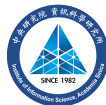


# Assume-Guarantee Reasoning – Overview

Bow-Yaw Wang

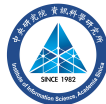
Institute of Information Science  
Academia Sinica, Taiwan

July 6, 2009



# Motivation

- Assume-guarantee reasoning can alleviate the state explosion problem in model checking.
- There are two major issues:
  - ▶ How to devise proper assumptions?
  - ▶ How to develop new proof rules?
- Interestingly, these two issues may be related to program analysis.
- Hopefully, we can combine static analysis and model checking in new perspectives.



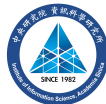
## 1 Assume-Guarantee Reasoning

## 2 Algorithmic Learning Theory

## 3 Our Related Works

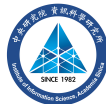
- Learning Regular  $\omega$ -Languages
- Learning Minimal Separating Finite Automata
- Intuitionistic Interpretation

## 4 Conclusions



# Automata-Theoretic Framework I

- Let  $M$  denote a system (a program, protocol, or even circuit).
- Let  $P$  denote a property (mutual exclusion, starvation freedom).
- $M \models P$  (informally) means “the system  $M$  satisfies the property  $P$ .”
- In early days, various formalizations of the  $\models$  relation were proposed.
- Automata-theoretic framework was introduced in the 80’s by Moshe Vardi.
  - ▶ It is widely used in model checking community now.

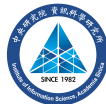


# Automata-Theoretic Framework II

- Let  $M$  be a finite automaton specifying a system.
  - ▶ System behaviors are strings accepted by  $M$ .
- Let  $P$  be a finite automaton specifying a property.
  - ▶ Intended behaviors are strings accepted by  $P$ .
- Let  $A$  be a finite automaton. Define

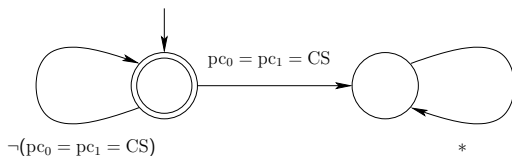
$$L(A) = \{\alpha : \alpha \text{ is accepted by } A\}.$$

- Then  $M \models P$  can be formalized as  $L(M) \subseteq L(P)$ .

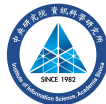


# Automata-Theoretic Framework III

- Many model checking problems can be reduced to formal language problems.
- For instance, suppose  $M_P$  is Peterson's algorithm.
- To check whether  $M_P$  satisfies mutual exclusion, consider the following automaton  $P_{ME}$ :

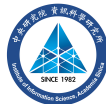


- Finite automata can be constructed from a logic formula.
  - ▶ For instance,  $G(pc_0 \neq CS \vee pc_1 \neq CS)$ .



# Automata-Theoretic Framework IV

- We can also consider a composition  $M_0 \parallel M_1$  of systems  $M_0$  and  $M_1$ .
- For instance, consider  $M_0 \parallel M_1$  to have all behaviors shared by  $M_0$  and  $M_1$ .
  - ▶ Thus,  $L(M_0 \parallel M_1) = L(M_0) \cap L(M_1)$ .
- Concise specifications of systems and properties are possible.
  - ▶ For instance,  $M_{P_0}$  and  $M_{P_1}$  specify the processes in Peterson's algorithm.
  - ▶ We can check if  $M_{P_0} \parallel M_{P_1} \models P_{ME}$ .
- However, the number of states grows exponentially in the number of processes.
  - ▶ This is called the **state explosion problem**.

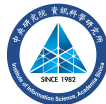


# Compositional Reasoning

- Compositional reasoning aims to solve verification problems by divide and conquer.
- Consider the following proof rule:

$$\frac{M_0 \models P}{M_0 \parallel M_1 \models P}$$

- ▶ Informally, if we can prove  $M_0$  satisfies  $P$ , then  $M_0 \parallel M_1$  satisfies  $P$ .
- It is easy to see that the rule is sound. But it is useless in practice.
  - ▶ Each process in Peterson's algorithm does not satisfy mutual exclusion alone.
- Contexts must be considered in compositional reasoning.



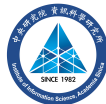


# Assume-Guarantee Reasoning

- In assume-guarantee reasoning, contextual assumptions are provided by users. Consider

$$\frac{M_0 \parallel A \models P \quad M_1 \models A}{M_0 \parallel M_1 \models P}$$

- ▶ Informally, if we can find an assumption  $A$  such that (1)  $M_0 \parallel A$  satisfies  $P$ ; and (2)  $M_1$  satisfies  $A$ , then  $M_0 \parallel M_1$  satisfies  $P$ .
- Intuitively, assumptions are abstractions of contexts.
- With proper abstractions, problems are divided into simpler subproblems.



# Issues in Assume-Guarantee Reasoning

However, there are two issues in assume-guarantee reasoning:

① How to come up with “proper assumption?”

- ▶ Finding assumptions requires insights to system designs.
  - ★ Try to find an assumption to prove Peterson’s algorithm.
- ▶ Is there a way to generate assumptions automatically?
  - ★ Yes! I will cover this exciting idea here.

② How to find new proof rules?

- ▶ Some proof rules are rather perplexing. Consider

$$\frac{M_0 \parallel P_1 \models P_0 \quad P_0 \parallel M_1 \models P_1}{M_0 \parallel M_1 \models P_0 \parallel P_1} \mathcal{C}$$

- ▶ Is there a way to verify these proof rules automatically?
  - ★ Yes! If you are interested, please talk to me.



# Issues in Assume-Guarantee Reasoning

However, there are two issues in assume-guarantee reasoning:

① How to come up with “proper assumption?”

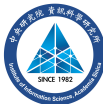
- ▶ Finding assumptions requires insights to system designs.
  - ★ Try to find an assumption to prove Peterson’s algorithm.
- ▶ Is there a way to generate assumptions automatically?
  - ★ Yes! I will cover this exciting idea here.

② How to find new proof rules?

- ▶ Some proof rules are rather perplexing. Consider

$$\frac{M_0 \parallel P_1 \models P_0 \quad P_0 \parallel M_1 \models P_1}{M_0 \parallel M_1 \models P_0 \parallel P_1} \mathcal{C}$$

- ▶ Is there a way to verify these proof rules automatically?
  - ★ Yes! If you are interested, please talk to me.



# Issues in Assume-Guarantee Reasoning

However, there are two issues in assume-guarantee reasoning:

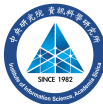
- ① How to come up with “proper assumption?”
  - ▶ Finding assumptions requires insights to system designs.
    - ★ Try to find an assumption to prove Peterson’s algorithm.
  - ▶ Is there a way to generate assumptions automatically?
    - ★ Yes! I will cover this exciting idea here.
- ② How to find new proof rules?
  - ▶ Some proof rules are rather perplexing. Consider
$$\frac{M_0 \parallel P_1 \models P_0 \quad P_0 \parallel M_1 \models P_1}{M_0 \parallel M_1 \models P_0 \parallel P_1} C$$
  - ▶ Is there a way to verify these proof rules automatically?
    - ★ Yes! If you are interested, please talk to me.



# Issues in Assume-Guarantee Reasoning

However, there are two issues in assume-guarantee reasoning:

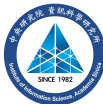
- ① How to come up with “proper assumption?”
  - ▶ Finding assumptions requires insights to system designs.
    - ★ Try to find an assumption to prove Peterson’s algorithm.
  - ▶ Is there a way to generate assumptions automatically?
    - ★ Yes! I will cover this exciting idea here.
- ② How to find new proof rules?
  - ▶ Some proof rules are rather perplexing. Consider
$$\frac{M_0 \parallel P_1 \models P_0 \quad P_0 \parallel M_1 \models P_1}{M_0 \parallel M_1 \models P_0 \parallel P_1} \mathcal{C}$$
  - ▶ Is there a way to verify these proof rules automatically?
    - ★ Yes! If you are interested, please talk to me.



# Issues in Assume-Guarantee Reasoning

However, there are two issues in assume-guarantee reasoning:

- ① How to come up with “proper assumption?”
  - ▶ Finding assumptions requires insights to system designs.
    - ★ Try to find an assumption to prove Peterson’s algorithm.
  - ▶ Is there a way to generate assumptions automatically?
    - ★ Yes! I will cover this exciting idea here.
- ② How to find new proof rules?
  - ▶ Some proof rules are rather perplexing. Consider
$$\frac{M_0 \parallel P_1 \models P_0 \quad P_0 \parallel M_1 \models P_1}{M_0 \parallel M_1 \models P_0 \parallel P_1} \mathcal{C}$$
  - ▶ Is there a way to verify these proof rules automatically?
    - ★ Yes! If you are interested, please talk to me.



# Algorithmic Learning Theory

- Let  $U$  be the universe.
- Let  $L \subseteq U$  be an unknown set.
- With the help of a “teacher,” a learning algorithm aims to find a finite representation  $R$  such that  $R$  represents  $L$ .
- For instance, consider  $U$  to be the set of finite strings and  $L$  a regular language. Suppose a teacher gives  $(\alpha_i, c_i)$  such that

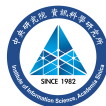
$$\begin{aligned}\alpha_i \in L & \quad \text{if } c_i = \text{true} \\ \alpha_i \notin L & \quad \text{if } c_i = \text{false}.\end{aligned}$$

- A learning algorithm aims to compute a finite automaton representing  $L$  by observing  $(\alpha_i, c_i)$  for  $i = 0, \dots, N$ .



# Advices are Important

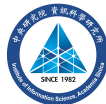
- Learning algorithms depends on types of teachers heavily.
  - ▶ In the previous slide, a learning algorithm cannot be correct if the teacher always gives the same advice (say,  $(\epsilon, \text{true})$ ).
- It is necessary to formulate problems carefully in algorithm learning theory.
  - ▶ Improper formulations can lead to infeasible solutions or even make the problem unsolvable.





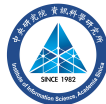
# The $L^*$ Algorithm

- In 80's, Angluin introduced the  $L^*$  algorithm for learning an unknown regular language  $L$ .
- The  $L^*$  algorithm can make two types of queries:
  - ① Membership: is the string  $\alpha$  in  $L$ ?
    - ★ The teacher answers “yes” or “no.”
  - ② Conjecture: is the finite automaton  $C$  accepting  $L$ ?
    - ★ If  $L(C) = L$ , the teacher answers “yes.”
    - ★ If  $L(C) \neq L$ , the teacher gives a counter example in  $(L(C) \setminus L) \cup (L \setminus L(C))$ .
- Given such a teach,  $L^*$  generates the minimal finite automaton  $M$  such that  $L(M) = L$  with a polynomial number of queries (in the size of  $M$ ).



# Why does Machine Learning Matter?

- Notice that a teacher in the  $L^*$  algorithm does not need any insight to the unknown language.
- Consider the language  $L = 1^*(1^*01^*01^*)^*1^*$ .
- By examining the regular expression, we realize that  $L$  contains all finite strings with an even number of 0's.
  - ▶ To see this, certain insights are needed.
- However, checking  $\alpha \stackrel{?}{\in} L$  or  $L(C) \stackrel{?}{=} L$  can be mechanized.
  - ▶ By translating the regular expression to a finite automaton.
- If we can simulate a teacher in assume-guarantee reasoning, the  $L^*$  can learn proper assumptions for us!



# Assume-Guarantee Reasoning through Learning I

- Consider again the proof rule:

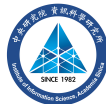
$$\frac{M_0 \parallel A \models P \quad M_1 \models A}{M_0 \parallel M_1 \models P}$$

- For membership query  $\alpha \stackrel{?}{\in} L(A)$ , we check if  $\alpha \in L(P) \cup \overline{L(M_0)}$ .
- For conjecture query  $L(C) \stackrel{?}{=} L(A)$ , we check if  $M_0 \parallel C \models P$  and  $M_1 \models C$ .
  - ▶ If yes, C is a proper assumption;
  - ▶ If no, check whether the counter example is in  $L(M_0 \parallel M_1) \setminus L(P)$ .
    - ★ If yes, P is falsified by the counter example;
    - ★ If no, returns the counter example to  $L^*$  to refine C.
- Eventually, either
  - the assumption A such that  $L(A) = L(P) \cup \overline{L(M_0)}$  is generated; or
  - a counter example is found.



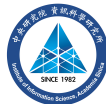
# Assume-Guarantee Reasoning through Learning II

- In principle, the unknown language is the weakest assumption  $L(P) \cup \overline{L(M_0)}$ .
  - ▶ Membership queries are resolved accordingly.
- However, we may find a proper assumption before reaching the weakest assumption.
  - ▶ Any assumption A such that  $L(M_1) \subseteq L(A) \subseteq L(P) \cup \overline{L(M_0)}$  will do.
- The idea is not to generate assumptions intrinsically but externally.
  - ▶ It is impossible to “guess” contextual assumptions.
  - ▶ It is nevertheless possible to “refine” them incrementally.



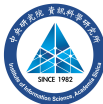
# Our Related Works

- Since the introduction of machine learning to assumption generation in 2003, several papers have been published in this area.
- We are interested in the following problems:
  - ① Extending the framework to regular  $\omega$ -languages;
  - ② Finding better assumptions;
  - ③ Deriving more proof rules automatically.



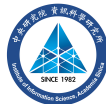
# Learning Regular $\omega$ -Languages I

- For liveness properties (“good things will happen”), it is more natural to use infinite strings (called  $\omega$ -strings).
- One can discuss  $\omega$ -strings accepted by a finite automaton.
  - ▶ The problem is to define accepting conditions since there is no final state in an infinite run.
- A finite automaton thus accepts set of  $\omega$ -strings (called an  $\omega$ -language).
- In 60’s, Büchi showed that the class of  $\omega$ -languages accepted by finite automata coincides with the class of regular  $\omega$ -languages.
- In model checking community, regular  $\omega$ -languages are used very often.



# Learning Regular $\omega$ -Languages II

- In order to extend assume-guarantee reasoning through learning to liveness properties, we need a learning algorithm for regular  $\omega$ -languages.
- However,  $L^*$  depends on the Myhill-Nerode theorem.
  - ▶ It is also used to prove the existence of minimal deterministic finite automaton for regular languages.
- But there is no similar theorem for finite automata over  $\omega$ -strings.
  - ▶ Most intriguingly, deterministic and non-deterministic finite automata may have different expressive power over  $\omega$ -strings.

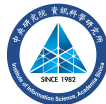


# Learning Regular $\omega$ -Languages III

- Our idea is to represent regular  $\omega$ -languages by regular languages.
- For any  $\omega$ -language  $L$ , define

$$L_{\#} = \{u\#v : uv^{\omega} \in L\}.$$

- If  $L$  is a regular  $\omega$ -language, then  $L_{\#}$  is a regular language.
- For any unknown regular  $\omega$ -language  $L$ , we use  $L^*$  to learn  $L_{\#}$ .
- When the finite automaton  $M_{\#}$  such that  $L(M) = L_{\#}$  is generated, we convert  $M_{\#}$  to a finite automaton  $M$  accepting  $L$ .
- This is a joint work with Azadeh Farzon, Yu-Fang Chen, Edmund M. Clarke, and Yih-Kuen Tsay.





# Learning Minimal Assumptions I

- Recall the proof rule:

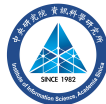
$$\frac{M_0 \parallel A \models P \quad M_1 \models A}{M_0 \parallel M_1 \models P}$$

- Any assumption A such that

$$L(M_1) \subseteq L(A) \subseteq L(P) \cup \overline{L(M_0)}$$

suffices.

- In practice, we would like A to have as few states as possible.
  - ▶ Otherwise, checking  $M_0 \parallel A \models P$  may be more expensive than checking  $M_0 \parallel M_1 \models P$ .
  - ▶ Particularly, generating the weakest assumption does not necessarily alleviate the state explosion problem.



# Learning Minimal Assumptions II

- Let  $J$  and  $K$  be regular languages.
- Consider a finite automaton  $S$  such that

$$J \subseteq L(S) \text{ and } L(S) \cap K = \emptyset.$$

- This is called a **separating automaton** for  $J$  and  $K$ .
- The **minimal** separating automaton for  $J$  and  $K$  is a separating automaton with the least number of states.
- Intuitively,  $J$  contains all the good behaviors and  $K$  contains all the bad behaviors.
- A separating automaton has all good behaviors and misses all bad behaviors.
- Finding the minimal separating automaton is a known problem in circuit synthesis.

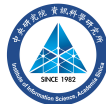


# Learning Minimal Assumptions III

- Now consider  $J = L(M_1)$  and  $K = \overline{L(P)} \cap L(M_0)$ .
- A separating automaton  $A$  for  $J$  and  $K$  satisfies

$$L(M_1) \subseteq L(A) \text{ and } L(A) \cap (\overline{L(P)} \cap L(M_0)) = \emptyset.$$

- Note that  $L(A) \cap (\overline{L(P)} \cap L(M_0)) = \emptyset$  iff  $L(A) \subseteq \overline{(\overline{L(P)} \cap L(M_0))}$  iff  $L(A) \subseteq L(P) \cup \overline{L(M_0)}$ .
- Hence a minimal separating automaton for  $J$  and  $K$  is a minimal assumption.
- This is a joint work with Yu-Fang Chen Azadeh Farzan, Edmund M. Clarke, and Yih-Kuen Tsay.



# Proof Rule Derivation I

- So far, we assume that the proof rule is fixed. We only consider the following rule:

$$\frac{M_0 \parallel A \models P \quad M_1 \models A}{M_0 \parallel M_1 \models P}$$

- Of course, it is not hard to establish the soundness of this rule.
  - ▶ Assume  $L(M_0) \cap L(A) \subseteq L(P)$  and  $L(M_1) \subseteq L(A)$ . We have  $L(M_0) \cap L(M_1) \subseteq L(M_0) \cap L(A) \subseteq L(P)$ .
- But there are other rules such as

$$\frac{M_0 \parallel A_0 \models P \quad M_1 \parallel A_1 \models P \quad \overline{A_0} \parallel \overline{A_1} \models P}{M_0 \parallel M_1 \models P}$$

- And circular rules such as

$$\frac{M_0 \parallel P_1 \models P_0 \quad P_0 \parallel M_1 \models P_1}{M_0 \parallel M_1 \models P_0 \parallel P_1} C$$



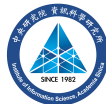
# Proof Rule Derivation II

- By identifying classes of languages as models of classical or intuitionistic logic, we can establish these rules automatically.
- If you are interested, we can discuss it offline.

```
Coq < Goal forall M0 M1 A P : Prop,  
Coq <   ((M0 /\ A -> P) /\ (M1 -> A)) ->  
Coq <   (M0 /\ M1 -> P) .  
1 subgoal
```

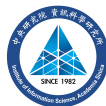
```
=====  
forall M0 M1 A P : Prop, (M0 /\ A -> P) /\ (M1 -> A) -> M0 /\ M1 -> P
```

```
Unnamed_thm < tauto .  
Proof completed.
```



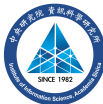
# Conclusions

- For decades, the research community tried to devise contextual assumptions by inspecting intrinsic properties.
- These attempts fail because it is so hard to “guess” designers’ intention.
- Algorithmic learning theory however does not posit to attain intelligence.
- Instead, it reformulates the problem such that mechanical solutions are possible.
- It is indeed useful in finding contextual assumptions.
  - ▶ Few people would believe it is possible a few years ago.



# Relation with Program Analysis

- Can machine learning be applied to invariant generation?
  - ▶ If invariants are in propositional logic, it may be possible.
    - ★ There is an algorithm for learning Boolean formulae.
  - ▶ For invariants in separation logic, it is unclear.
    - ★ What about the fragment of symbolic heaps  $\Pi|\Sigma$ ?
- Interpolations have been used rather often recently.
  - ▶ An **interpolant**  $I$  of  $A$  and  $B$  is a formula such that (1)  $A \Rightarrow I$ ; and (2)  $I \wedge B$  is a contradiction. Moreover,  $\text{var}(I) \subseteq \text{var}(A) \cap \text{var}(B)$ .
  - ▶ Techniques for abstraction refinement have been developed.
  - ▶ Is there an interpolation theorem for (fragments of) separation logic?



# Thank you!

