Active Learning : From Blackbox To Timed Connectors

Yi Li* and Meng Sun*
*DI, School of Mathematical Sciences, Peking University, Beijing, China
liyi_math@pku.edu.cn, summeng@math.pku.edu.cn

***Abstract*—Coordination is becoming more and more important, especially in concurrent programming and distributed systems. More and more tools are built to verify coordination models while most of them are facing the same problem: *lack of models*. In many cases, somehow we need to check a connector with nothing more than a binary file. In this paper, we present a learning-based framework to formally verify a Reo connector in case there is no source code. From some practical cases, we've shown the effciency of our approach. Both the algorithm and Reo itself have been ported to *Golang*, making it fully-prepared for parallel programming.**

***Index Terms*—active learning, coordination, formal method.**

I. Introduction

In the past few years, researchers have been focusing on this area, and come up with a series of impressive works. However, most of these works are based on models, instead of binaries. Then it comes a well-known problem: *how can we obtain these models?*

To solve this problem, many techniques in model constructing were proposed, for instance in [1], [4], [5].

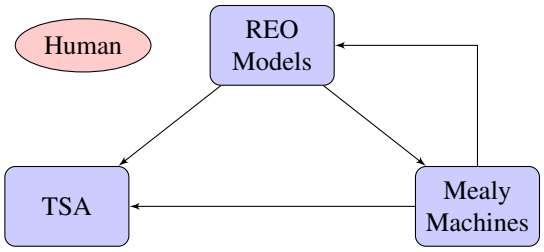

Fig. 1. Our Idea

In this paper, we presented an adapted active learning algorithm to extract timed Reo connectors from binaries with no source code needed.

II. Preliminaries

A. Reo Coordination Language

Reo Coordination Language [2]

B. Active Learning and Mealy Machines

III. Timed Connectors as Mealy Machines

A. External Behavior of Reo Connectors

So far as we can tell, most works [3], [5] on automata learning are not capable of infinite models. Considering the semantics defined in section II-A, it's apparent that every finite connector has a corresponding constraint automata, and the automata is also finite. Unfortunately, when checking the further definition of Reo connectors' input, we find that it's not the case.

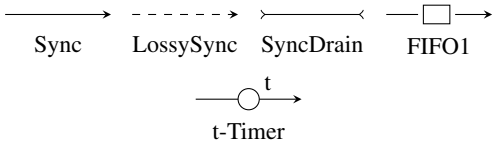

Fig. 2. Basic Reo Channels

Fig. 3. Infinite States in a Reo Model

While focusing on behavior of connectors, Reo doesn't give detail depiction on behavior of components. As shown in Fig.3, connectors are able to reject any datum if they are not ready. But what will happen outside the connector, if the datum is rejected? This question deserves careful consideration if we are taking an external view.

B. Time Domain

Time is involved in several extension of Reo like timed Reo, hybrid Reo, etc. Generally, these models are designed to handle real-time behavior where time is defined in $\mathbb{R}$. However, we also found a lot of works where authors use rational time to make things easier.

In this paper, we choose the rational number field $\mathbb{Q}$ as our time domain, which simplifies discretization of timed behaviors greatly.

As presented in section II-A, all real-time behavior in timed Reo comes with the *t-timer* channels, and the number of these channels are apparently finite. We use $t_i \in \mathbb{Q}$ to denote the delays of these timer channels, and now we can define a precision function $prec$.

$$prec(t_1, \cdots, t_n) = \max_T \{\forall t_i . \exists n_i \in \mathbb{N} . t_i = n_i \cdot T\}$$

It's easy to prove that such a T is always existing.

In real systems, the concept *time precision* is widely used with the name "clock-period". Most of the time, we know the clock-cycle of some hardware components, even without

any idea of its structure. With such precision T given, it's reasonable to assume that all t-timers are actually nT-timers. In following sections, we'll use n-timers instead.

Besides, we're going to add a single "T" action in mealy machines. It indicates that a transition will take a time unit to finish, and all outputs would come out after that.

C. Parameterized Mealy Machine

We present a model named *parameterized mealy machine* to represent timed connectors (hereinafter referred to as PMM). This model is supposed to behave as a middle representation. Connectors are defined as parameterized mealy machines, and composed via its production operator. Then original mealy-machine model will be taken as semantics of PMM model.

Definition 1 (Parameterized Mealy Machine): A *Parameterized Mealy Machine* is defined as a function $\mathcal{PM}(\Sigma) = \langle S(\Sigma), s_0, I, O, \delta(\Sigma), \lambda(\Sigma)\rangle$ that maps an alphabet to its corresponding Mealy Machine.

- Σ is a *finite* datum alphabet (hereinafter referred to as an alphabet)
- S is a function that maps an alphabet to a *finite* set of states. We use $S(\Sigma)$ to denote the state set.
- I is a finite set of source-ends.
- O is a finite set of sink-ends.
- s_0 is the initial state. It satisfies $\forall\Sigma, s_0 \in S(\Sigma)$
- δ maps a *finite* datum alphabet to an *output function*. We use $\delta(\Sigma) : S(\Sigma) \times Input(\Sigma, I) \to Output(\Sigma, O)$ to denote the output function.
- λ maps an alphabet to a *transition function*. We use $\lambda(\Sigma) : S(\Sigma) \times Input(\Sigma, I) \to S(\Sigma)$ to denote the transiton function.

In the definition above, *Input* and *Output* are used to generate input actions and output actions by the corresponding alphabets and ends.

How to say?

$$Input(\Sigma, I) = \{\} \cup \{T\}$$

where we use T to denote the time action.

$$Output(\Sigma, I) = \{\}$$

Now we can use Parameterized Mealy Machines to define a new semantics to Reo.

Example 1 (PMM Semantics of FIFO channel PM_{FIFO}): The semantics of FIFO channel with source end A and sink end B can be defined as follows

- $S(\Sigma) = \{q_0\} \cup \{q_d | d \in \Sigma\}$
- $I = \{A\}$
- $O = \{B\}$
- $s_0 = q_0$
- output function

$$\delta(\Sigma)(s,i) = \begin{cases} (B:\perp) & s = q_0 \wedge i = (A:\_, B:\frownie) \\ (B:d) & s = q_0 \wedge i = (A:d, B:\smilie) \\ \ominus & s = q_0 \wedge i = (A:\perp, B:\smilie) \\ (B:d) & s = q_d \wedge i = (A:\_, B:\smilie) \\ \ominus & s = q_d \wedge i = (A:d, B:\frownie) \\ (B:\perp) & s = q_d \wedge i = (A:\_, B:\frownie) \end{cases}$$

- transition function

$$\lambda(\Sigma)(s,i) = \begin{cases} q_d & s = q_0 \wedge i = (A:d, B:\frownie) \\ q_d & s = q_{d'} \wedge i = (A:d, B:\smilie) \\ q_0 & otherwise \end{cases}$$

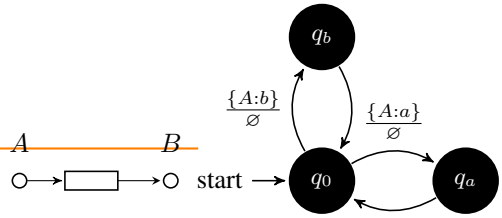

Fig. 4. PMM-based Semantics of FIFO, where $\Sigma = \{a, b\}$

Definition 2 (Production of Parameterized Mealy Machines): Now we're going to define the production operator *prod* of two parameterized mealy machines as,

$$prod(PM_1, PM_2) = PM_3$$

as follows. Here we assume that $PM_2.O \cap PM_1.I = \varnothing$

- $\forall\Sigma, PM_3.S(\Sigma) = PM_1.S(\Sigma) \times PM_2.S(\Sigma)$
- $PM_3.I = PM_1.I \cup PM_2.I - PM_1.O$
- $PM_3.O = PM_1.O \cup PM_2.O - PM_2.I$
- $PM_3.s_0 = (PM_1.s_0, PM_2.s_0)$
- $\forall\Sigma, PM_3.\delta(\Sigma)((s_1, s_2), i) =$

$$\begin{cases} (Out_1 + Out_2)|_{PM_3.O} & \ominus \wedge Out_2 \neq \ominus \\ \ominus & otherwise \end{cases}$$

  where we have
  - $In_1 = i|_{PM_1.I}$
  - $Out_1 = PM_1.\delta(\Sigma)(s_1, In_1)$
  - $In_2 = (Out_1 + i)|_{PM_2.I}$
  - $Out_2 = PM_2.\delta(\Sigma)(s_2, In_2)$
- $\forall\Sigma, PM_3.\lambda(\Sigma)((s_1, s_2), i) = (s_1', s_2')$ where we have
  - $s_1' = PM_1.\lambda(\Sigma)(s_1, In_1)$
  - $s_2' = PM_2.\lambda(\Sigma)(s_2, In_2)$

IV. From Blackbox to Timed Connectors

A. Adapted L* Algorithm

B. Equivalence Query

Generally, equivalence query has been proved impossible in blackbox models [1]. However, in this section, we're showing that in certain circumstances, equivalence query can be implemented with no approximation.

Equivalence queries are used to search for counter-examples. But what makes counter-examples even existing? In Reo models, we believe that *FIFO* channels and *Timer* channels are to blame. Here we present a brief example. To make things eaiser, we have only untimed Reo here.

A *Switching* connector in Fig.5 has two source-ends A, B and one sink-end C. In a nutshell, datum come from A and be stored temporarily in buffers. These datum will never flow out

example of production, maybe fifo+timer

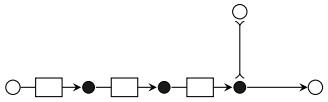

Fig. 5. A Switching Connector S with Three Buffers

until signals come to B. With the Mealy-Machine semantics given, the semantics of *Switching* connectors can be defined as following Fig.6.

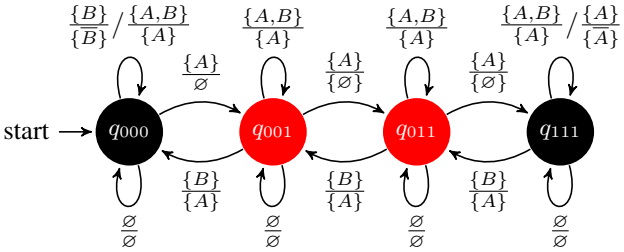

Fig. 6. *Gate* as Mealy Machine $\mathcal{M}(S)$

According to the production of Mealy-Machines, 8 different states should be found in $\mathcal{M}(S)$. We use q_{abc} to denote them. q_{001} indicates that the last buffer is filled and others are empty, and q_{111} means that there are no space in any buffer. It's obvious that some states like q_{100} is unreachable.

We say if two states are *similiar* iff. they have the same output signature.

Theorem 1 (Bound of Counter-Example): Assume that an observation table has already been closed with maximum query length l. An counter-example of length $l+2$ would be found iff. there're possible counter-examples.

find a better phrase?

Proof: Sketch of this proof are shown in several points:

1) Hypothesises are subgraphs of the semantics graph.
2) Inputs on reverse edges are complementary.
3)

■

V. Case Studies

A. Alternator

B. Timed TODO

VI. Conclusion and Future Work

References

[1] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75(2):87–106, 1987.

[2] F. Arbab. Reo: a channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.

[3] Y. Chen, C. Hsieh, O. Lengál, T. Lii, M. Tsai, B. Wang, and F. Wang. PAC learning-based verification and model synthesis. *CoRR*, abs/1511.00754, 2015.

[4] W. Daelemans. Colin de la higuera: Grammatical inference: learning automata and grammars - cambridge university press, 2010, iv + 417 pages. *Machine Translation*, 24(3-4):291–293, 2010.

[5] H. Raffelt and B. Steffen. Learnlib: A library for automata learning and experimentation. In L. Baresi and R. Heckel, editors, *Fundamental Approaches to Software Engineering, 9th International Conference, FASE 2006, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2006, Vienna, Austria, March 27-28, 2006, Proceedings*, volume 3922 of *Lecture Notes in Computer Science*, pages 377–380. Springer, 2006.

Todo list