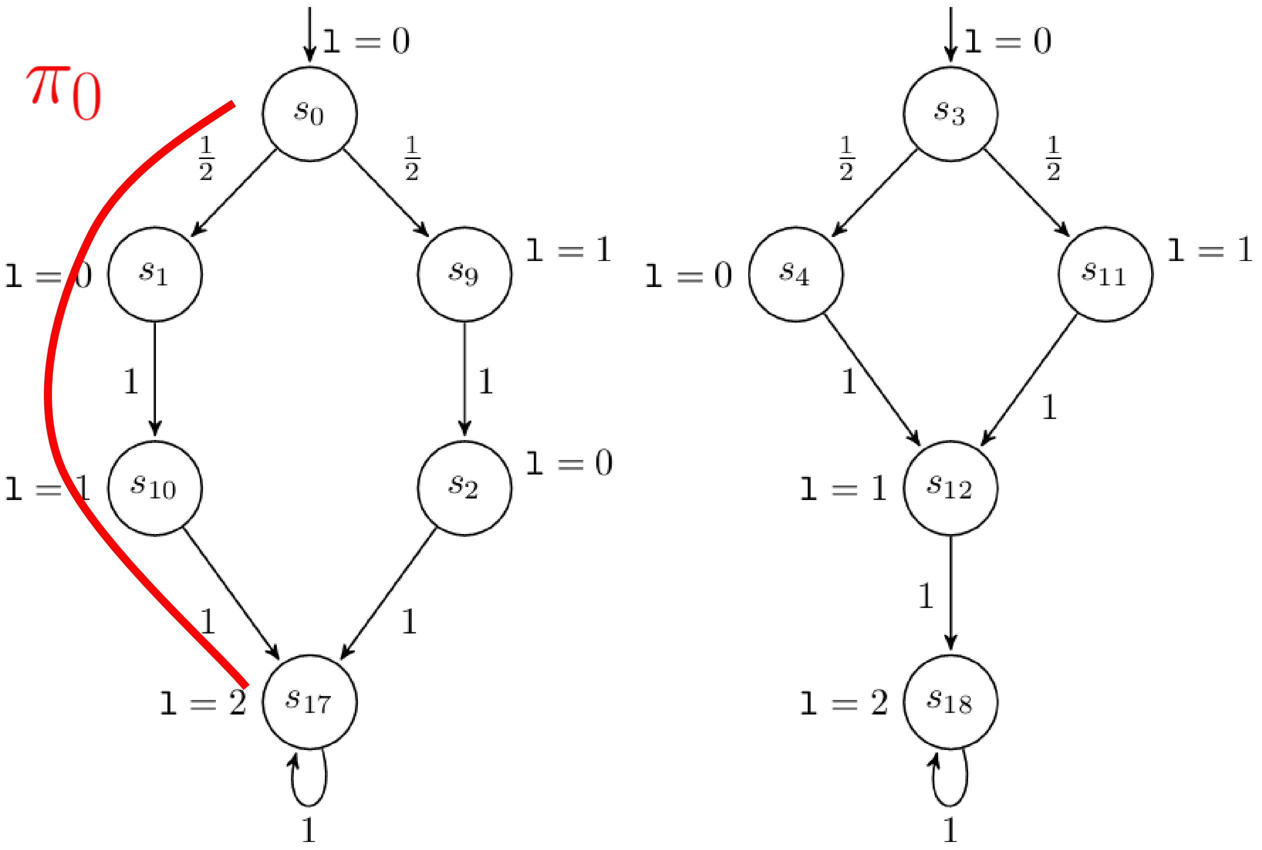
Initial and final states



Background

Path

$\pi_0 = s_0\ s_1\ s_{10}\ s_{17}$



Background

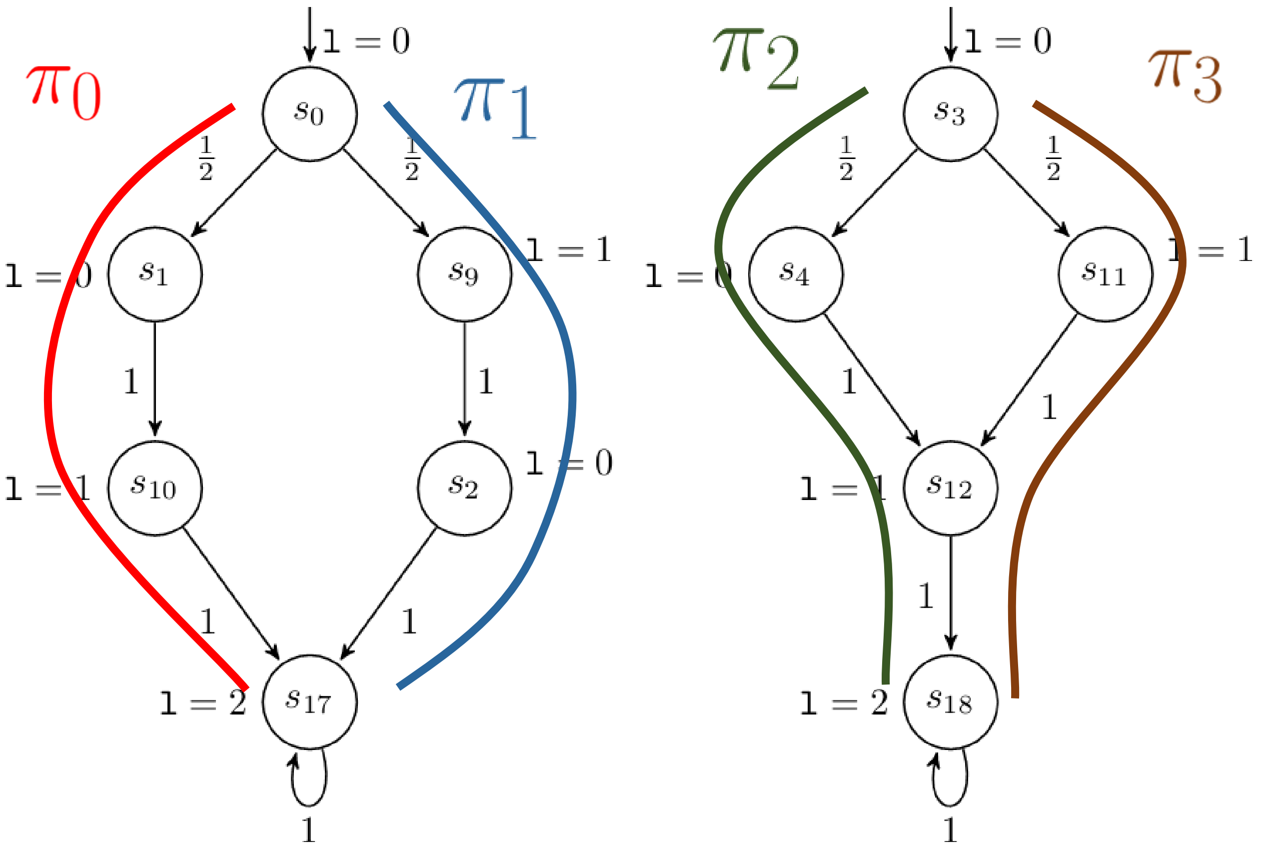
Path

$\pi_0 = s_0\ s_1\ s_{10}\ s_{17}$

$\pi_1 = s_0\ s_9\ s_2\ s_{17}$

$\pi_2 = s_3\ s_4\ s_{12}\ s_{18}$

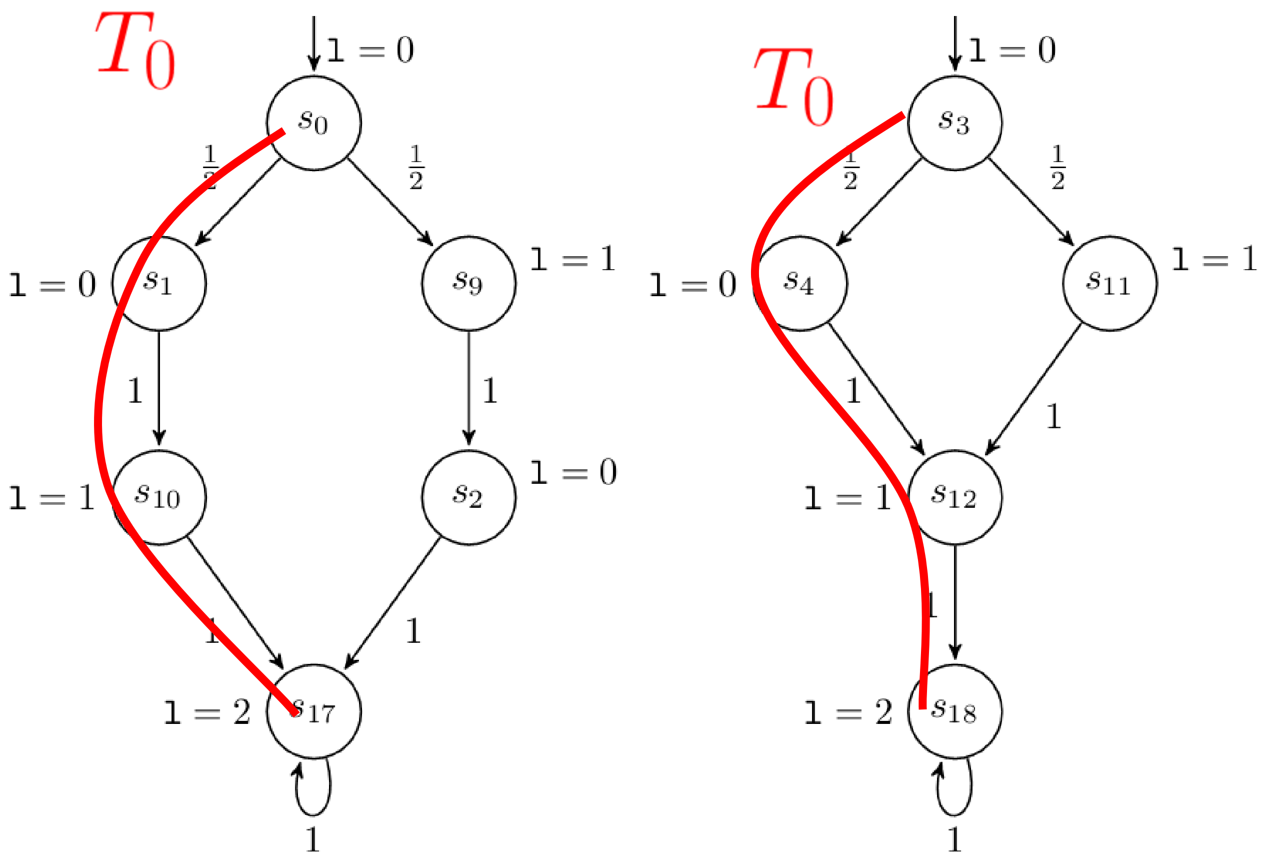
$\pi_3 = s_3\ s_{11}\ s_{12}\ s_{18}$



Background

Trace

$$T_0 = [0, 0, 1, 2^\omega]$$



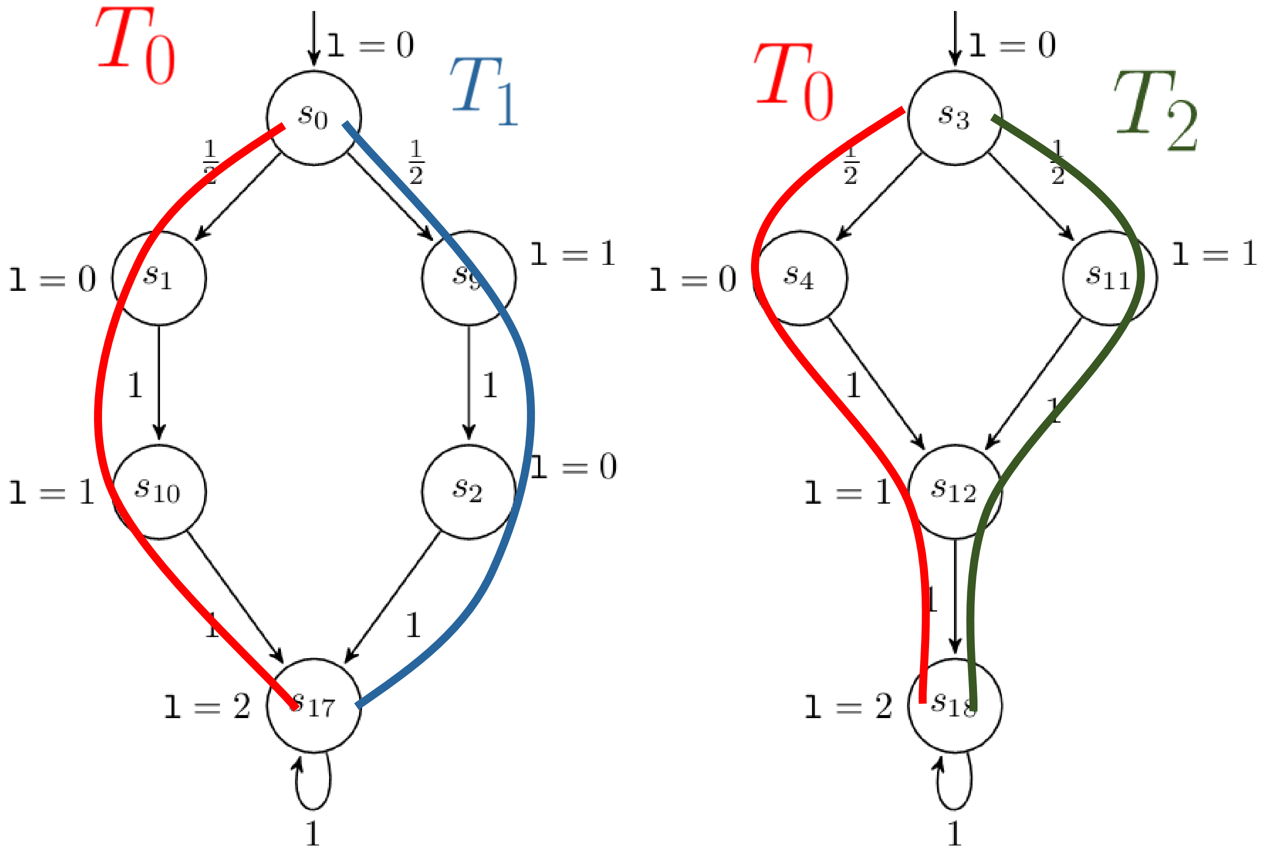
Background

Trace

$T_0 = [0, 0, 1, 2^\omega]$

$T_1 = [0, 1, 0, 2^\omega]$

$T_2 = [0, 1, 1, 2^\omega]$



Background

Stutter equivalence

$$T_0 = [0, 1, 1, 2^\omega] \longrightarrow T_0^{sf} = [0, 1, 2]$$

$$\triangleq \qquad\qquad =$$

$$T_1 = [0, 0, 1, 2, 2^\omega] \longrightarrow T_1^{sf} = [0, 1, 2]$$

Background

Stutter and prefix equivalence

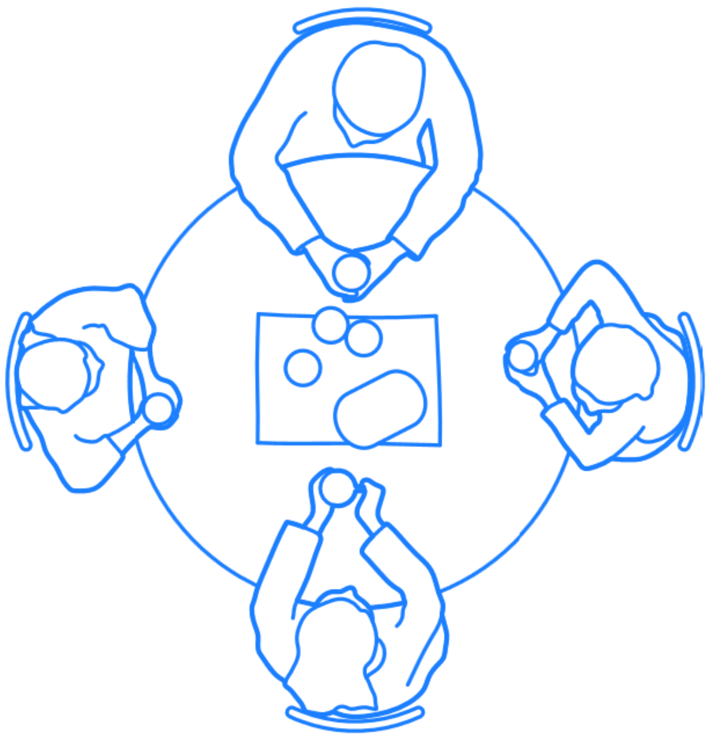
$$T_0 = [0, 2, 1, 1, 4, 4^\omega] \longrightarrow T_0^{sf} = [0, 2, 1, 4]$$

$$\triangleq_p \qquad\qquad \text{prefix}$$

$$T_1 = [0, 0, 0, 2, 1^\omega] \longrightarrow T_1^{sf} = [0, 2, 1]$$

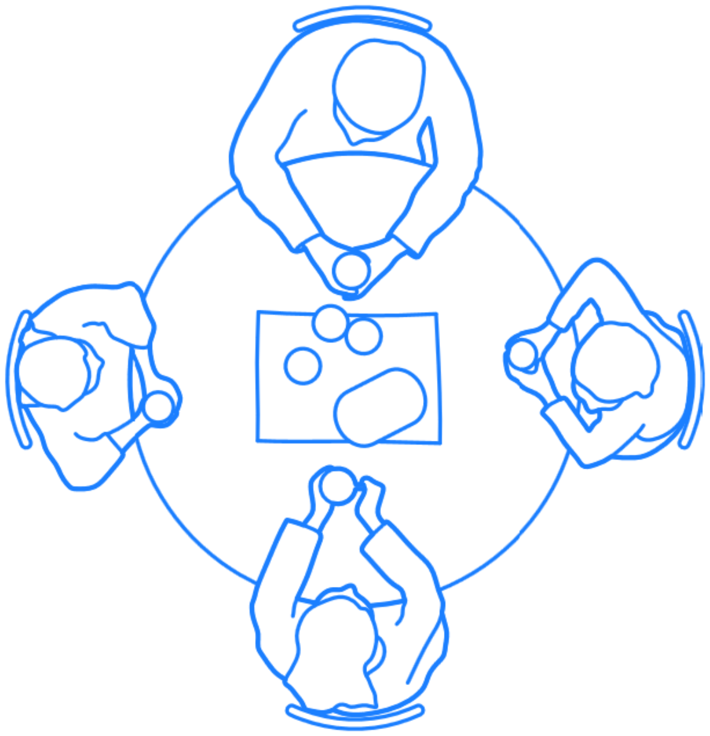
Background

Dining cryptographers protocol



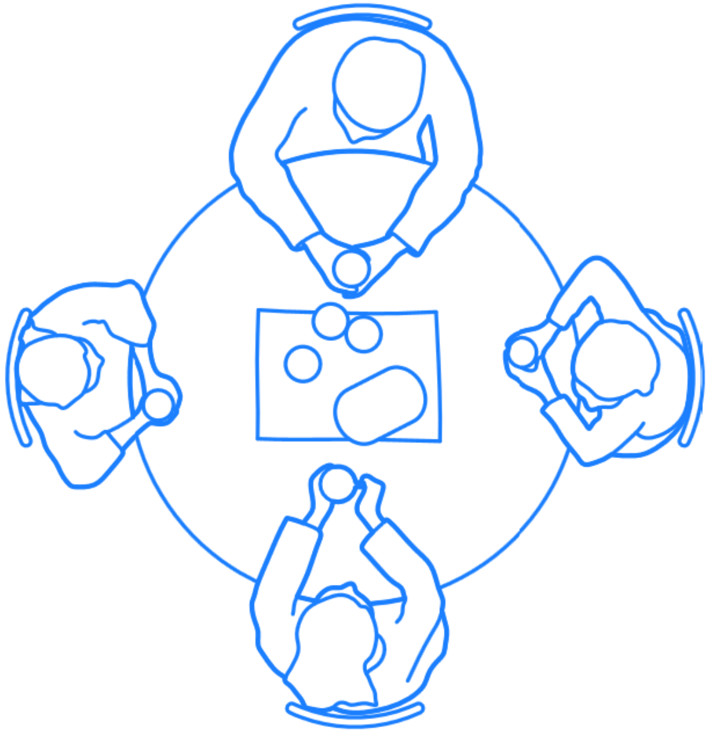
Background

Dining cryptographers protocol



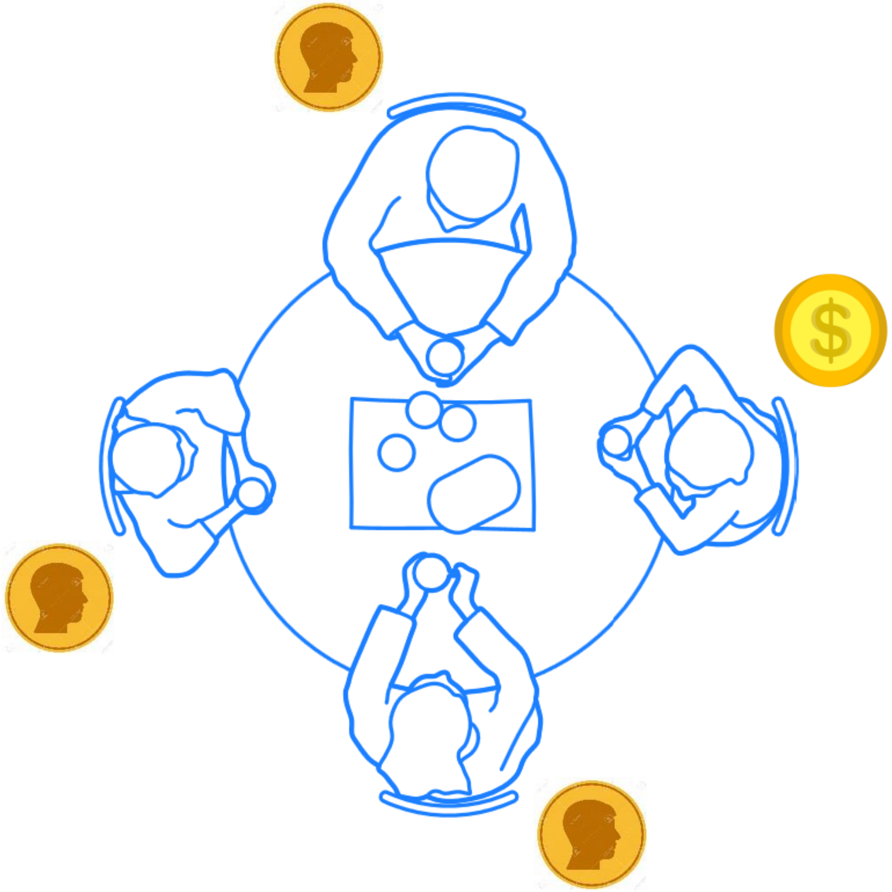
Background

Dining cryptographers protocol



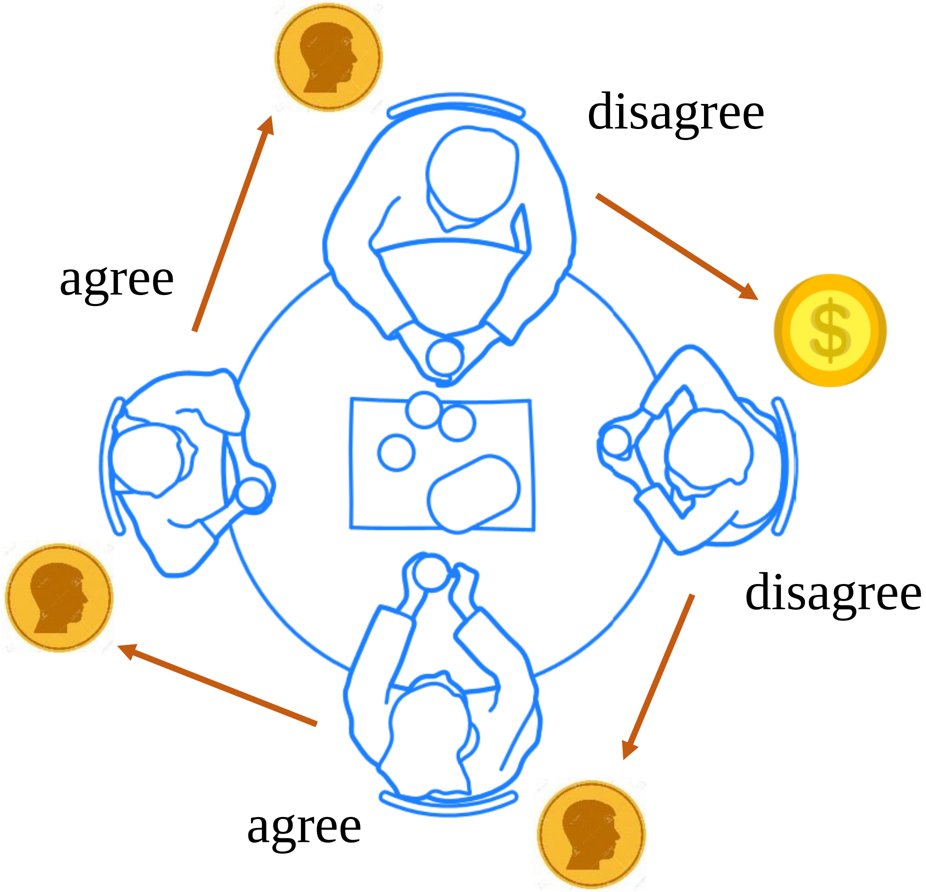
Background

Dining cryptographers protocol



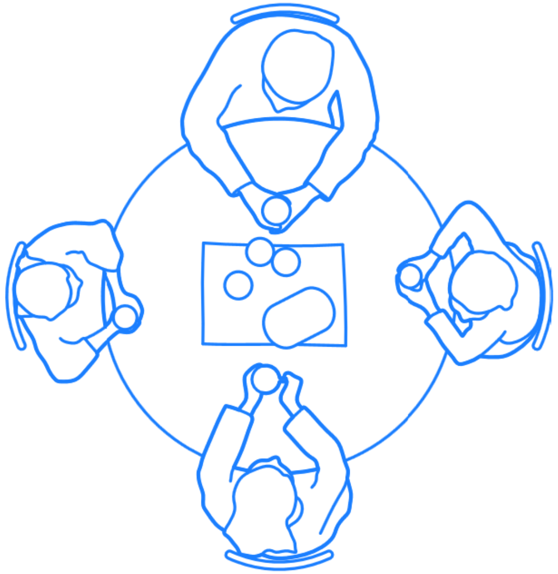
Background

Dining cryptographers protocol



Background

Dining cryptographers protocol

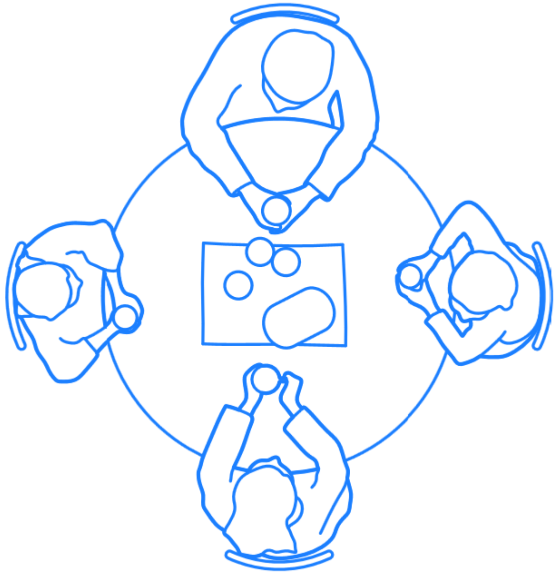


Case 1: $Val_{payer} = \{c_1, \ldots, c_n\}$

secure

Background

Dining cryptographers protocol



Case 2: $Val_{payer} = \{m, c_1, \ldots, c_n\}$

insecure

Contents

Related work

Observational determinism

Zdancewic and Myers, 2003

$$\forall T, T' \in Traces(\mathcal{M}^{\mathsf{P}}_{\delta}), l \in L.\ \ T_{|l} \triangleq_p T'_{|l}$$

Verification by type systems

Related work

Observational determinism

- Huisman and Blondeel, 2012
- Karimpour et al., 2015
- Dabaghchian and Azgomi, 2015

$$\forall T, T' \in Traces(\mathcal{M}_\delta^P).\ T_{|L} \triangleq T'_{|L};$$

Logic-based and algorithmic model checking

Related work

Observational determinism

Ngo et al., 2014

SSOD-1: $\forall T, T' \in Traces(\mathcal{M}_{\delta}^{P}), l \in L.\ T_{|l} \triangleq T'_{|l}$

SSOD-2: $\forall T \in Traces(s_0), \exists T' \in Traces(s'_0).\ T_{|L} \triangleq T'_{|L}$

Algorithmic verification

Related work

Observational determinism

Snelting et al., 2015-2019

JOANA tool: LSOD, RLSOD, iRLSOD

Program dependence graph

Related work

Information leakage tools:

- LeakWatch: Chothia et al., 2014
- QUAL: Biondi et al., 2015
- HyLeak: Biondi et al., 2017

Contents

The proposed approach

1. Specifying observational determinism

2. Verifying observational determinism

4 The proposed approach

Specifying observational determinism

$$OD_1 : \forall T, T' \in Traces(\mathcal{M}_\delta^P), l \in L.\ T_{|l} \triangleq_p T'_{|l},$$

$$OD_2 : \forall T \in Traces(s_0), \exists T' \in Traces(s'_0).\ T_{|L} \triangleq T'_{|L}.$$

The proposed approach

Algorithm 1 Verifying OD_1

Input: finite MC $\mathcal{M}_\delta^{\mathsf{P}}$

Output: *true* if the program satisfies OD_1; otherwise, *false*

// *Consider an empty string as a witness for each public variable*

for l **in** L **do**

  Let $witnesses[l]$ be an empty string;

Let π be an empty list of states for storing a path;

for s_0 **in** $Init(\mathcal{M}_\delta^{\mathsf{P}})$ **do**

  result = explorePathsOD1(s_0, π, $witnesses$);

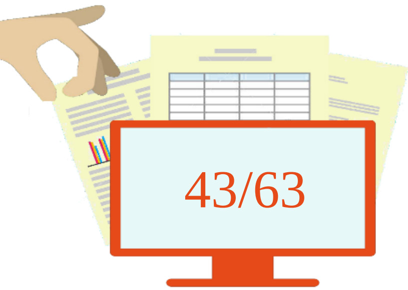
  if not result **then**

    return *false*;

return *true*;

The proposed approach

function explorePathsOD1(s, π, $witnesses$)
    π.add(s); // *add state s to the current path from the initial state*
    if s is a final state **then** // *found a path stored in π*
        for l **in** L **do**
            $T_{|l} = trace_{|l}(\pi)$;
            Remove stutter data from $T_{|l}$, yielding stutter-free trace $T_{|l}^{sf}$;
            $T_w = witnesses[l]$;
            if $length(T_{|l}^{sf}) \leq length(T_w)$ **then**
                if $T_{|l}^{sf}$ is not prefix of T_w **then**
                    return *false*;
            else
                if T_w is not prefix of $T_{|l}^{sf}$ **then**
                    return *false*;
                else
                    $witnesses[l] = T_{|l}^{sf}$;
    else
        for s' **in** $Post(s)$ **do**
            result = explorePathsOD1(s', π, $witnesses$);
            if not result **then**
                return *false*;
    π.pop(); // *done exploring from s, so remove it from π*
    return *true*;

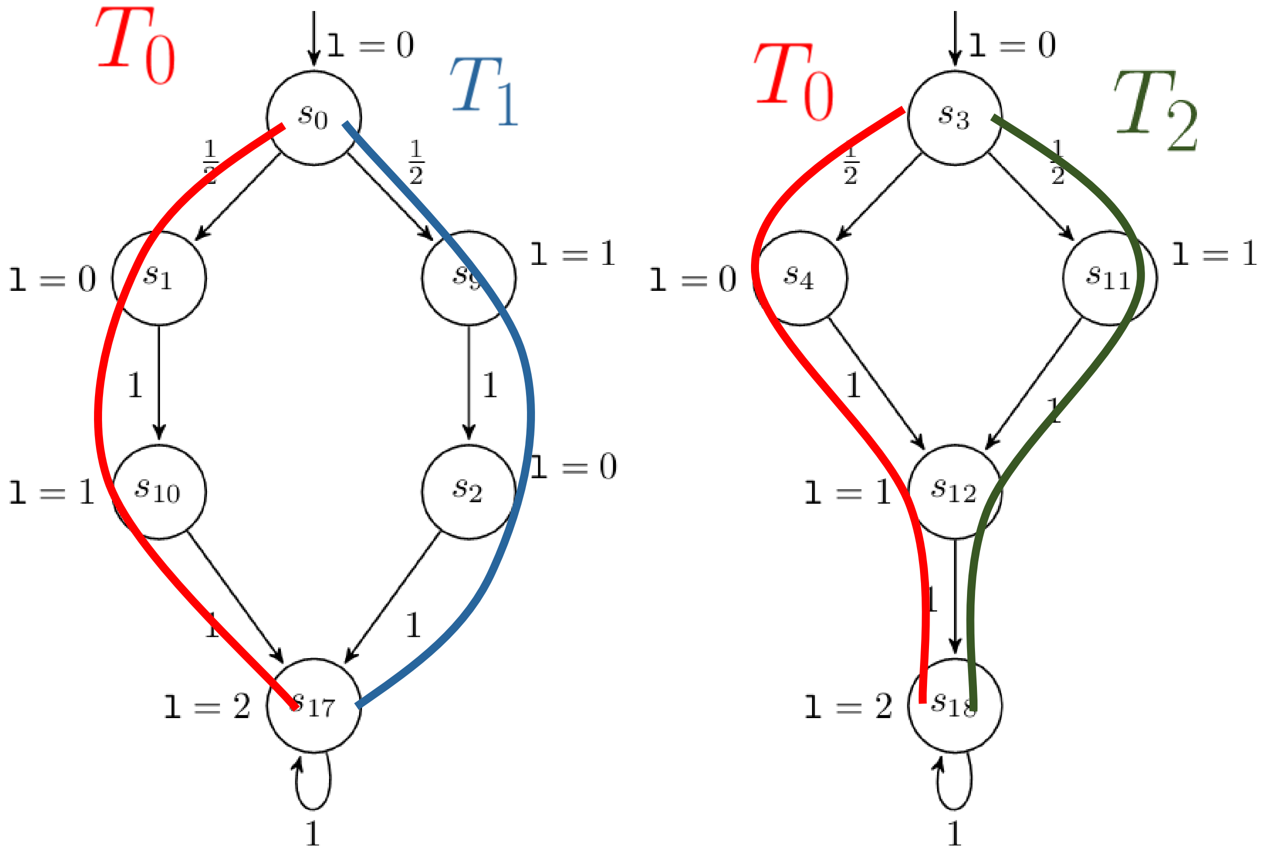


The proposed approach

$T_w = T_0$

$T_w \overset{?}{\triangleq}_p T_1$

$T_w \overset{?}{\triangleq}_p T_2$



4 The proposed approach

Time complexity of Algorithm 1

$$O(2^n)$$

The proposed approach

Verifying OD_2

$$OD_2 : \forall s_0, s_0' \in Init(\mathcal{M}_\delta^{\mathsf{P}}).\ \ Traces_{sf}(s_0) = Traces_{sf}(s_0').$$

The proposed approach

Algorithm 2 Verifying OD_2

Input: finite MC $\mathcal{M}_\delta^{\mathsf{P}}$

Output: *true* if the program satisfies OD_2; otherwise, *false*

```
Let π be an empty list of states for storing a path;
for s_0 in Init(M_δ^P) do
    // Consider an empty set of stutter-free traces for each initial state
    Let allTraces[s_0] be an empty set;
    explorePathsOD2(s_0, π, allTraces);
for each pair of initial states (s_0, s'_0) do
    if allTraces[s_0] != allTraces[s'_0] then
        return false;
return true;
```

The proposed approach

function explorePathsOD2(s, π, $allTraces$)

  π.add(s); *// add state s to the current path from the initial state*

  if s is a final state **then** *// found a path stored in π*

    $T_{|L} = trace_{|L}(\pi)$;

    Remove stutter data from $T_{|L}$, yielding stutter-free $T_{|L}^{sf}$;

    $s_0 = \pi[0]$; *// initial state of π*

    $allTraces[s_0].add(T_{|L}^{sf})$;

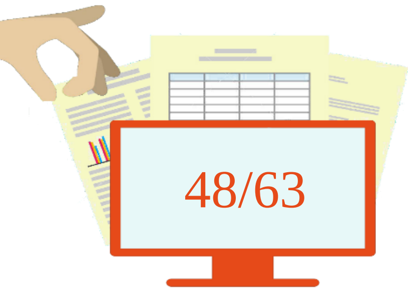
  else

    for s' **in** $Post(s)$ **do**

      explorePathsOD2(s', π, $allTraces$);

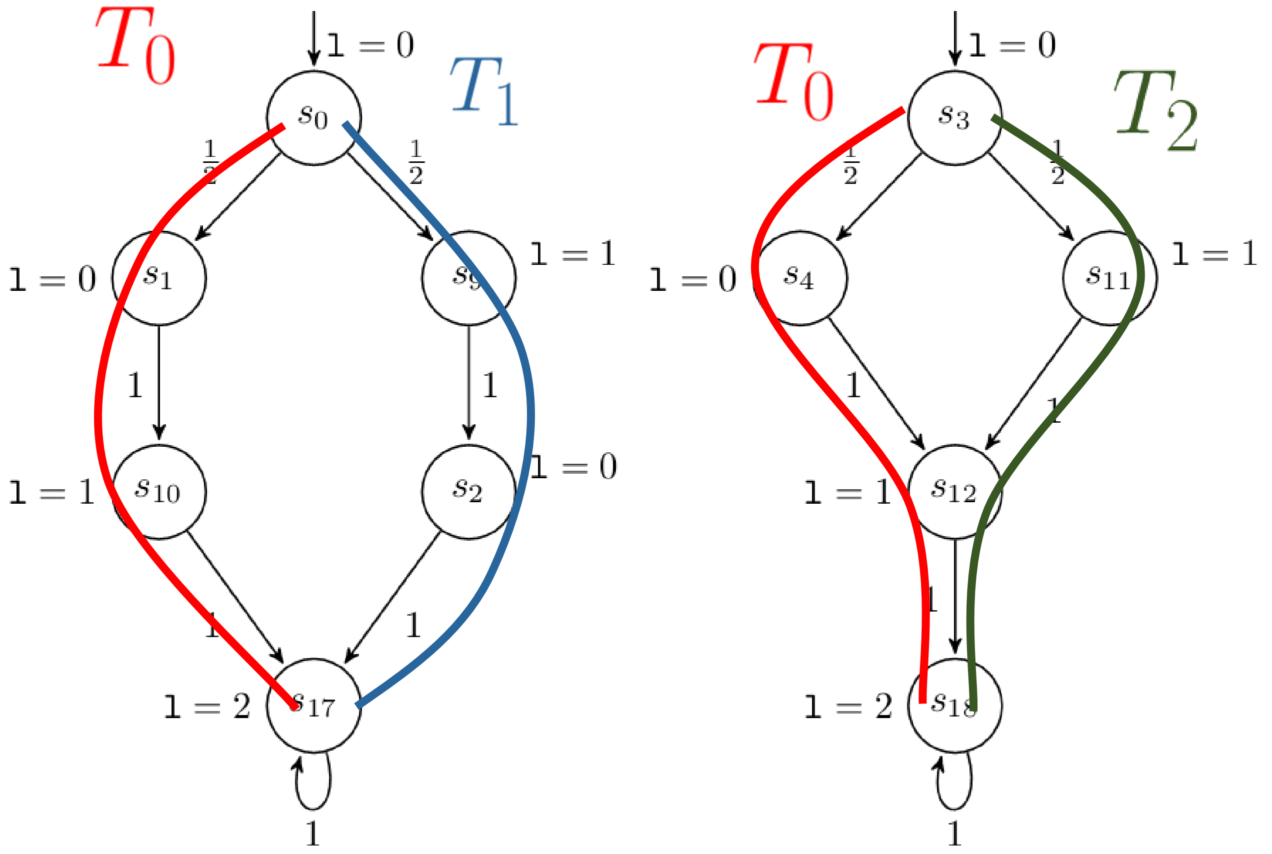
  π.pop(); *// done exploring from s, so remove it from π*

  return ;



The proposed approach

$$\{T_0^{sf}, T_1^{sf}\} \overset{?}{=} \{T_0^{sf}, T_2^{sf}\}$$



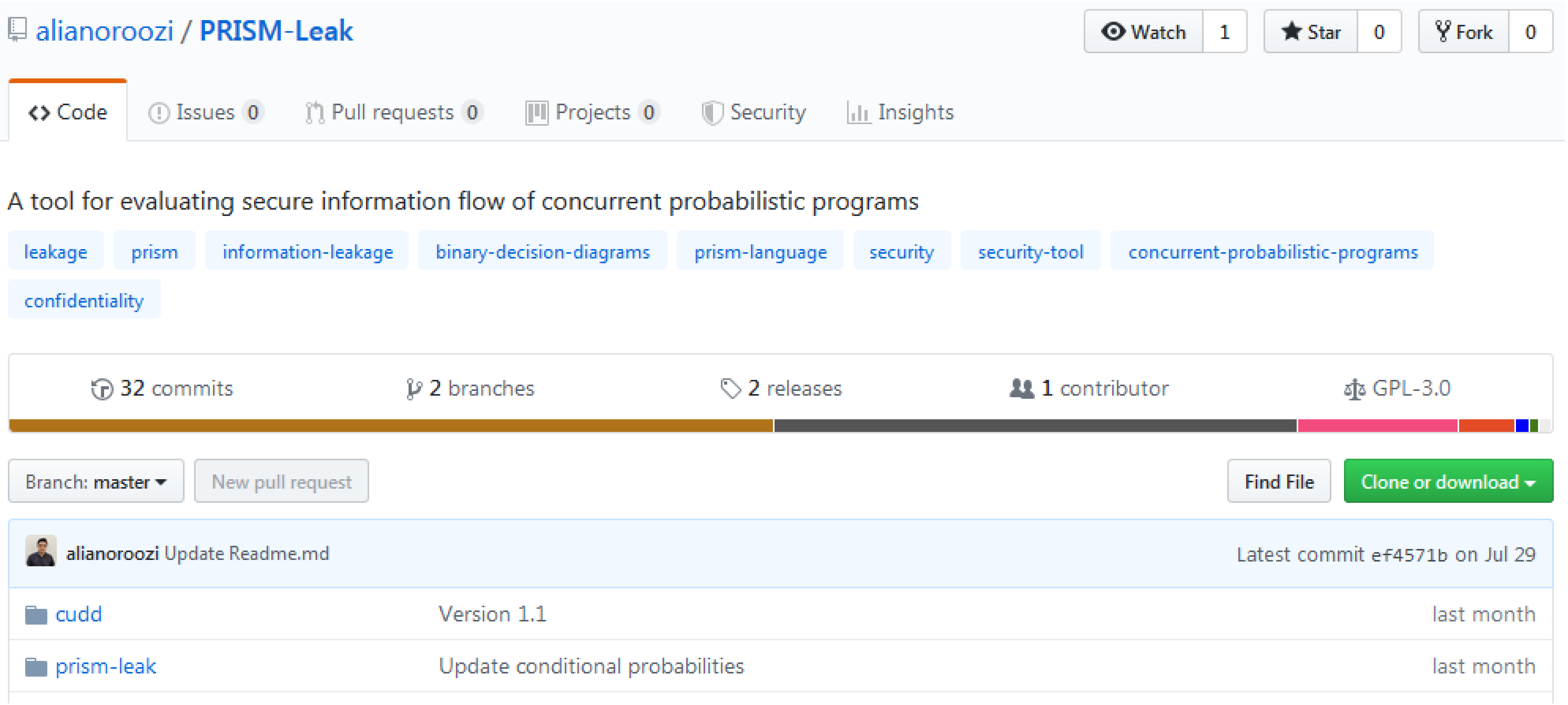
The proposed approach

Time complexity of Algorithm 2

$$O(2^n)$$

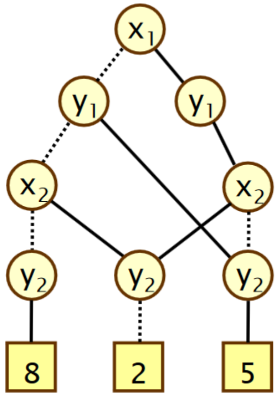
Contents

Experimental evaluation



Experimental evaluation

MTBDDs

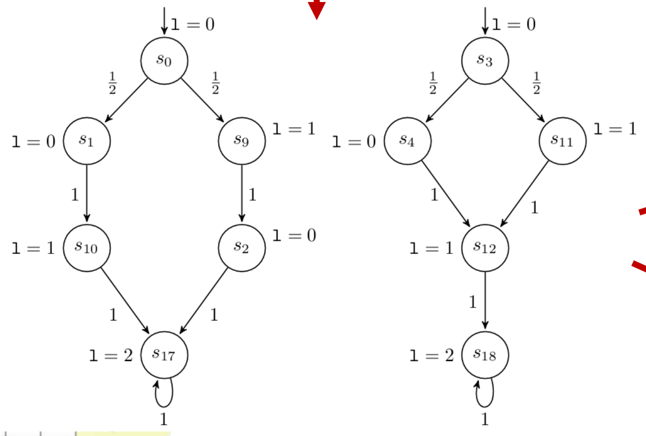



```
module M1

    x : [0..2] init 0;

    [] x=0 -> 0.8:(x'=0) + 0.2:(x'=1);
    [] x=1 & y!=2 -> (x'=2);
    [] x=2 -> 0.5:(x'=2) + 0.5:(x'=0);

endmodule
```



Qualitative package

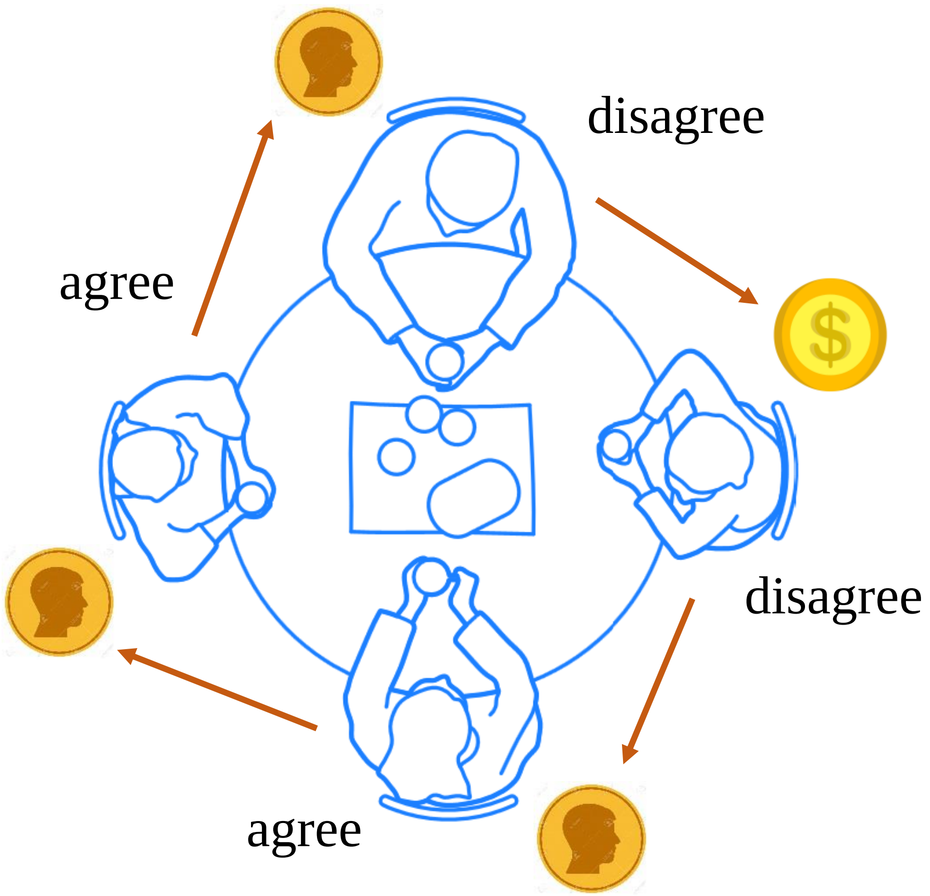
- OD-secure
- OD-insecure

Quantitative package

- Information leakage,
- Channel capacity,
- …

Experimental evaluation

Case study: the dining cryptographers protocol



Experimental evaluation

Runtime comparison for the 1st case of dining cryptographers

n	LeakWatch [8]	QUAIL [5]	HyLeak [4]	PRISM-Leak [24]	
				Quantitative method [22]	Observational determinism
7	2	1.8	30.5	0.6	**0.7**
8	3.7	3.1	39.7	0.8	**1.2**
9	7.5	6.3	55	1.3	**1.9**
10	15	12.6	72.2	2.9	**3.9**
11	32.2	26.5	97	7.3	**9.6**
12	72.4	62.1	135.4	18.7	**25.2**
13	150.7	151.6	249.3	49.9	**66.7**
14	Timeout	Timeout	Timeout	145.7	**192.4**

Experimental evaluation

Runtime comparison for the 2nd case of dining cryptographers

n	LeakWatch [8]	QUAIL [5]	HyLeak [4]	PRISM-Leak [24]	
				Quantitative method [22]	Observational determinism
7	3.1	2.4	30.8	0.6	**0.6**
8	6	4.5	41.7	1	**0.9**
9	12.3	9.7	57	1.5	**1.4**
10	28.2	17.5	75.3	3.5	**3.3**
11	60.5	35	99.3	7.7	**7.4**
12	122.1	78.5	144	20.4	**20.5**
13	Timeout	156.2	277.1	60.5	**58.8**
14	Timeout	Timeout	Timeout	215	**211.8**

Contents

Summary

A qualitative approach

Formal ✓

Fully-automatic ✓

Scalable ✓

High precision ✓

Future work

1. Symbolic model checking for verifying OD
2. OD checking of non-terminating programs
3. Estimating leakage by statistical methods

References

4. Biondi, F., Kawamoto, Y., Legay, A., Traonouez, L.-M.: HyLeak: hybrid analysis tool for information leakage. In: D'Souza, D., Narayan Kumar, K. (eds.) ATVA 2017. LNCS, vol. 10482, pp. 156–163. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68167-2_11
5. Biondi, F., Legay, A., Quilbeuf, J.: Comparative analysis of leakage tools on scalable case studies. In: Fischer, B., Geldenhuys, J. (eds.) SPIN 2015. LNCS, vol. 9232, pp. 263–281. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23404-5_17
6. Bischof, S., Breitner, J., Graf, J., Hecker, M., Mohr, M., Snelting, G.: Low-deterministic security for low-nondeterministic programs. J. Comput. Secur. **3**, 335–366 (2018)
7. Chaum, D.: The dining cryptographers problem: unconditional sender and recipient untraceability. J. Cryptol. **1**(1), 65–75 (1988)

References

8. Chothia, T., Kawamoto, Y., Novakovic, C.: LeakWatch: estimating information leakage from Java programs. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014. LNCS, vol. 8713, pp. 219–236. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11212-1_13
9. Dabaghchian, M., Abdollahi Azgomi, M.: Model checking the observational determinism security property using promela and spin. Form. Asp. Comput. **27**(5–6), 789–804 (2015)
10. Giffhorn, D., Snelting, G.: A new algorithm for low-deterministic security. Int. J. Inf. Secur. **14**(3), 263–287 (2015)
11. Graf, J., Hecker, M., Mohr, M., Snelting, G.: Tool demonstration: JOANA. In: Piessens, F., Viganò, L. (eds.) POST 2016. LNCS, vol. 9635, pp. 89–93. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49635-0_5
12. Huisman, M., Blondeel, H.-C.: Model-checking secure information flow for multi-threaded programs. In: Mödersheim, S., Palamidessi, C. (eds.) TOSCA 2011. LNCS, vol. 6993, pp. 148–165. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-27375-9_9

References

14. Huisman, M., Worah, P., Sunesen, K.: A temporal logic characterisation of observational determinism. In: Proceedings of the 19th IEEE Workshop on Computer Security Foundations, CSFW 2006. IEEE Computer Society (2006)
15. Karimpour, J., Isazadeh, A., Noroozi, A.A.: Verifying observational determinism. In: Federrath, H., Gollmann, D. (eds.) 30th IFIP International Information Security Conference (SEC). ICT Systems Security and Privacy Protection, Hamburg, Germany, Part 1: Privacy, vol. AICT-455, pp. 82–93, May 2015

20. Ngo, T.M., Stoelinga, M., Huisman, M.: Effective verification of confidentiality for multi-threaded programs. J. Comput. Secur. **22**(2), 269–300 (2014)

32. Zdancewic, S., Myers, A.C.: Observational determinism for concurrent program security. In: 2003 Proceedings of the 16th IEEE Computer Security Foundations Workshop, pp. 29–43, June 2003. https://doi.org/10.1109/CSFW.2003.1212703

Thanks for you attention.