Regular Model Checking Revisited

Anthony W. Lin
(TU Kaiserslautern and Max-Planck Institute for Software Systems, Germany)
[Joint with Philipp Rümmer]

SynCoP'22, Munich

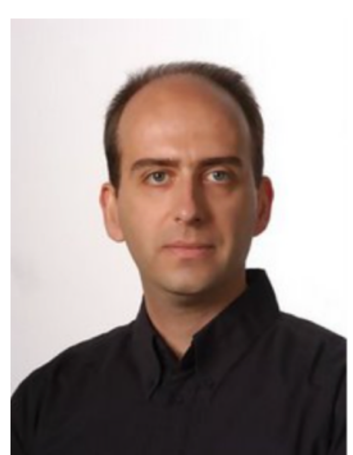
Acknowledgment











Outline of the talk

- RMC background and some history
- Reformulation of RMC in terms regular synthesis problem
- Brief overview of automata learning for regular synthesis

Parameterized Systems

Definition: An infinite family of finite-state systems

$\mathcal{F}$:= {Distributed Protocol with n finite processes : $n \in \mathbb{N}$}

Plethora of examples from distributed computing, e.g., Dining Philosopher protocol, Bakery Protocol, etc.

Undecidability for simple safety properties (Apt & Kozen'86)

Lots of work on parameterized systems dating back to 1990s by Emerson, Pnueli, and others

Regular Model Checking

A symbolic framework for verifying parameterized systems

Symbolic model checking with rich assertional langua
Y Kesten, O Maler, M Marcus, A Pnueli… - … Conference on Computer
… In this section we demonstrate the use of the class of regular langua
languages. As a running example, we … Going back to the use of FSAs
language, we observe that if A is an automaton characterizing a set of s
Save Cite Cited by 208 Related articles All 20 versions

Regular model checking
A Bouajjani, B Jonsson, M Nilsson, T Touili - International Conference on …, 2000 - Spring
… We present **regular model checking**, a framework for algorithmic verification of infinite-
systems with, … States are represented by strings over a finite alphabet and the transition
relation by a **regular** length-… We introduce the program **model** used in **regular model ch**
Save Cite Cited by 369 Related articles All 24 versions

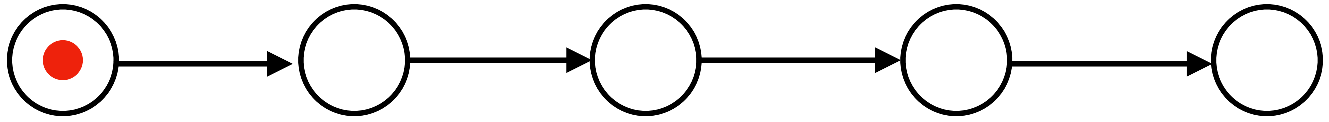
Verifying systems with infinite but regular state spaces
P Wolper, B Boigelot - … Conference on Computer Aided **Verification**, 1998 - Springer
… to consider a larger class of **systems**, **but** to be satisfied with a … of closed **systems** wh
infinite state space originates from the … The focus on closed **systems** is typical of many
Save Cite Cited by 193 Related articles All 14 versions Web of Science: 65

Crux:

1. Model configuration as a string
2. Represent an infinite set of strings using **regular languages**
3. Model transition relation by a **length-preserving transducer** R

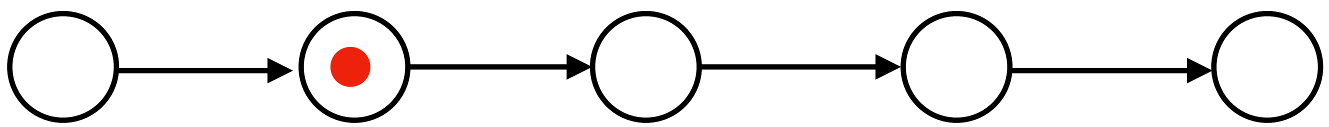
other notions of transducers (e.g. over trees, ω-words) are possible, but we restrict to this for simplicity

Token Passing Protocol



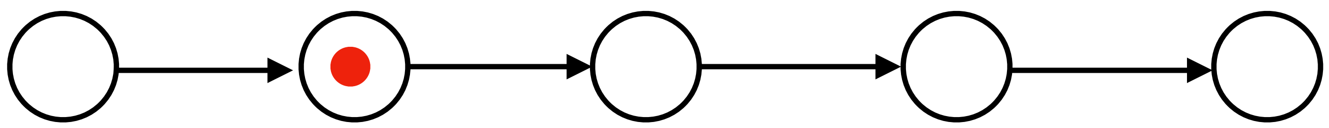
$t = 1$

Token Passing Protocol



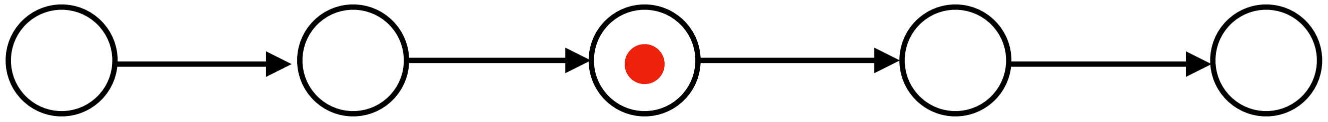
$t = 2$

Token Passing Protocol



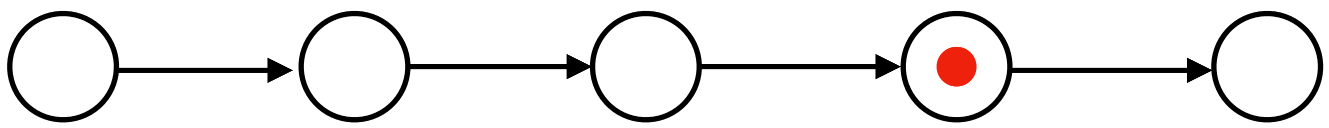
$t = 3$

Token Passing Protocol



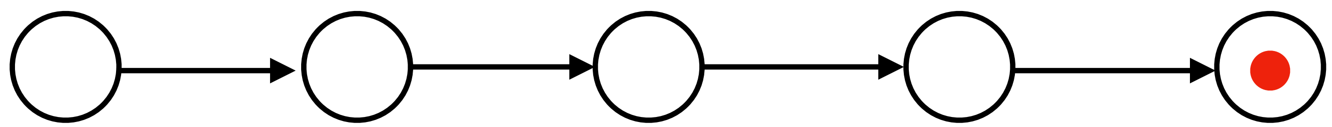
$t = 4$

Token Passing Protocol



$t = 5$

Token Passing Protocol

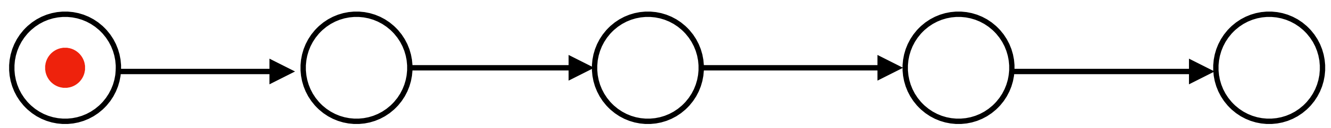


$t = 6$

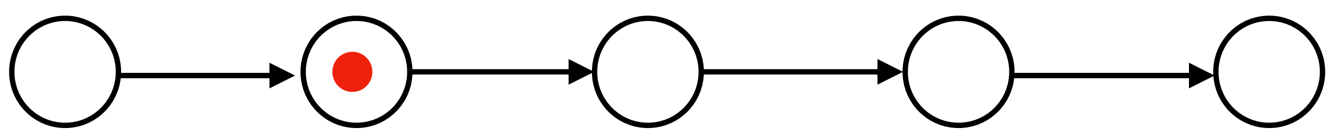
Safety: prove that the token never disappears

Liveness: prove that the token always reaches the last process (under some fairness assumption)

Model in RMC



represented as 10000



represented as 01000

The set Init of <u>initial configurations</u> is a regular language:

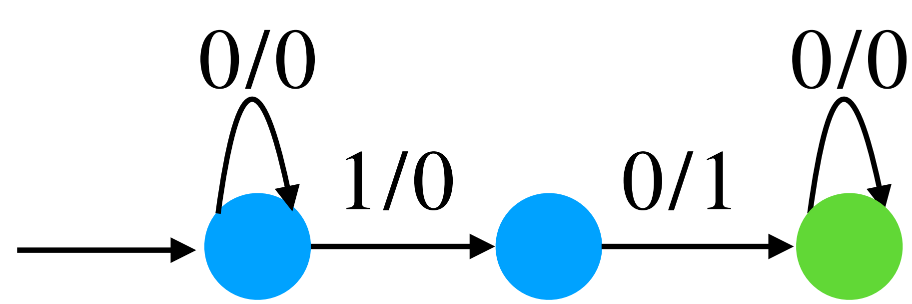
$$\mathit{Init} = 10^*$$

The set Bad of <u>bad configurations</u> is a regular language:

$$\mathit{Init} = 0^*$$

The <u>transition relation</u> (over strings) can be represented as transducer:

$R :=$

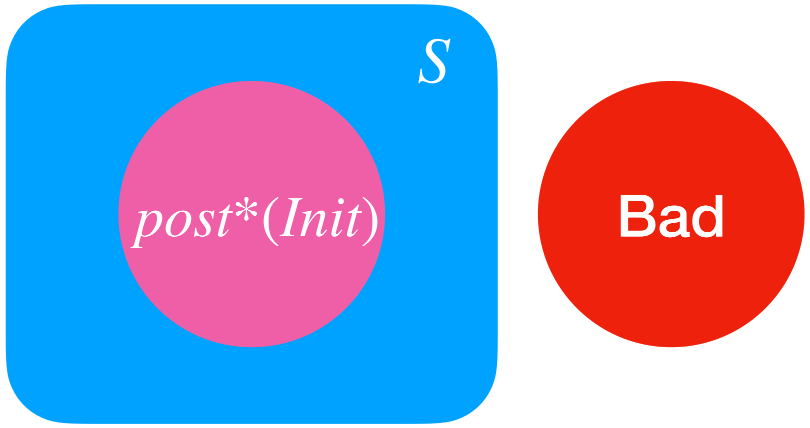


Computing Closures

Reachability Set

Compute a regular language representing $post_R^*(Init)$ or its overapproximation

Useful for safety: $S \supseteq post_R^*(Init) \wedge S \cap Bad = \emptyset \longrightarrow$ safe



For our simple token-passing example:

$Init = 10^*$

$Bad = 0^*$

So: $post^*(Init) = 0^*10^*$

We could also take $S = (0+1)^*1(0+1)^*$

Problem and Solutions

Problem with computing closures:

1. Non-regularity
2. Non-termination
3. Regular but extremely large states

General solutions:

1. Acceleration (Abdulla, Jonsson, Nilsson, Orso; Boigelot et al.)
2. Widening (Bouajjani and Touili; Boigelot, Legay, and Wolper; Yu, Alkhalaf, Bultan, and Ibarra)
3. Abstraction (Bouajjani, Habermehl, Vojnar)
4. **Automata learning** → **inspired our new framework**
   (pre 2010: Vardhan et al.; Habermehl and Vojnar)
   (post 2010: Neider and Jansen; Chen, Hong, Lengal, L., Majumdar, Markgraf, Neider, Rümmer, Stan)

RMC Beyond Safety

So far, **only automata learning** enjoys some success:

Liveness of Randomized Distributed Protocols (L. & Rümmer, Lengal, L., Majumdar, and Rümmer)

Solving Safety Games (Neider and Topcu; Markgraf, Hong, L., Najib and Neider)

Probabilistic Bisimulation and Anonymity Protocols (Hong, L., Majumdar, Rümmer)

Knowledge Reasoning in Multi-Agent Systems (Stan and L.)

Symmetry Detection in RMC (L., Nguyen, Rümmer, and Sun)

RMC as a Regular Synthesis Problem

Deductive Verification

Commonly used in program verification (among others)

```
x = 0
while true:
    x = x + 2
    if x % 2 == 1:
        print "error"
```

Prove "error" is never printed

Safety as "Invariant checking" in some decidable theory:

$\text{Init}(x) := x = 0 \qquad \text{Bad}(x) := x \equiv_2 1$

Want to synthesize formula $\text{Inv}(x)$ s.t.

$$\forall x(\text{Init}(x) \to \text{Inv}(x))$$

$$\forall x(\text{Inv}(x) \to \neg\text{Bad}(x))$$

$$\forall x(\text{Inv}(x) \to \text{Inv}(x+2))$$

Satisfied by $\text{Inv}(x) := x \equiv_2 0$

Proposition: Given Presburger Inv, invariant checking is decidable

Decidable Theory for RMC

Which decidable theory of regular languages and transducers is suitable for deductive verification in RMC?

Our answer: universal automatic structure (Blumensath&Grädel'00)

$$\mathfrak{S}_u = \langle \Sigma^* : \preceq, eql, \{L\}_{L \in REG} \rangle$$

Domain is the set of all words over Σ

$\preceq$ is the prefix-of relation: $v \preceq w$ iff v is a prefix of w

eql is the equal-length relation: $eql(v, w)$ iff $|v| = |w|$

L is any regular language: $L(x)$ iff $x \in L$

Theorem (BG'00): FO theory over $\mathfrak{S}_u$ is decidable

Regular relations

r-ary relation over Σ^* definable by a synchronous automaton A

A synchronous automaton is simply an automaton over the alphabet $(\Sigma_\perp)^r$

where $\Sigma_\perp := \Sigma \cup \{ \perp \}$

How A defines a relation?

given a tuple $\bar{v} = (w_1, \ldots, w_r)$, write it down as a matrix $M_{\bar{v}}$ with each w_i being the ith row (pad shorter string with $\perp$)

Example:

$\bar{v} := (aaa, cb, a)$

$$M_{\bar{v}} := \begin{pmatrix} a & a & a \\ b & c & \perp \\ a & \perp & \perp \end{pmatrix}$$

Regular relations

r-ary relation over Σ^* definable by a synchronous automaton A

A synchronous automaton is simply an automaton over the alphabet $(\Sigma_\perp)^r$

where $\Sigma_\perp := \Sigma \cup \{ \perp \}$

How A defines a relation?

Run A on $M_{\bar{v}}$ column-by-column

$\bar{v} := (aaa, cb, a)$

$$M_{\bar{v}} := \begin{pmatrix} a & a & a \\ b & c & \perp \\ a & \perp & \perp \end{pmatrix}$$

A

Regular relations

r-ary relation over Σ^* definable by a synchronous automaton A

A synchronous automaton is simply an automaton over the alphabet $(\Sigma_\perp)^r$

where $\Sigma_\perp := \Sigma \cup \{ \perp \}$

How A defines a relation?

Run A on $M_{\bar{v}}$ column-by-column

$$\bar{v} := (aaa, cb, a) \qquad M_{\bar{v}} := \begin{pmatrix} a & a & a \\ b & c & \perp \\ a & \perp & \perp \end{pmatrix}$$

A

Regular relations

r-ary relation over Σ^* definable by a synchronous automaton A

A synchronous automaton is simply an automaton over the alphabet $(\Sigma_\perp)^r$

where $\Sigma_\perp := \Sigma \cup \{ \perp \}$

How A defines a relation?

Run A on $M_{\bar{v}}$ column-by-column

$\bar{v} := (aaa, cb, a)$ $\qquad$ $M_{\bar{v}} := \begin{pmatrix} a & a & \boxed{a} \\ b & c & \perp \\ a & \perp & \perp \end{pmatrix}$

A

Define: $Rel(A) := \{\bar{v} : M_{\bar{v}} \in L(A)\}$

Regular Relations in $\mathfrak{S}_u$

A relation $R \subseteq (\Sigma^*)^r$ is **definable** in $\mathfrak{S}_u$ iff there is an FO formula $\varphi(x_1, \ldots, x_r)$ s.t.

$$R = \{(w_1, \ldots, w_r) : \mathfrak{S}_u \models \varphi(w_1, \ldots, w_r)\}$$

Theorem (BG'00): Regular relations coincide precisely with relations definable in $\mathfrak{S}_u$

RMC as a Regular Synthesis Problem

Existential Second-Order (ESO) formulas over $\mathfrak{S}_u$

$$\Phi := \exists R_1, \ldots, R_n \varphi$$

where

(1) R_i is a second-order variable of arity r_i

(2) φ is an FO formula over $\mathfrak{S}_u \cup \{R_1, \ldots, R_n\}$

<u>ESO model checking over $\mathfrak{S}_u$</u>:

given an ESO formula Φ over $\mathfrak{S}_u$, decide if $\mathfrak{S}_u \models \Phi$

<u>Regular Synthesis for RMC</u>:

given an ESO formula Φ over $\mathfrak{S}_u$, decide if there exist r_i-ary regular relations R_i such that $\mathfrak{S}_u \models \varphi$

Empirically: Regular proofs suffice in practice

Safety as Regular Synthesis

Inputs: (1) regular languages $Init, Bad$,
(2) length-preserving regular relation R

Verification Condition:

$$\exists Inv(Init \subseteq Inv \wedge Inv \cap Bad = \emptyset \wedge post_R(Inv) \subseteq Inv)$$

$\forall x(\mathrm{Init}(x) \rightarrow \mathrm{Inv}(x))$

$\forall x(\mathrm{Inv}(x) \rightarrow \neg \mathrm{Bad}(x))$

$\forall x, y(\mathrm{Inv}(x) \wedge R(x, y) \rightarrow \mathrm{Inv}(y))$

Our simple token-passing example:

$Init = 10^*$

$Bad = 0^*$

Can take $Inv_1 = 0^*10^*$

or $Inv_2 = (0+1)^*1(0+1)^*$

Termination as Regular Synthesis

Termination: no infinite runs exist

Inputs: (1) regular languages $Init$,

(2) length-preserving regular relation R

Verification Condition:

$\exists Inv \subseteq \Sigma^*$, $Rank \subseteq \Sigma^* \times \Sigma^*$:

(1) $Init \subseteq Inv$,

(2) Inv is inductive

(3) $Rank$ covers reachable transitions: $R \cap (Inv \times Inv) \subseteq Rank$

(4) $Rank$ is transitive and irreflexive

Verification Condition:
$\exists Inv \subseteq \Sigma^*, Rank \subseteq \Sigma^* \times \Sigma^*$:
(1) $Init \subseteq Inv$,
(2) Inv is inductive
(3) $Rank$ covers reachable transitions: $R \cap Inv \times Inv \subseteq Rank$
(4) $Rank$ is transitive and irreflexive

Example

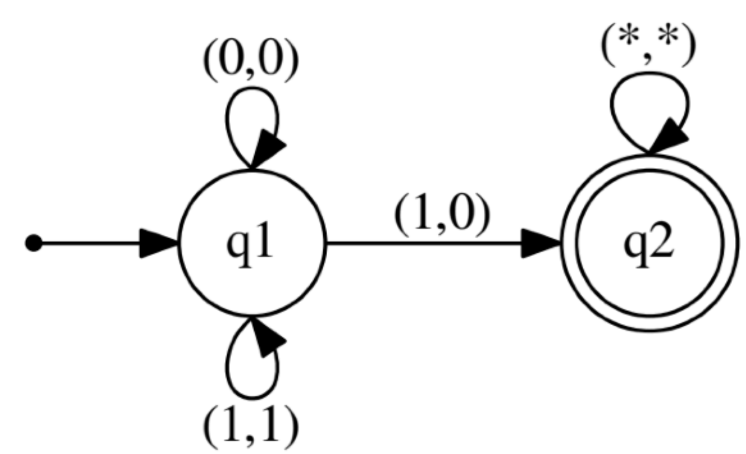
Consider a length-preserving regular relation over $\Sigma = \{0,1\}$ that nondeterministically rewrites 10 to 01

$$1010 \to 0110 \to 0101 \to 0011$$

$$Init = 0\Sigma^*1$$

$$R := ((0,0) + (1,1))^*(1,0)(0,1)((0,0) + (1,1))^*$$

$Inv = \Sigma^*$

$Rank =$ 

Lexicographic order

Reachability Games as Regular Synthesis

Inputs: (1) regular languages $Init$, F

(2) length-preserving regular relations R_1, R_2 with

$post^*_{R_i}(\Sigma^*) \cap pre^*_{R_i}(\Sigma^*) = \emptyset$ [i.e. strictly alternating.]

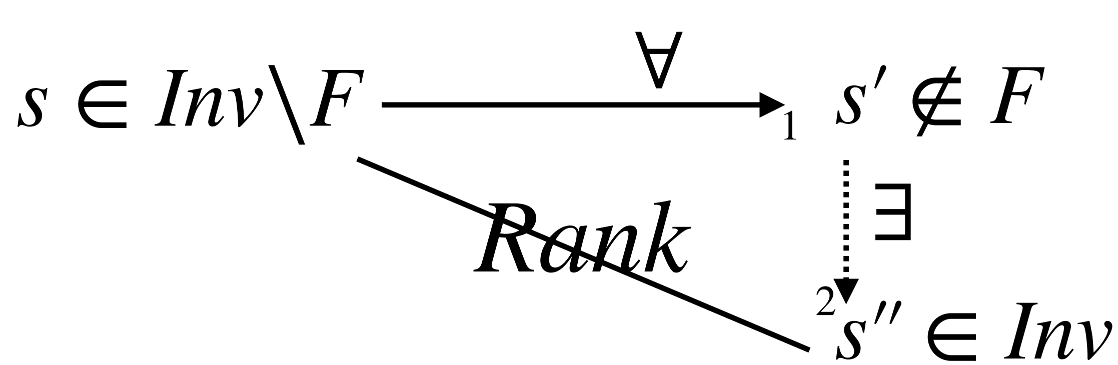
Goal: Player 2 (resp. 1) tries to reach (resp. avoid) F from $Init$

Verification Condition:

$\exists Inv \subseteq \Sigma^*$, $Rank \subseteq \Sigma^* \times \Sigma^*$:

(1) $Init \subseteq Inv$, (2) $Rank$ is transitive and irreflexive

(3) Player 0 can force the game to progress according to $Rank$



Example: Take-Away Game

There are n coins on the table

At each turn, a player can take 1,2, or 3 coins

Player who is to move when no coins are left loses

Initially, Player 1 moves

The game strictly alternates

Example: Take-Away Game

Regular modelling:

$$Dom = (p_1 + p_2)1^*0^*$$

e.g. $p_1 111000$ represents Player 1's turn on



Say we want to prove that, starting with $4k$ coins, Player 2 has a winning strategy

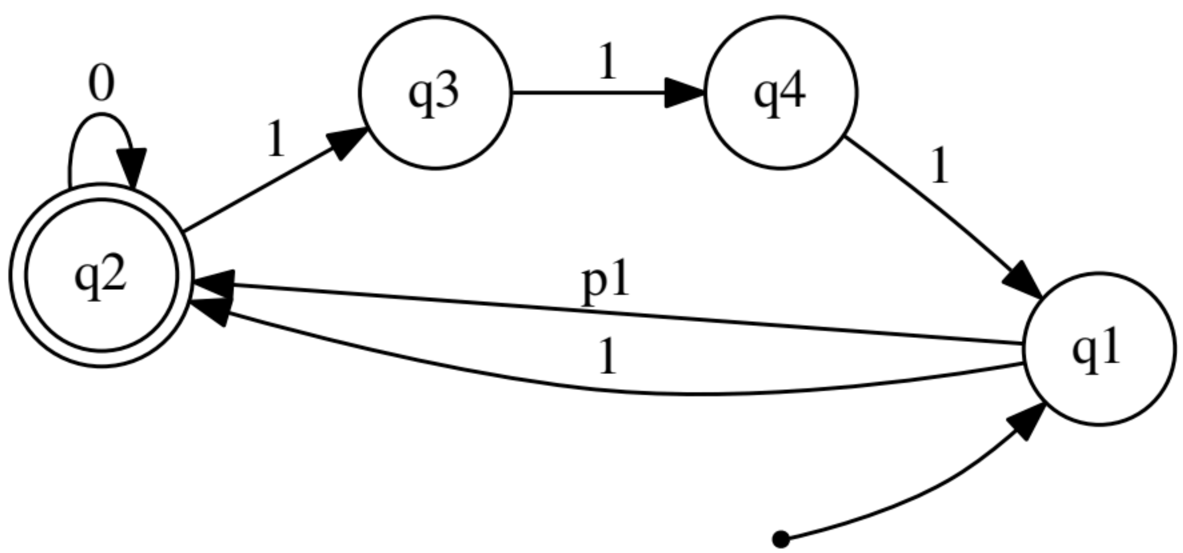
$$Init = p_1(1111)^*0^*$$

$$F = p_1 0^*$$

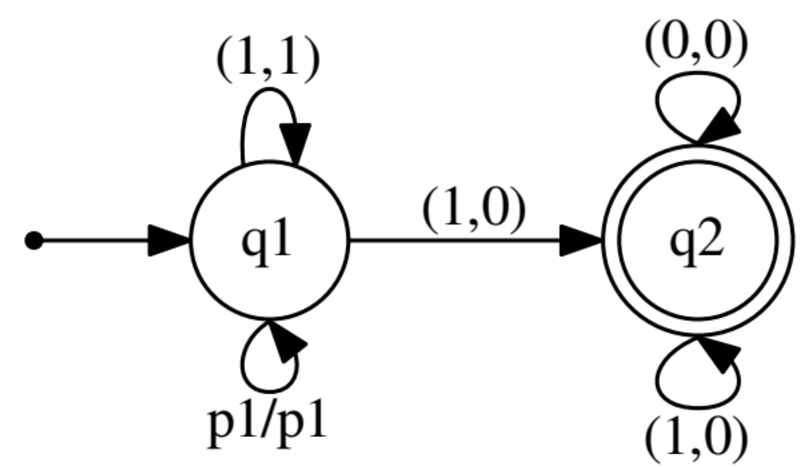
Transitions:

$$R_1 = (p_1, p_2)(1,1)^*((1,0) + (11,00) + (111,000))(0,0)^*$$
$$R_2 = (p_2, p_1)(1,1)^*((1,0) + (11,00) + (111,000))(0,0)^*$$

Regular Proofs



Inv



Rank

Liveness of Randomized Parameterized Systems

Similar regular encoding as in 2-player reachability games is possible

Lots of examples:

1. Lehmann-Rabin dining philosopher protocol
2. Israeli-Jalfon self-stabilizing protocol
3. Herman self-stabilizing protocol
4. …

Regular synthesis algorithms via automata learning: Brief Overview

The Gist of Automata Learning (a la Angluin)

Learner tries to learn L_T from Teacher

Typical queries:

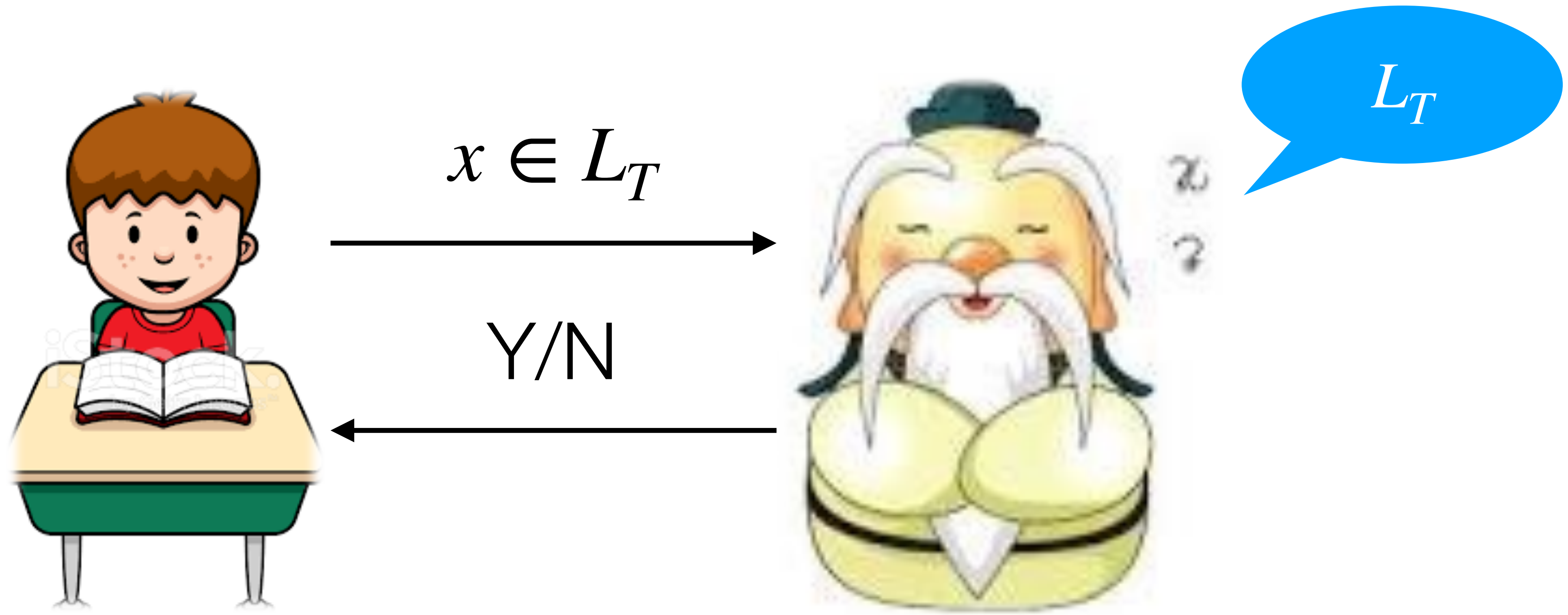
(ME) Membership query: $x \in L_T$

(EQ) Equivalence query: $L = L_T$

Two common variations in RMC:

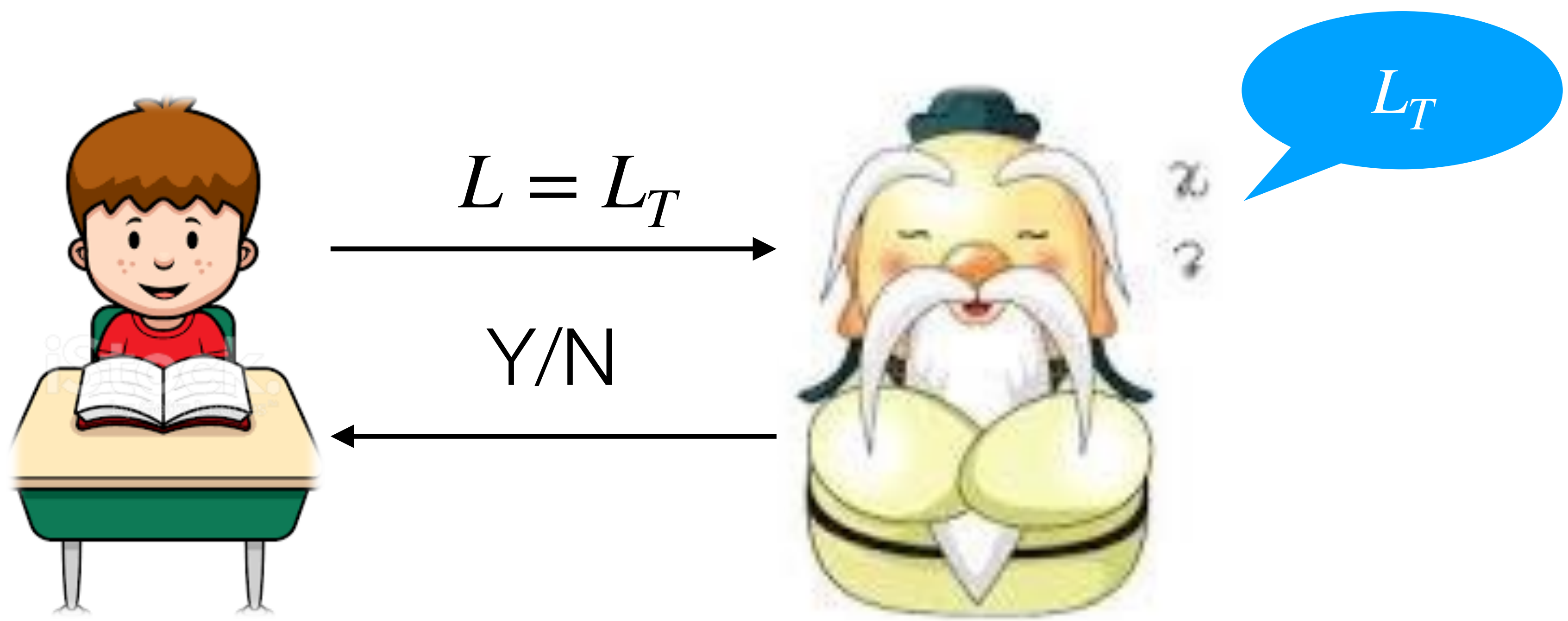
1. EQ only
2. ME+EQ

Membership



Learner tries to learn L_T from Teacher

Equivalence



Learner tries to learn L_T from Teacher

Counterexample: $w \in (L_T \backslash L) \cup (L \backslash L_T)$

Theorem (Angluin): there is a polynomial-time algorithm for inferring an unknown DFA from a teacher with ME+EQ

Problem with Membership

In general, difficult to implement a teacher for membership

Example: to learn a $post_R^*(Init)$, checking whether $w \in post_R^*(Init)$ is typically undecidable

In restricted cases, solutions are available (we will see later)

In general case, it seems a good idea to *dispense with memberships*

Automata Learning with Equivalence (and SAT-solver)

(Heule and Verwer'10)

Learner: keeps a Boolean formula φ representing a set S_φ of DFAs with n states (n is incremented as needed)

Main loop:

1. Learner guesses $A_\varphi \in S_\varphi$ [using SAT-solver]
2. If $L(A_\varphi) \neq L_T$ with cex $w \in \Sigma^*$, incorporate w into φ as a "blocking clause" and goto (1)

Non-Uniqueness of Target Automata

Safety: $\exists Inv(Init \underset{(1)}{\subseteq} Inv \wedge Inv \cap Bad \underset{(2)}{=} \emptyset \wedge post_R(Inv) \underset{(3)}{\subseteq} Inv)$

Inv is not unique in general!

Solution: Teacher *returns a boolean formula as a blocking clause*

For L_T violating (1)-(2), teacher can reply a +/- cex

For L_T violating (3), teacher replies an implication cex

$v \in L_T \rightarrow w \in L_T$

Sometimes Membership can be implemented

Membership query: $x \in L_T$

Using finite-state model checker to check if

$$\{y \in \text{Init} : |y| = |x|\} \to^* x$$

Possible because of length-preserving assumption

Empirical observation:
when learning with ME+EQ can be applied, it's faster than SAT-based learning

Experimental Results

The safety property								Learning			SAT			T(O)RMC			ARMC
Name	#L	S_{init}	T_{init}	S_{tran}	T_{tran}	S_{bad}	T_{bad}	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}	Time	S_{inv}	T_{inv}	Time
Bakery	3	3	3	5	19	3	9	0.0s	6	18	0.5s	2	5	0.0s	6	11	0.0s
Burns	12	3	3	10	125	3	36	0.2s	8	96	1.1s	2	10	0.1s	7	38	0.6s
Szymanski	11	9	9	123	469	13	40	0.6s	48	528	t.o.	–	–	0.9s	51	102	t.o.
German	581	3	3	17	9.5k	4	2112	4.8s	14	8134	t.o.	–	–	t.o.	–	–	2.3s
Dijkstra, linear	42	1	1	13	827	3	126	0.1s	9	378	1.7s	2	24	6.1s	8	83	t.o.
Dijkstra, ring	12	3	3	13	199	3	36	1.4s	22	264	0.9s	2	14	t.o.	–	–	t.o.
Dining Crypt.	14	10	30	17	70	12	70	0.1s	32	448	t.o.	–	–	0.2s	37	164	1.3s
Coffee Can	6	8	18	13	34	5	8	0.0s	3	18	0.2s	2	7	0.1s	6	13	0.0s
Herman, linear	2	2	4	4	10	1	1	0.0s	2	4	0.2s	2	4	0.0s	2	4	0.0s
Herman, ring	2	2	4	9	22	1	1	0.0s	2	4	0.4s	2	4	0.0s	2	4	0.0s
Israeli-Jalfon	2	3	6	24	62	1	1	0.0s	4	8	0.1s	2	4	0.0s	4	8	0.0s
Lehmann-Rabin	6	4	4	14	96	3	13	0.1s	8	48	0.5s	2	11	0.8s	19	105	0.0s
LR Dining Phil.	4	4	4	3	10	3	4	0.0s	4	16	0.2s	2	6	0.1s	7	18	0.0s
Mux Array	6	3	3	4	31	3	18	0.0s	5	30	0.4s	2	7	0.2s	4	14	0.1s
Res. Allocator	3	3	3	7	25	4	11	0.0s	5	15	0.0s	3	7	0.0s	4	9	0.0s
Kanban	3	25	48	98	250	37	68	t.o.	–	–	t.o.	–	–	t.o.	–	–	t.o.
Water Jugs	11	5	6	23	132	5	12	0.1s	24	264	t.o.	–	–	t.o.	–	–	t.o.

Conclusion

Summary

- RMC can be in general formulated as a regular synthesis problem
- Future work:
  (1) more general and faster synthesis algorithm for regular synthesis

  (2) Extension to non-length-preserving RMC and ω-RMC (proof rules are more complicated requiring Ramsey quantifiers)

ANNEX

Strict but Generous Teacher

(FMCAD'17)

Since there could be multiple Inv, we implement a teacher that is:

1. <u>strict</u>: provides hints consistent with minimal invariant $L_T = post^*(\text{Init})$
2. <u>generous</u>: accepts *any* invariant

How to answer membership query $x \in L_T$

$$x \in L_T \text{ IFF } \{y \in \text{Init} : |y| = |x|\} \to^* x$$

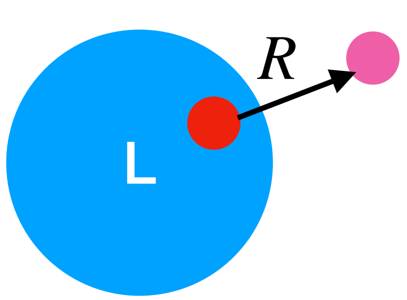
use finite-state model checker

How to answer equivalence query $L = L_T$

$$\text{Init} \subseteq L \wedge L \cap \text{Bad} = \varnothing \wedge R(L) \subseteq L$$

use automata algorithm

Counterexample for $R(L) \subseteq L$



● is NOT reachable (or ● $\in$ Bad) => remove ● from L

● is reachable ==> add ● to L