ASSUME GUARANTEE OR REPAIR

COMPOSITIONAL VERIFICATION AND REPAIR OF C-LIKE PROGRAMS

Hadar Frenkel
the Technion, Israel

Joint work with Orna Grumberg, Corina Pasareanu, and Sarai Sheinvald

@TACAS 2020
Model Checking
Model Checking

Number of states in the system model grows exponentially with the number of components in the system.

YES!

Repair!

specification

component

component

component
Model Checking

Component specification

Yes! NO! +

Counter example

Number of states in the system model grows exponentially with the number of components in the system

State Explosion Problem

Repair!
COMPOSITIONAL VERIFICATION AND REPAIR OF C-LIKE PROGRAMS

• Model checking and repair algorithm for communicating systems
• Exploit the partition of the system into components
Communicating Systems

- C-like programs
- Each component is described as a control-flow graph (automaton)
  - Alphabet: program statements & communication channels
- \( \text{In}\ x_1 \) – reads a value to \( x_1 \) through channel \( \text{In} \)
- \( \text{enc}\ x_1 \) – sends the value of \( x_1 \) through channel \( \text{enc} \)

1: while (true)
2:   pass = readInput;
3:   while (pass \leq 999)
4:       pass = readInput;
5:   pass2 = encrypt(pass);
Example

Synchronization using read-write channels, Interleaving on all other alphabet
Example

Synchronization using read-write channels, Interleaving on all other alphabet

\[
\begin{align*}
M_1: &\quad y_1 := x_1 \\
M_2: &\quad x_2 := y_1 \\
\end{align*}
\]
Example

Synchronization using read-write channels, Interleaving on all other alphabet
Example
Specifications

- Safety properties
- Alphabet:
- (Common) communication channels
- Syntactic requirements: program behavior through time
Specifications

• Safety properties
• Alphabet:
• (Common) communication channels
• Syntactic requirements: program behavior through time
• Constraints over local variables
• Semantic requirements:
  • “the entered password is different from the encrypted password”
  • “there is no overflow”
Compositional Verification

- **Assume-Guarantee (AG) paradigm** [Pnueli, 1985]:
  - assumptions represent component’s environment

- Under assumption $A$ on its environment, does the component guarantee the property?
AG Rule for Safety Properties

Find an **assumption** $A$ such that

1. Component $M_1$ **guarantees** $P$ when it is a part of a system satisfying $A$

$$M_1 \parallel A \models P$$
AG Rule for Safety Properties

Find an assumption $A$ such that

1. Component $M_1$ guarantees $P$ when it is a part of a system satisfying $A$

$$M_1 \parallel A \models P$$

2. $M_2$ satisfies $A$

$$M_2 \models A$$
AG Rule for Safety Properties

Find an **assumption** $A$ such that

1. Component $M_1$ **guarantees** $P$ when it is a part of a system satisfying $A$

\[ M_1 || A \models P \]

2. $M_2$ satisfies $A$

\[ M_2 \models A \]

Conclude that $M_1 || M_2 \models P$

\[ M_1 \ || \ M_2 \ \models \ P \]
AG Rule for Safety Properties

Find an assumption \( A \) such that

1. Component \( M_1 \) guarantees \( P \) when it is a part of a system satisfying \( A \)

\[
M_1 || A \models P
\]

Can we automatically construct \( A \)?

2. \( M_2 \) satisfies \( A \)

\[
M_2 \models A
\]

Conclude that \( M_1 || M_2 \models P \)

\[
M_1 \parallel M_2 \models P
\]
$L^*$ Algorithm for Learning Regular Languages [D. Angluin 1987]

- Given a regular language $L$, we learn a DFA $A$ such that $\mathcal{L}(A) = L$
\( L^* \) Algorithm for Learning Regular Languages [D. Angluin 1987]

- Learning assumptions for compositional verification
  [J. M. Cobleigh, D. Giannakopoulou and C. S. Pasareanu TACAS 2003]

- Given a regular language \( L \), we learn a DFA \( A \) such that \( \mathcal{L}(A) = L \)

- Membership + equivalence queries
$L^*$ Algorithm for Learning Regular Languages [D. Angluin 1987]

- Learning assumptions for compositional verification
  [J. M. Cobleigh, D. Giannakopoulou and C. S. Pasareanu TACAS 2003]

- Given a regular language $L$, we learn a DFA $A$ such that $L(A) = L$

- Try to use intermediate candidates $A_i$ as assumptions for AG rule

- But, the weakest assumption is not regular in our case

\[
\begin{align*}
M_1 || A_i & \models P \\
M_2 & \models A_i \\
M_1 || M_2 & \models P
\end{align*}
\]
A New Goal for Learning

• The teacher answers queries according to the *syntactic language* of $M_2$
• Regular since it is given as an automaton
A New Goal for Learning

- The teacher answers queries according to the *syntactic language* of $M_2$
- Regular since it is given as an automaton

\[
\begin{align*}
M_1 \parallel M_2 & \models P \\
M_2 & \models M_2 \\
M_1 \parallel M_2 & \models P
\end{align*}
\]

But I already know $M_2$ ...

You might find a much smaller assumption!
AG rule with learning

Membership queries

Is $w \in L$?
AG rule with learning

Membership queries

Is $w \in L$?

Yes \ no
AG rule with learning

Real error!

Membership queries

Is \( w \in L? \)

Yes \( \mid \) no

P is violated in \( M_1 \parallel M_2 \)
AG rule with learning

- Membership queries
- Equivalence queries

$A_i$ is violated in $M_1 \parallel M_2$
AG rule with learning

Assume - Guarantee Setting

Membership queries

Equivalence queries

1. $A_i \parallel M_1 \models P$

P is violated in $M_1 \parallel M_2$
AG rule with learning

```
AG rule with learning

Membership queries

Equivalence queries

1. $A_i \parallel M_1 \models P$

false

real error? cex$\in M_2$?

P is violated in $M_1 \parallel M_2$
```
AG rule with learning

- **Equivalence queries**
  - $A_i \parallel M_1 \models P$

- **Membership queries**

- **Strengthen assumption**

- **Real error?**
  - $cex \in M_2$?
    - **No**

- **P is violated in $M_1 \parallel M_2$**
AG rule with learning

1. $A_i \parallel M_1 \models P$

- Membership queries
- Equivalence queries

strengthen assumption

real error? $cex \in M_2$?

false

Real error!

Yes

$P$ is violated in $M_1 \parallel M_2$

Assume-Guarantee

AG rule & learning

Repair
AG rule with learning

Assume-Guarantee Setting
AG rule & learning
Repair

Membership queries

Equivalence queries
1. $A_i \parallel M_1 \models P$
2. $M_2 \subseteq A_i$

strengthen assumption

false

real error? $cex \in M_2$?

true

Yes

P is violated in $M_1 \parallel M_2$
AG rule with learning

**Setting**

**Assume-Guarantee**

**AG rule & learning**

**Repair**

---

**Membership queries**

**Equivalence queries**

1. $A_i \parallel M_1 \models P$

2. $M_2 \subseteq A_i$

---

**true**

---

**false**

---

**real error?**

$cex \in M_2$?

---

**true**

---

**false**

---

**P holds in $M_1 \parallel M_2$**

**Yes**

---

**P is violated in $M_1 \parallel M_2$**

**Yes**
AG rule with learning

Assume - Guarantee

Setting

Membership queries

Equivalence queries

1. $A_i \parallel M_1 \models P$

2. $M_2 \subseteq A_i$

true

false

strengthen assumption

true

false

weaken assumption

real error? $cex \in M_2$?

Yes

$P$ is violated in $M_1 \parallel M_2$

No

$P$ holds in $M_1 \parallel M_2$
AG rule with learning

Assume-
Guarantee

AG rule & learning

Setting

Repair

Membership queries

Equivalence queries

1. $A_i \parallel M_1 \models P$
   - true
   - false

2. $M_2 \subseteq A_i$
   - true
   - false

real error? $cex \in M_2$?

P holds in $M_1 \parallel M_2$

P is violated in $M_1 \parallel M_2$

Repair $M_2$

Return to verification with the repaired $M_2$
Assume Guarantee or Repair

• Repair by elimination of error traces

• Two types of repair
  • Syntactic repair
  • Semantic repair
Assume Guarantee or Repair

Syntactic repair – counterexample does not contain constraints
Syntactic Repair

- Implemented 3 methods to removing the trace $t$:
  - **Exact**
    remove exactly $t$ from $M_2$
  - **Approximate**
    add an intermediate state and use it to direct some traces off the accepting state, including $t$
  - **Aggressive**
    make the accepting state that $t$ reaches not-accepting
Assume Guarantee or Repair

Semantic repair – counterexample contains violated constraints of the specification
Semantic Repair

- AGR returns a counterexample $t$, for input $x_1 = 2^{63}$

- Goal: make $t$ infeasible by adding a new constraint $C$ such that
  - $(\varphi_t \land C \rightarrow false)$

- Applying abduction, quantifier elimination and simplification results in $C = (x_1 < 2^{63})$
Result

1: while (true)
2:   pass = readInput;
3:   while (pass ≤ 999)
4:     pass = readInput;
5:   pass2 = encrypt(pass);
6:   assume pass<2^{63};
AG rule with learning

Again, where $M_2 := \text{Repaired } M_2$

Model Checking

1. $M_1 \parallel M_2$
2. $M_2 \subseteq A_i$

true
false

Weaken assumption

Strengthen assumption

P holds in $M_1 \parallel M_2$

Yes

No

Real error? $cex \in M_2$?

P is violated in $M_1 \parallel M_2$

Repair $M_2$

Return to verification with the repaired $M_2$
Termination

- In case $M_1 \parallel M_2 \models P$
- $M_2$ is a correct assumption for the AG rule
- $M_2$ is regular, therefore $L^*$ terminates
  → In the case of verification, termination is guaranteed

- In case $M_1 \parallel M_2 \not\models P$
- Every iteration with an erroneous $M_2$ will result in a cex
  → In the case of an error, progress is guaranteed
Comparing Repair Methods (logarithmic scale)

#15, #16, #18, #19 apply also abduction
AGR Summary

• Modular verification for communicating systems
• Adjusting automata learning to systems with data
• Iterative and incremental verification and repair to prove correctness of repaired system
• Modular verification for communicating systems
• Adjusting automata learning to systems with data
• Iterative and incremental verification and repair to prove correctness of repaired system

Thank you! Questions?