Compositional Verification III

Dimitra Giannakopoulou and Corina Păsăreanu
CMU / NASA Ames Research Center
recap in reverse order...

- assume-guarantee reasoning
- learning framework for 2 components
- weakest assumption
assume-guarantee reasoning

**Reasons about triples:**

\[ \langle A \rangle M \langle P \rangle \]

is *true* if whenever \( M \) is part of a system that satisfies \( A \), then the system must also guarantee \( P \)

**Simplest assume-guarantee rule (ASYM):**

1. \[ \langle A \rangle M_1 \langle P \rangle \]
2. \[ \langle \text{true} \rangle M_2 \langle A \rangle \]

\[ \langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle \]
Learning assumptions for AG reasoning

Assumptions conjectured by $L^*$ are not comparable semantically.
the weakest assumption

- given component $M$, property $P$, and the interface $\Sigma$ of $M$ with its environment, generate the weakest environment assumption $WA$ such that: $\langle WA \rangle M \langle P \rangle$ holds

- weakest means that for all environments $E$: $\langle true \rangle M \parallel E \langle P \rangle \text{ IFF } \langle true \rangle E \langle WA \rangle$
STEP 1: composition, hiding, minimization

STEP 2: backward propagation of error along $\tau$ transitions

STEP 3: property extraction (subset construction & completion)
part III

- assume-guarantee reasoning
- learning framework for 2 components
- weakest assumption
  
- interface generation
- implementations & applications
- other approaches
interface generation

- beyond syntactic interfaces
  - will not invoke “close” on a file if “open” has not previously been invoked

- **safe**: accept NO illegal sequence of calls
- **permissive**: accept ALL legal sequences of calls

safe & permissive interface = weakest assumption
learning, again…

iterative solution + intermediate results
(queries)
should word w be included in L(A)?

yes / no

(conjectures)
here is an A – is it safe & permissive?

yes!

no: word w should (not) be in L(A)
safe?
permissive?
problem
queries (simulate / model check)

conjecture – safe (model check)

conjecture – permissive?

Alur et al, 2005, Henzinger et al, 2005
solution

model check for \((M_i, A_{\text{error}})\)

reached \((M_1, A_{\text{error}})\) by “a b”

query “a b”

no (“a b” should not be in \(A\))

backtrack and continue search…
invoke a model checker within a model checker?
model check for \((M_i, A_{\text{error}})\) reached \((M_1, A_{\text{error}})\) by “a b”
if (memoized(“a b”) == no)
backtrack and continue search…
example

module M

Input

\[\text{in} \rightarrow \text{send} \rightarrow \text{ack} \rightarrow \text{out} \rightarrow \text{send} \rightarrow \text{ack} \rightarrow \text{out} \rightarrow \text{in}\]

Order_{err}

\[\text{in} \rightarrow \text{out} \rightarrow \text{out} \rightarrow \text{in}\]

Output

\[\text{send} \rightarrow \text{out} \rightarrow \text{ack}\]

\[\langle \text{ack, out} \rangle?\]

\[\langle \text{ack, out, send} \rangle?\]

assumption learned for AG reasoning

weakest assumption
complete module for permissiveness check

module M

Input

Order_{err}

Complete_{Input}

queries performed on Input || Order_{err}

safety checked on Input || Order_{err} || A_{err}

permissiveness performed on Complete_{Input} || Order_{err} || A_{err}

check reachability of states:
(sink, *, error) or (*, non error, error)

\langle \text{ack, out} \rangle: \quad \text{(sink, error, error)}
in summary…

generate precise component interfaces

resolve non-determinism
dynamically & selectively
JavaPathfinder
UML statecharts

assume-guarantee reasoning

interface generation / discharge

http://javapathfinder.sourceforge.net
JPF supports model checking of UML state-machines with an approach that consists of three steps:

- translate the UML model into a corresponding Java program, using JPF’s state chart (sc) extension and application model
- choose model properties to verify, and configure verification tools accordingly
- optionally provide a guidance script that represents the environment of the model (external event sequence)
package ICSETutorial;

import gov.nasa.jpf.sc.State;

public class Input extends State {
    class S0 extends State {
        public void input() {
            setState(s1);
        }
    } S0 s0 = makeInitial(new S0());

    class S1 extends State {
        public void send() {
            setState(s2);
        }
    } S1 s1 = new S1();

    class S2 extends State {
        public void acknowledge() {
            setState(s0);
        }
    } S2 s2 = new S2();
}
AG reasoning in JPF

JavaPathfinder
(CVState.AutomatonState)

SCSafetyListener

SCSafetyAutomaton
assumptions

- choiceGeneratorAdvanced
  - if selected action leads assumption to error state then do
    “vm.getSystemState().setIgnore(true)” (backtrack)

- instructionExecuted
  - advance automaton & set CVState.AutomatonState

- stateBacktracked
  - get CVState.AutomatonState
properties

- instructionExecuted
  - advance automaton & set CVState.AutomatonState
  - if automaton reaches error state, then check() returns false

- stateBacktracked
  - get CVState.AutomatonState
how to...

run: gov.nasa.jpf.JPF

with the following arguments:

+jpf.listener=.cv.SCSafetyListener
+safetyListener1.property= Foo
interface generation in JPF

- queries and assumption safety checks
  - same as assume-guarantee reasoning
- assumption permissiveness check
  - requires special listener
**conformance listener**

- **executeInstruction**
  - if instruction to be executed is assertion violation, then perform
    “ti.skipInstruction()” (do not process exception) and
    “vm.getSystemState().setIgnored(true)” (backtrack)

- **instructionExecuted**
  - advance automaton & set CVState.AutomatonState
  - if automaton reaches error state, check memoized table (why?)
    - if counterexample stored and spurious, backtrack
    - else check() returns false

- **stateBacktracked**
  - get CVState.AutomatonState
boolean done = false;
while (!done){
    counterexample = null;
    
    SCConformanceListener assumption = new SCConformanceListener(
        new SCSafetyAutomaton(false, assume, alphabet_, "Assumption",
        CompleteModule, memoized_));
    JPF jpf = createJPFInstance(assumption, property, CompleteModule);
    jpf.run();

    Path jpfPath = assumption.getCountercaseexample();
    if (jpfPath != null){
        // nonerror in M & error in Aerr - this is what we are looking for

        counterexample = assumption.convert(jpfPath);
        if (query(counterexample)){ // cex is in L(A)
            done = true; // a real counterexample for L*
        } // otherwise you need to continue with your loop
    }
else
    done = true; // interface is permissive
}
how to…

run: gov.nasa.jpf.tools.cv.ScRunCV

with the following arguments:

+assumption.alphabet=a,b,c
+assumption.outputFile=Foo
Input component with Order Property:

```java
package ICSETutorial;

import gov.nasa.jpf.sc.State;
import gov.nasa.jpf.cv.CVState;

public class InputWithProperty extends CVState {
    class S0 extends State {
        public void input() {
            setState(s1);
        }
        public void output() {
            assert(false);
        }
    }

    S0 s0 = makeInitial(new S0());
}
```

JPF Run Configuration:

- **main:**
  ```
  gov.nasa.jpf.tools.cv.ScRunCV
  ```

- **arguments:**
  ```
  +jpf.listener=.tools.ChoiceTracker
  +assumption.outputFile=
      examples/ICSETutorial/generatedAssumption
  +assumption.alphabet=output,send,acknowledge
  +jpf.report.console.property_violation=error
  +vm.store_steps=true
  +log.info=gov.nasa.jpf.sc
  ICSETutorial.InputWithProperty
  ```

Input output example

\[
S0 = (\text{send} \rightarrow S2
     \mid \text{acknowledge} \rightarrow S1),
S1 = (\text{output} \rightarrow S1
     \mid \text{send} \rightarrow S1
     \mid \text{acknowledge} \rightarrow S1),
S2 = (\text{output} \rightarrow S3
     \mid \text{send} \rightarrow S1),
S3 = (\text{send} \rightarrow S1
     \mid \text{acknowledge} \rightarrow S0).
\]
example 2
model of the Ascent and EarthOrbit flight phases of a spacecraft

properties:

- “An event IsamRendezvous, which represents a docking maneuver with another spacecraft, fails if the LAS (launch abort system) is still attached to the spacecraft”

- “Event tliBurn (trans-lunar interface burn takes spacecraft out of the earth orbit and gets it into transition to the moon) can only be invoked if EDS (Earth Departure Stage) rocket is available”
Assumption 1:

```
iasJetisson
```

```
lasJetisson
IsamRendezvous
```

Assumption 2:

```
IsamRendezvous
```

```
tliBurn
IsamRendezvous
```

Generated interface assumptions encode Flight Rules in terms of events.
JPF run configurations

- main:
  gov.nasa.jpf.tools.cv.ScRunCV

- arguments for property 1:
  +jpf.listener=.tools.ChoiceTracker:.cv.AssertionFilteringListener
  +assertionFilter.include=tlIBurn
  +assumption.alphabet=tlIBurn,lsamRendezvous
  +assumption.outputFile=examples/jpfESAS/script/generatedAssumption1
  +jpf.report.console.property_violation=error
  +vm.store_steps=true

  jpfESAS.CEV_15EOR_LOR

- arguments for property 2:
  +jpf.listener=.tools.ChoiceTracker:.cv.AssertionFilteringListener
  +assertionFilter.include=lsamRendezvous
  +assumption.alphabet=lasJettison,lsamRendezvous
  +assumption.outputFile=examples/jpfESAS/script/generatedAssumption2
  +jpf.report.console.property_violation=error
  +vm.store_steps=true

  jpfESAS.CEV_15EOR_LOR
CEGAR for compositional verification

- **CEGAR**: counterexample guided abstraction refinement – Clarke et al. 00
  - incremental construction of abstractions
  - abstractions are conservative
  - abstract counterexamples obtained may be spurious (due to over-approximation)
  - spurious counterexamples are used for abstraction refinement
- two level compositional abstraction refinement – Chaki et al. 03
  - analyze $C_1 \parallel C_2 \parallel \ldots \parallel C_n \models P$
  - build finite-state abstractions: $A_1, A_2, \ldots A_n$
  - minimize: $M_1, M_2, \ldots M_n$
  - analyze: $M_1 \parallel M_2 \parallel \ldots \parallel M_n \models P$?
  - refine based on counterexamples
- permissive interfaces – Henzinger et al. 05
  - uses CEGAR to compute interfaces
- new result at CAV’08
build A as an abstraction of M₂

〈true〉 M₂ 〈A〉 holds by construction

check Premise 1: 〈A〉 M₁ 〈P〉

obtained counterexamples are analyzed and used to refine A

variant of CEGAR (Counter-example Guided Abstraction Refinement) with differences:

– use counterexample from one component (M₁) to refine abstraction of the other component (M₂)

– A keeps information only about the interface (and abstracts away the internal information)

implemented in LTSA; combined with alphabet refinement; compares favorably with learning approach
other related work

- minimal separating automaton for disjoint languages $L_1$ and $L_2$
  - accept all words in $L_1$
  - accept no words in $L_2$
  - have the least number of states

- assume-guarantee reasoning
  - minimal separating automaton for $L(M_2)$ and $L(M_1) \cap L(\text{coP})$

- algorithms
  - Gupta at al. 07: query complexity exponential in the size of the minimal DFAs for the two input languages
  - Chen et al. 09: query complexity quadratic in the product of the sizes of the minimal DFAs for the two input languages. Use 3 valued DFAs

- compositional verification in symbolic setting (Alur et al. 05)
- learning omega-regular languages for liveness (Farzan et al. 08)
- learning non-deterministic automata (Bollig et al. 09)
thank you!