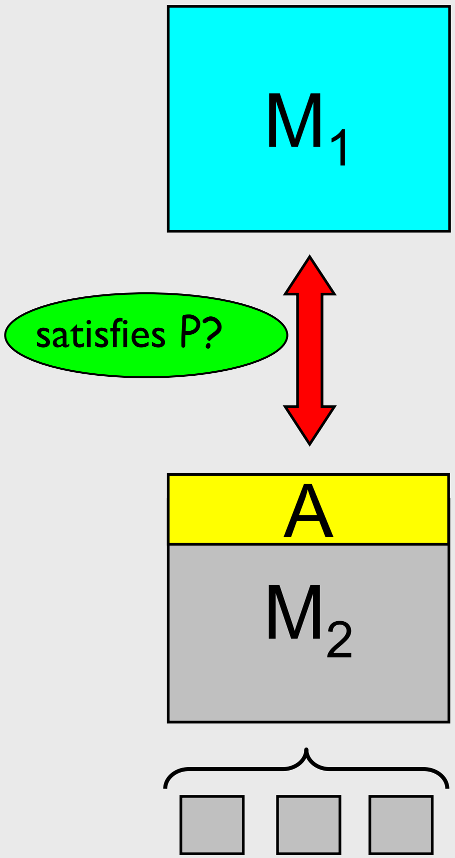
Compositional Verification III

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:

$\langle A \rangle\ M\ \langle P \rangle$

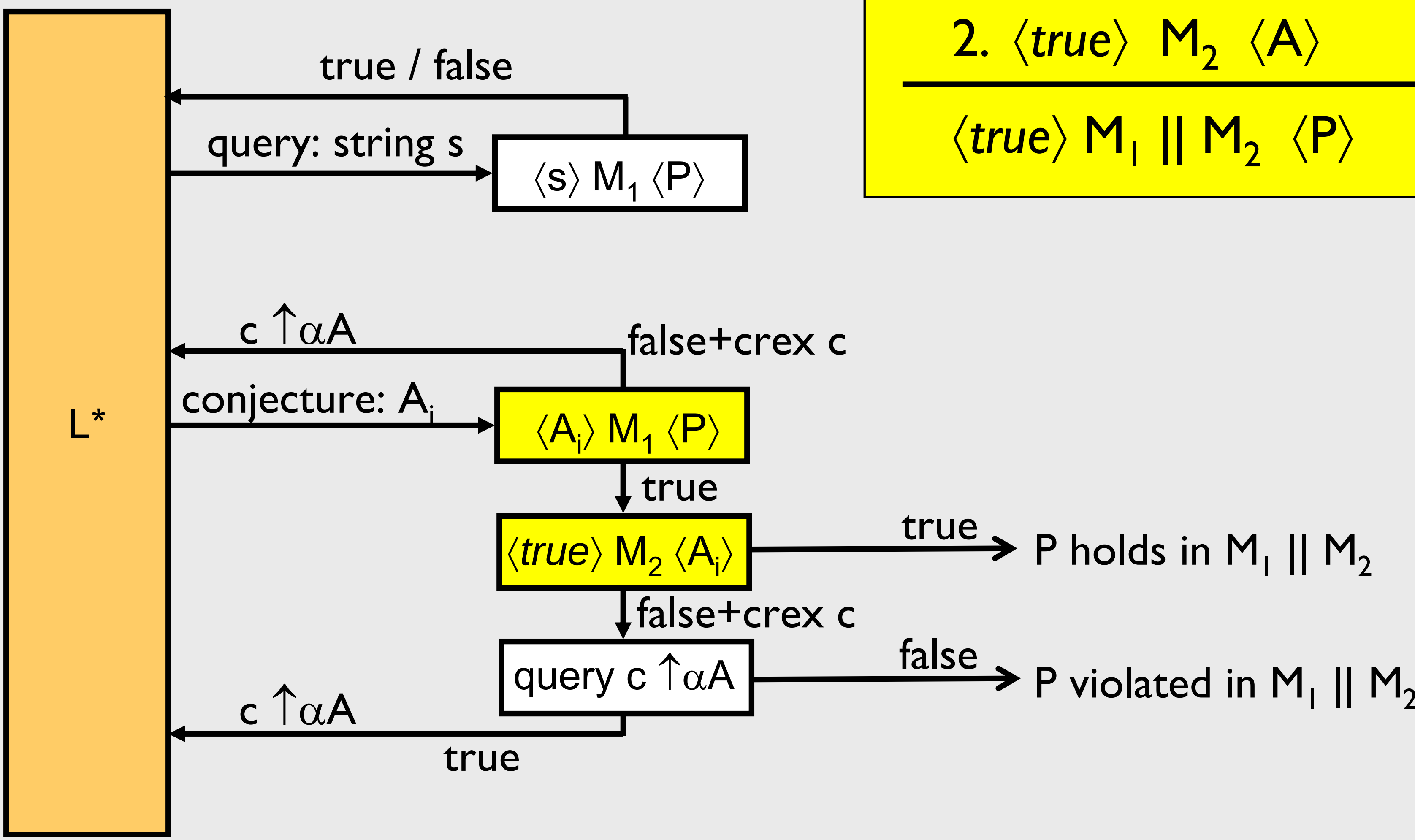
is *true* if whenever M is part of a system that satisfies A, then the system must also guarantee P

simplest assume-guarantee rule (ASYM):

$$\frac{\begin{array}{ll} 1. & \langle A \rangle\ M_1\ \langle P \rangle \\ 2. & \langle true \rangle\ M_2\ \langle A \rangle \end{array}}{\langle true \rangle\ M_1 \parallel M_2\ \langle P \rangle}$$

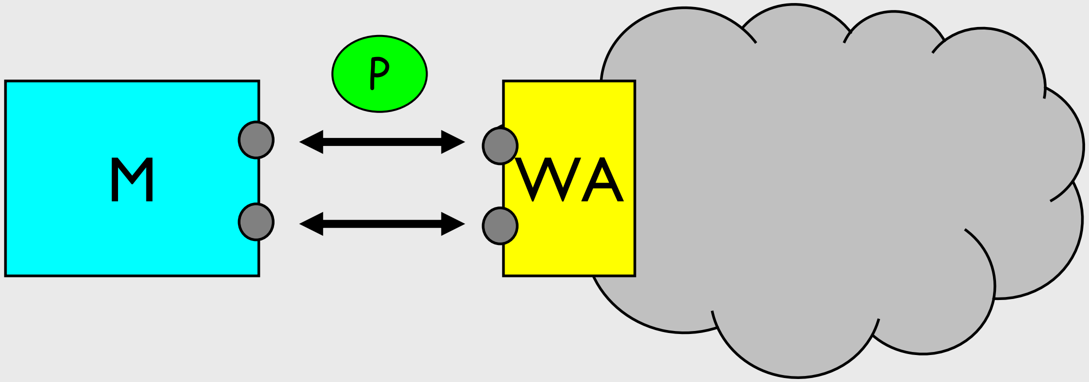
learning assumptions for AG reasoning

$$\frac{\begin{array}{ll} 1. & \langle A \rangle\ M_1\ \langle P \rangle \\ 2. & \langle true \rangle\ M_2\ \langle A \rangle \end{array}}{\langle true \rangle\ M_1 \parallel M_2\ \langle P \rangle}$$



assumptions conjectured by L* are not comparable semantically

the weakest assumption

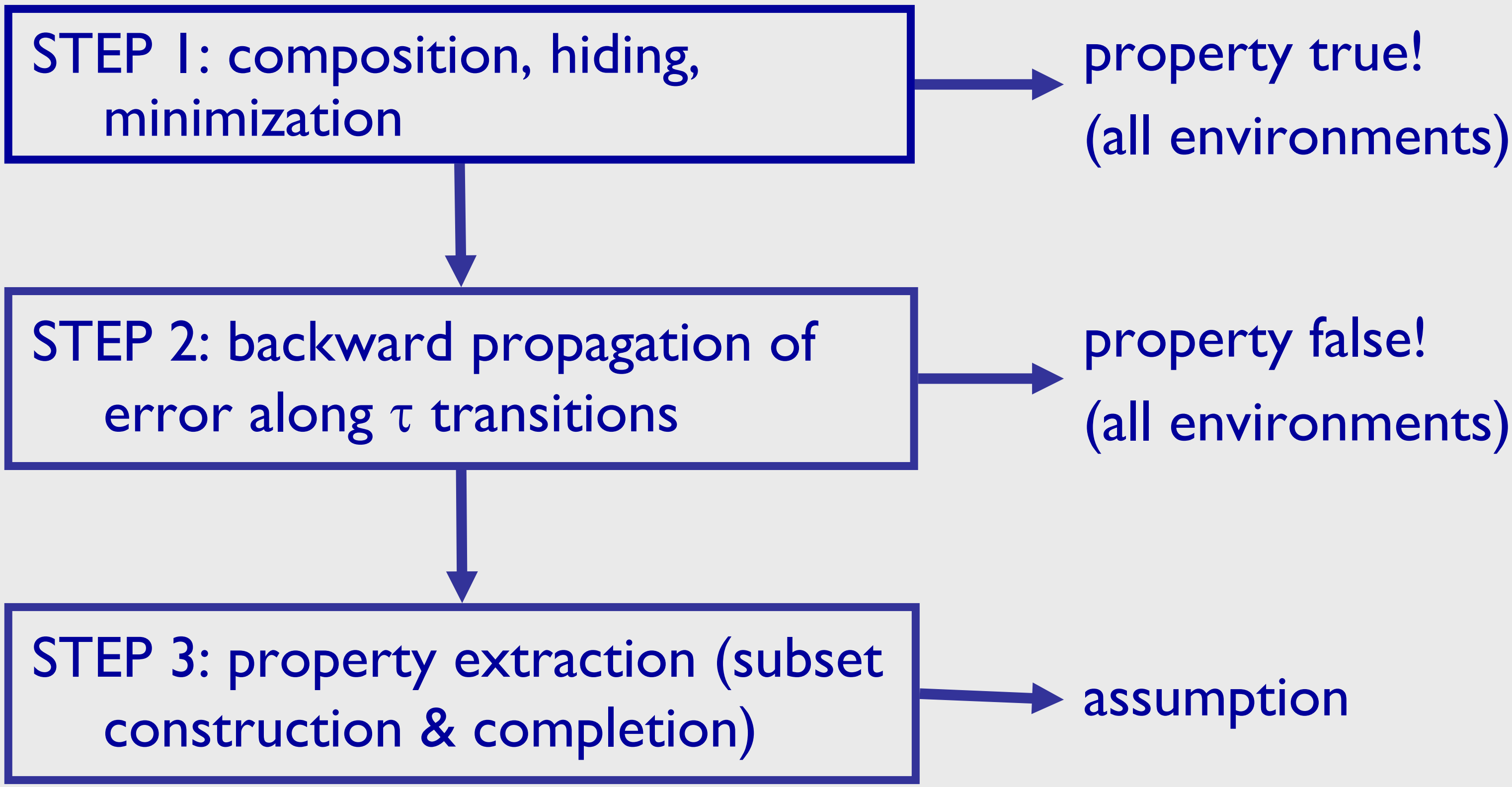


- given component M, property P, and the interface Σ of M with its environment, generate the weakest environment assumption WA such that: ⟨WA⟩ M ⟨P⟩ holds

- weakest means that for all environments E:

⟨*true*⟩ M || E ⟨P⟩ **IFF** ⟨*true*⟩ E ⟨WA⟩

assumption generation [ASE'02]



part III

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

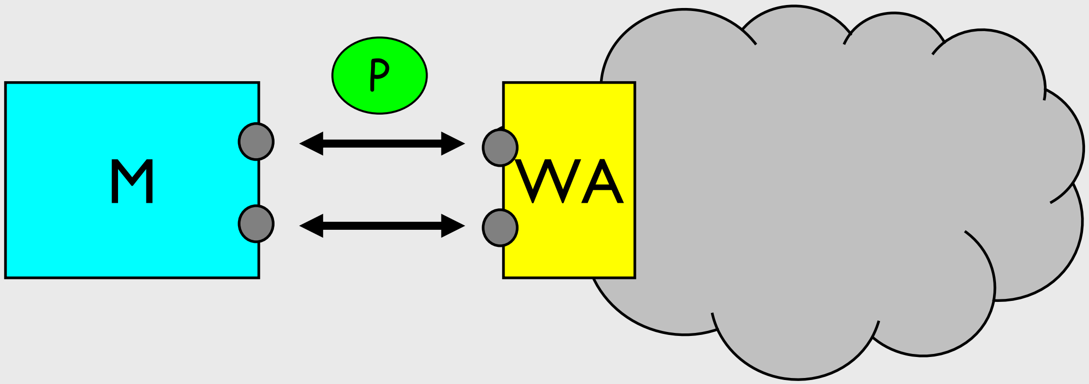
- interface generation
- implementations & applications
- other approaches

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

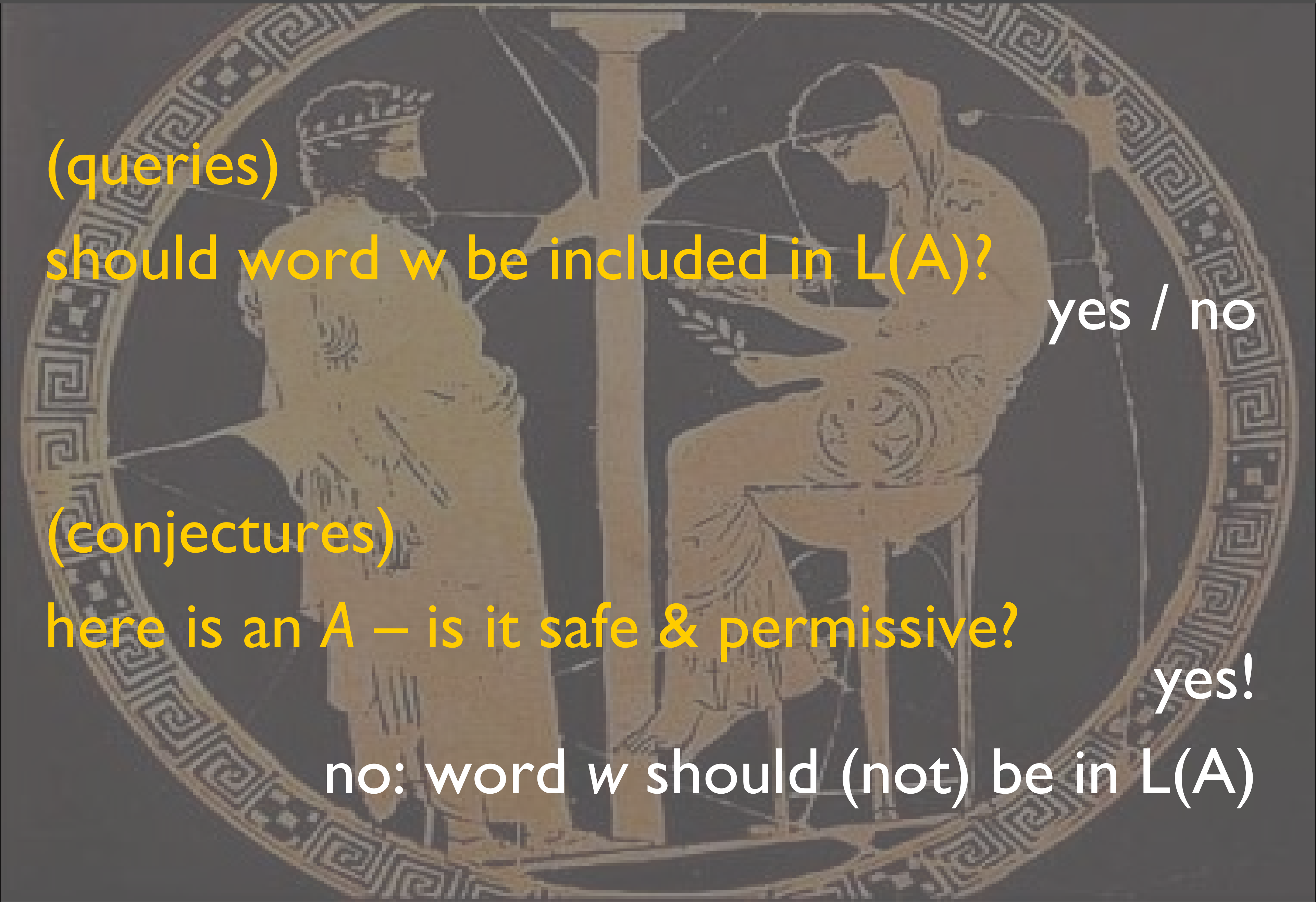
safe & permissive interface = weakest assumption

learning, again…



iterative solution + intermediate results

L* learner — the oracle



(queries)

should word w be included in L(A)?

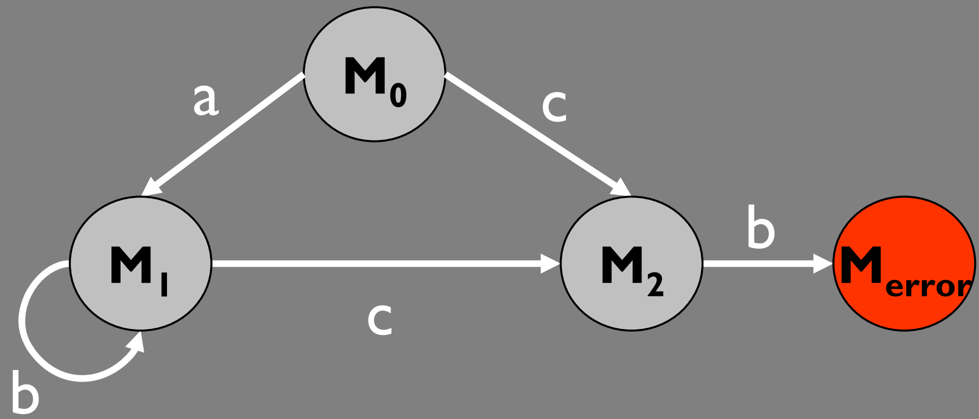
yes / no

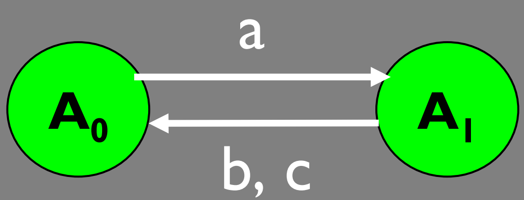
(conjectures)

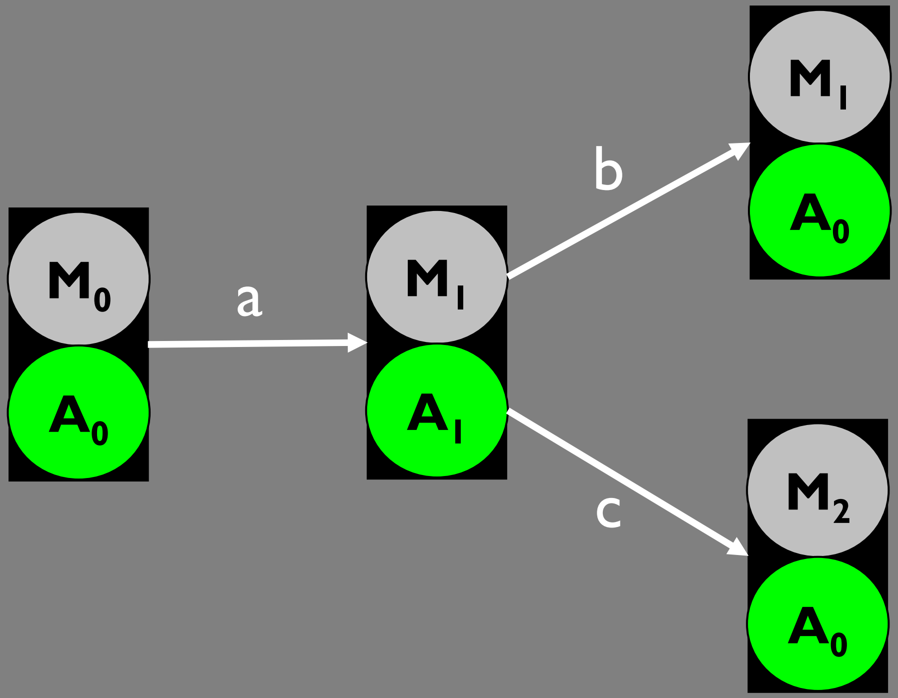
here is an *A* – is it safe & permissive?

yes!

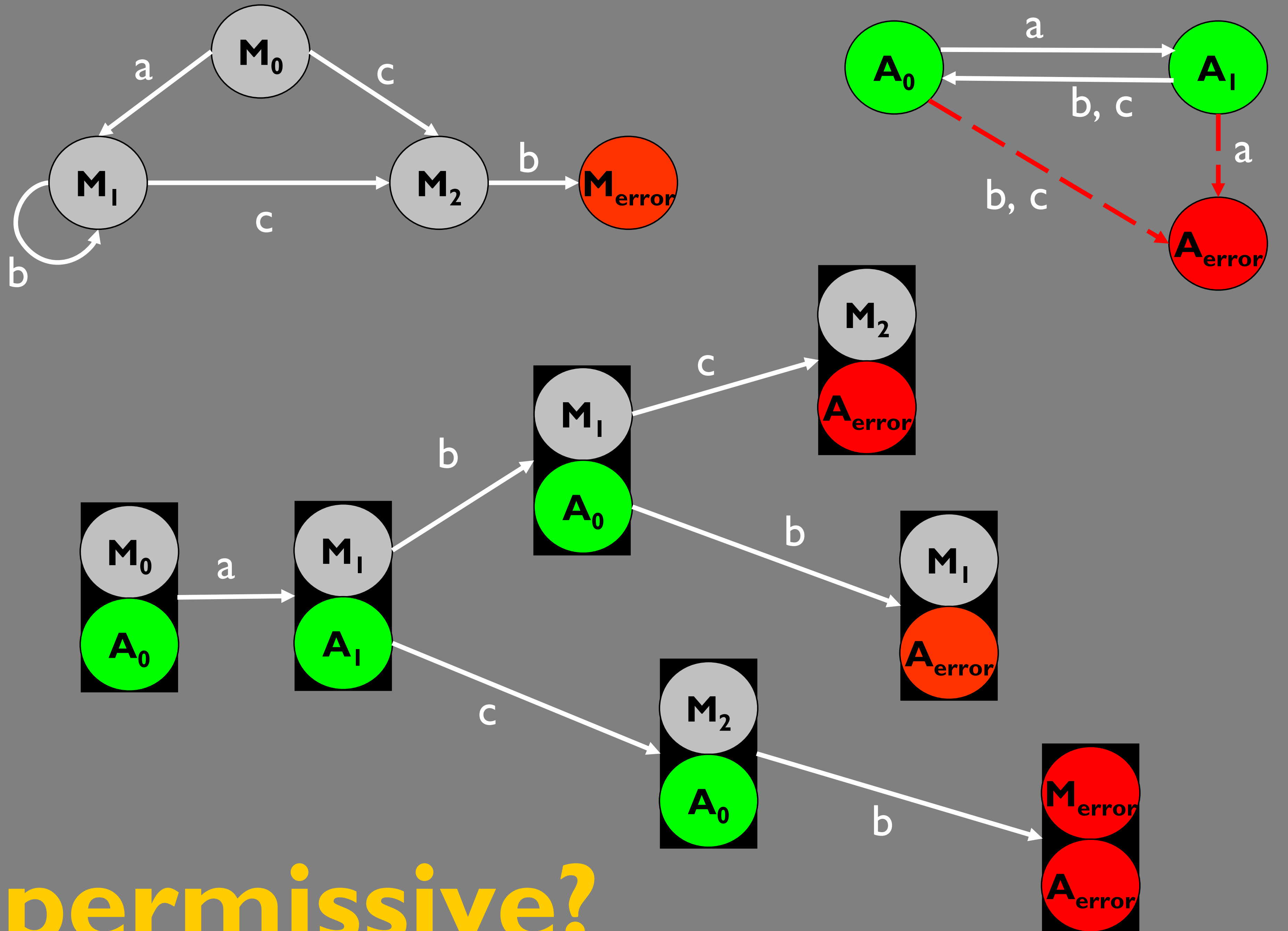
no: word *w* should (not) be in L(A)



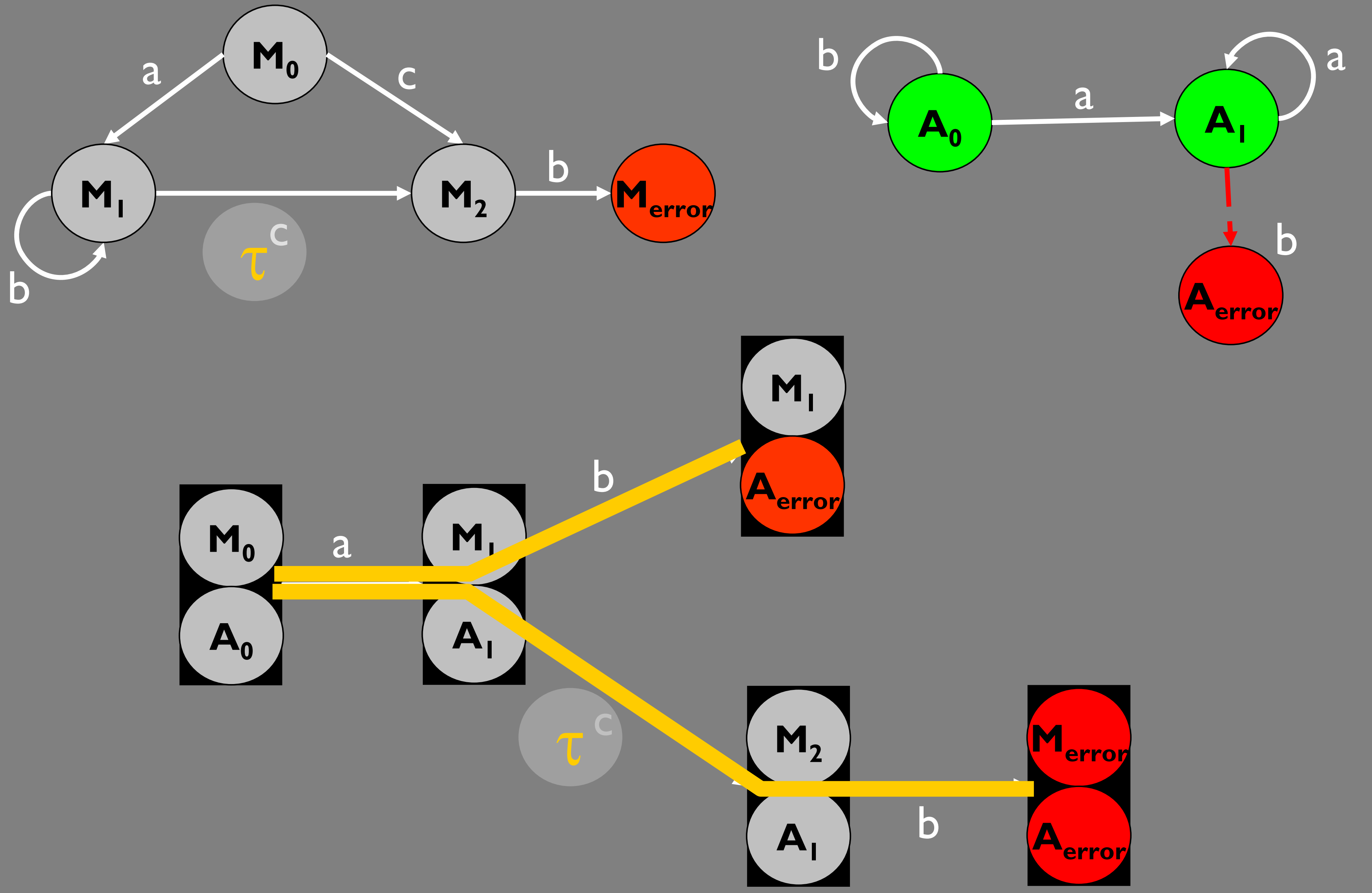




safe?



permissive?



M_0
a
c
M_1
M_2
b
M_{error}
b
τ^c
b
A_0
a
A_1
a
b
A_{error}
M_1
A_{error}
b
M_0
A_0
a
M_1
A_1
τ^c
M_2
A_1
b
M_{error}
A_{error}

problem

learning interfaces

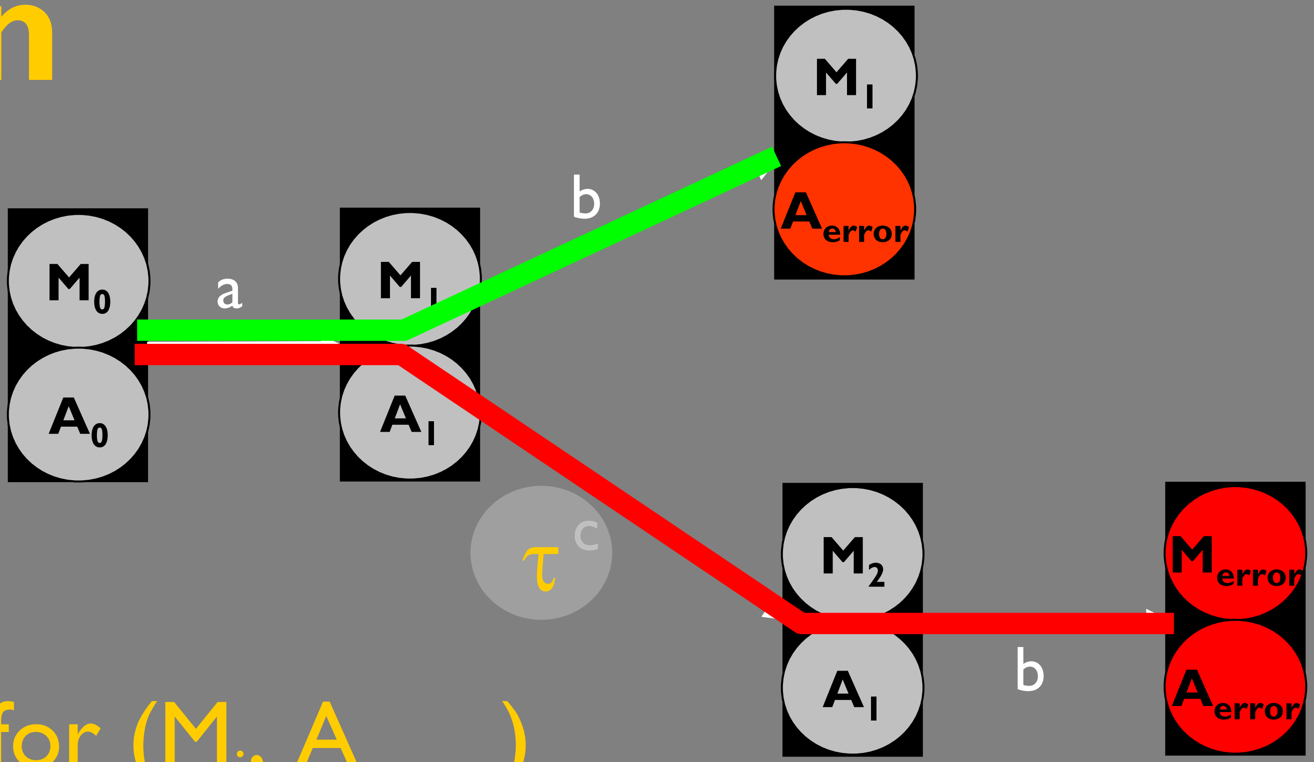
queries (simulate / model check)

conjecture – safe (model check)

conjecture – permissive?

Alur et al, 2005, Henzinger et al, 2005

solution



model check for (M_i, A_{error})

reached (M_1, A_{error}) by "a b"

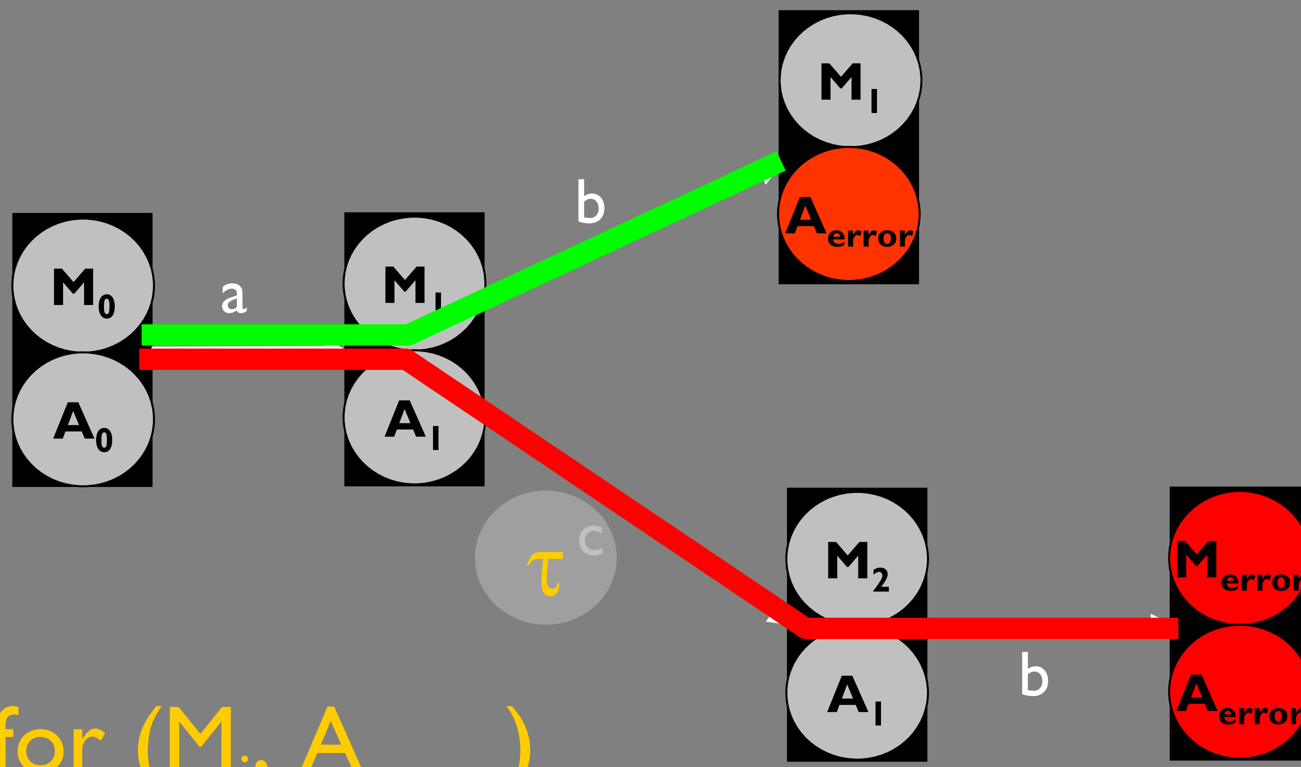
query "a b"

no ("a b" should not be in A)

backtrack and continue search…

invoke a model checker

within a model checker?



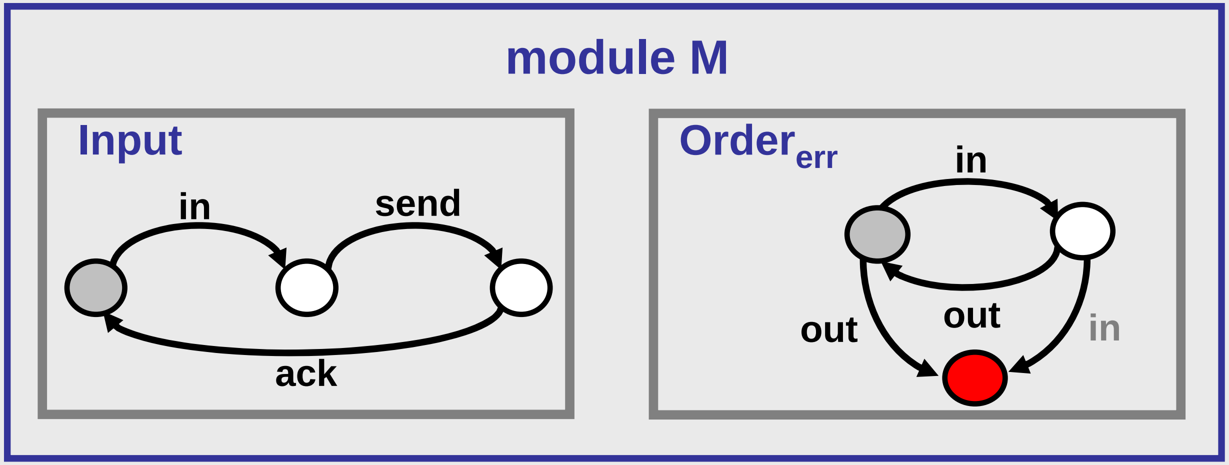
model check for (M_i, A_{error})

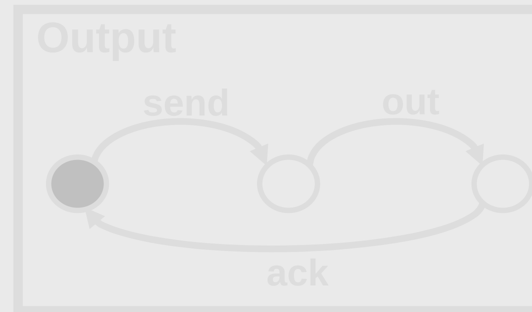
reached (M_1, A_{error}) by “a b”

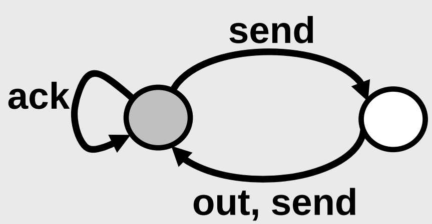
if (memoized(“a b”) == no)

backtrack and continue search…

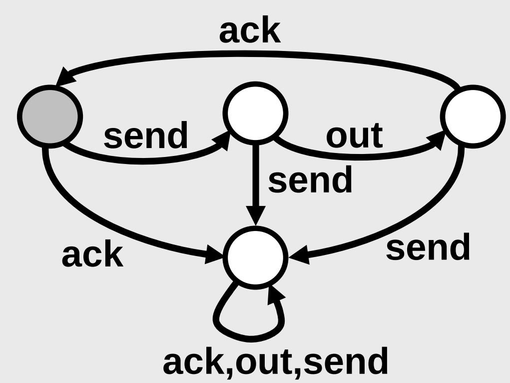
example







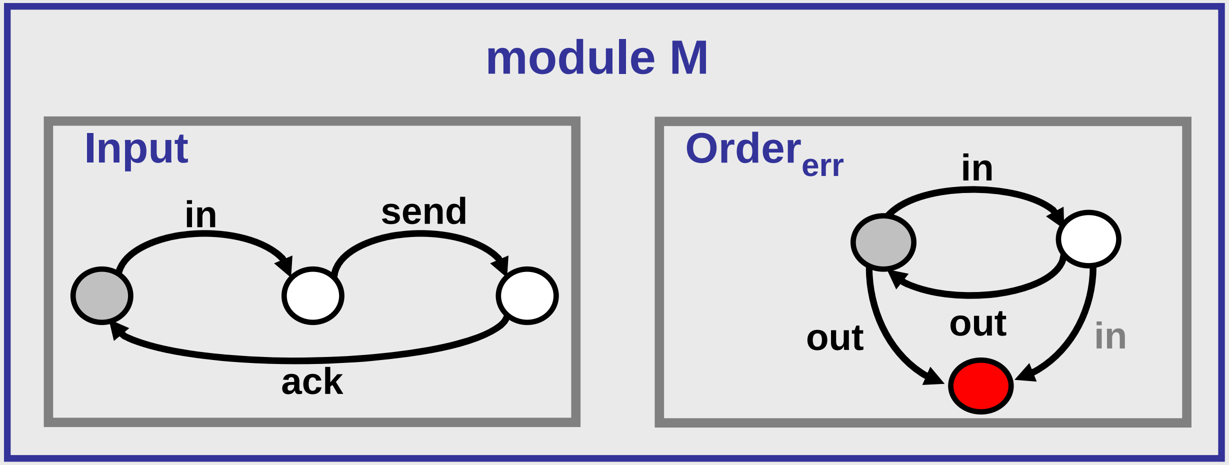
⟨ ack, out ⟩ ?

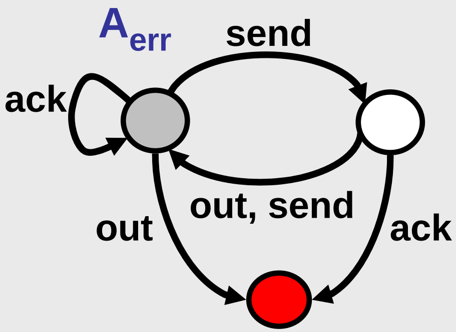


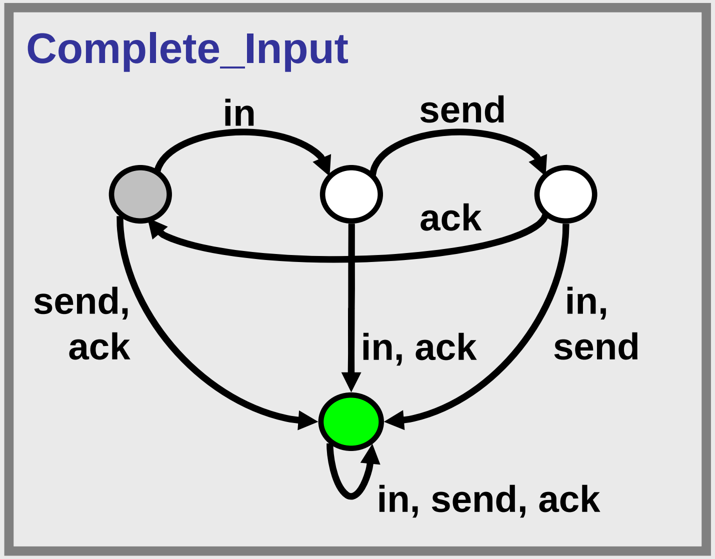
assumption learned for AG reasoning

weakest assumption

complete module for permissiveness check







- 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)

⟨ ack, out ⟩: (sink, error, error)

in summary…

generate precise component interfaces

resolve non-determinism

dynamically & selectively

JavaPathfinder

UML statecharts

assume-guarantee reasoning

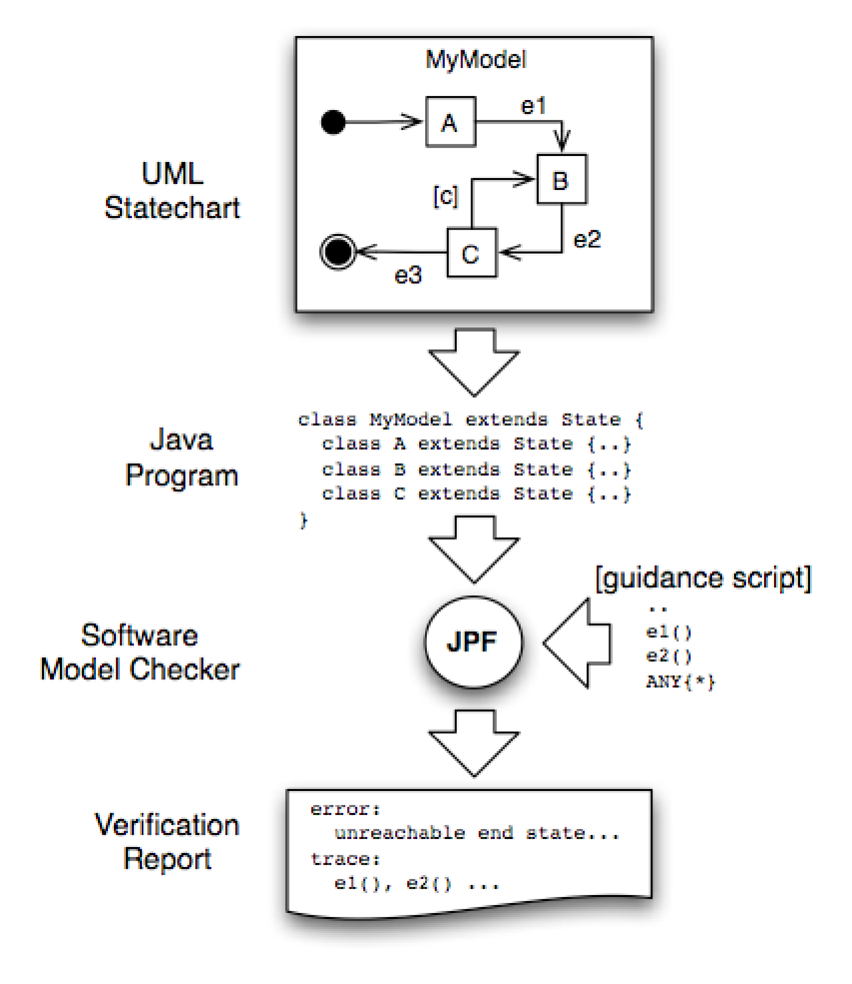
interface generation / discharge

extensions/cv

http://javapathfinder.sourceforge.net

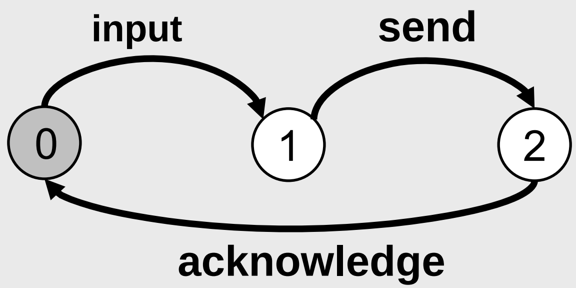
UML framework in JPF

- 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)

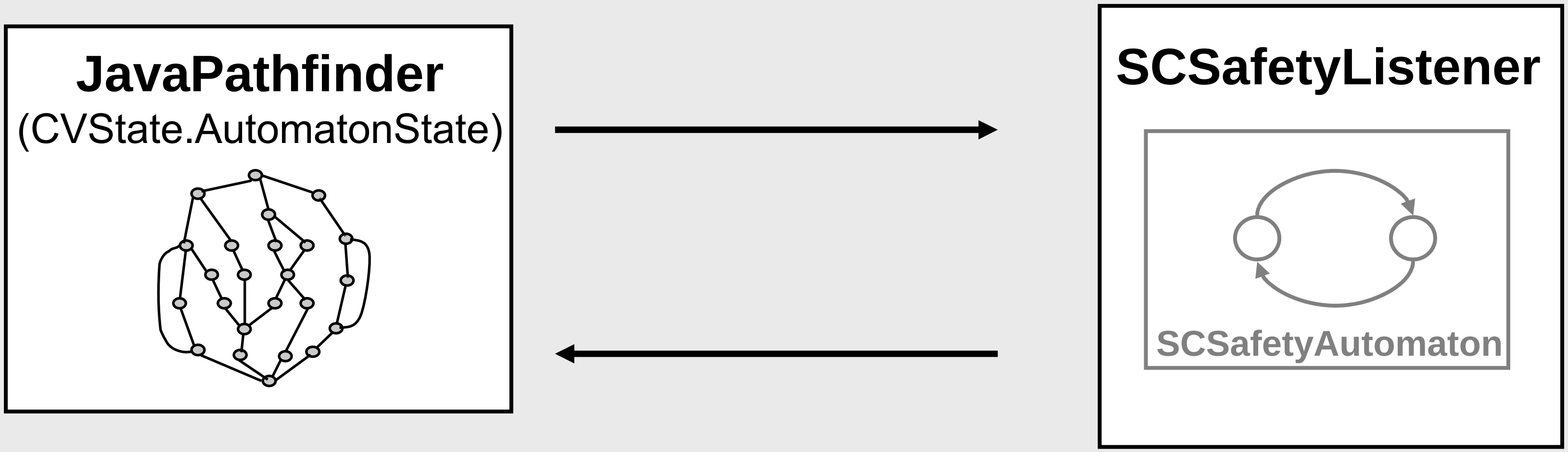


example

```
package ICSETutorial;

import gov.nasa.jpf.sc.State;

public class Input extends State {

  class S0 extends State {

      public void input() {
          setNextState(s1);
      }
  } S0 s0 = makeInitial(new S0());

  class S1 extends State {

      public void send() {
          setNextState(s2);
      }
  } S1 s1 = new S1();

  class S2 extends State {

      public void acknowledge() {
          setNextState(s0);
      }
  } S2 s2 = new S2();
}
```

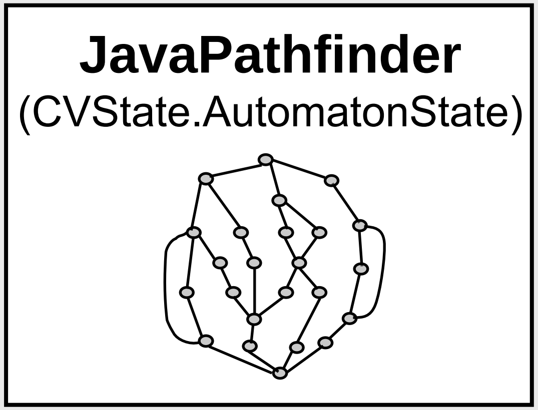


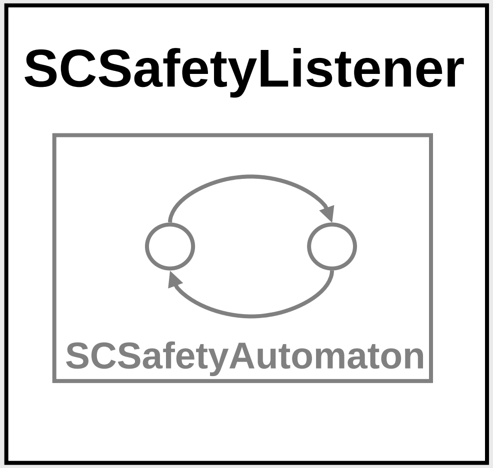
AG reasoning in JPF



assumptions

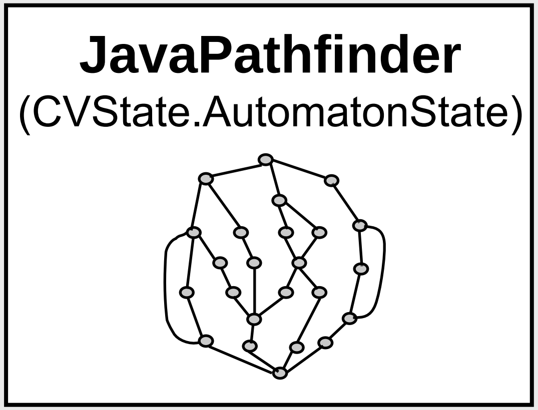
- choiceGeneratorAdvanced
  - if selected action leads assumption to error state then do "vm.getSystemState().setIgnored(true)" (backtrack)
- instructionExecuted
  - advance automaton & set CVState.AutomatonState
- stateBacktracked
  - get CVState.AutomatonState

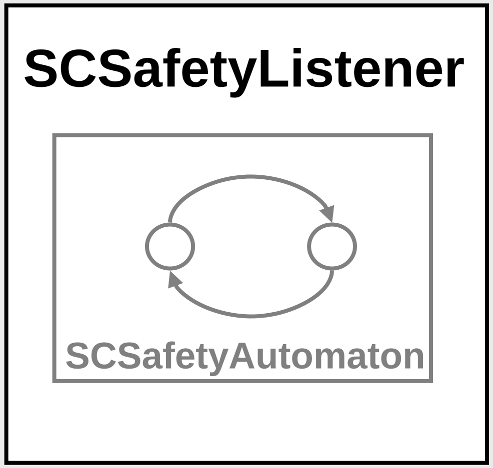




properties

- 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

permissiveness check

```
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.getCounterexample();
    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 output example

Input component with Order Property:

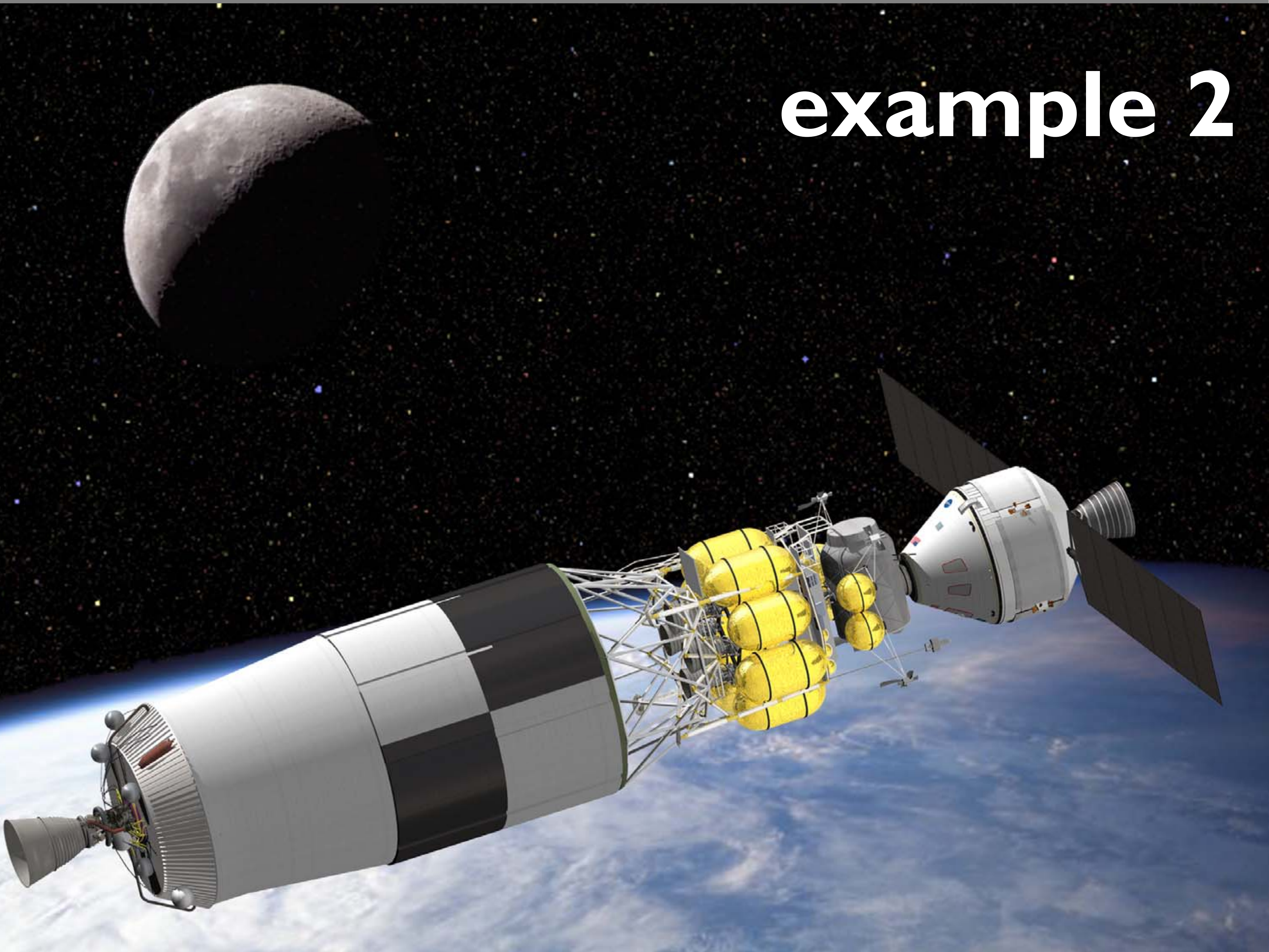
```
package ICSETutorial;

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


public class InputWithProperty
extends CVState {

  class S0 extends State {

      public void input() {
          setNextState(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

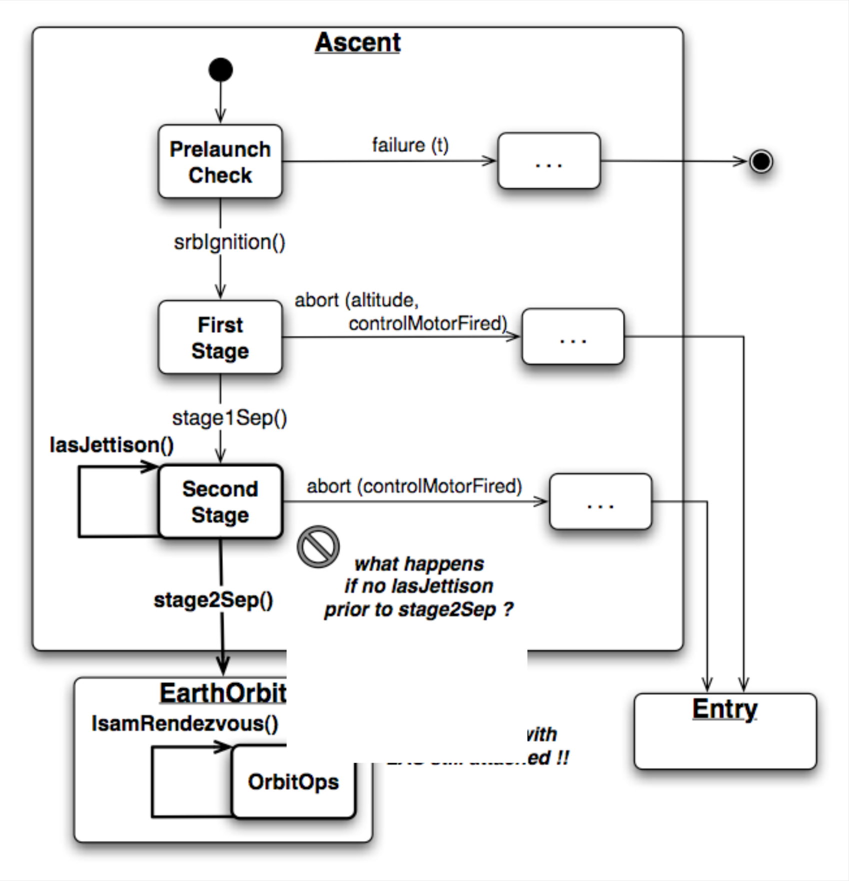
  ICSETutorial.InputWithProperty

```
S0 = ( send -> S2
     | acknowledge -> S1),
S1 = ( output -> S1
     | send -> S1
     | acknowledge -> S1),
S2 = ( output -> S3
     | send -> S1),
S3 = ( send -> S1
     | acknowledge -> S0).
```


example 2

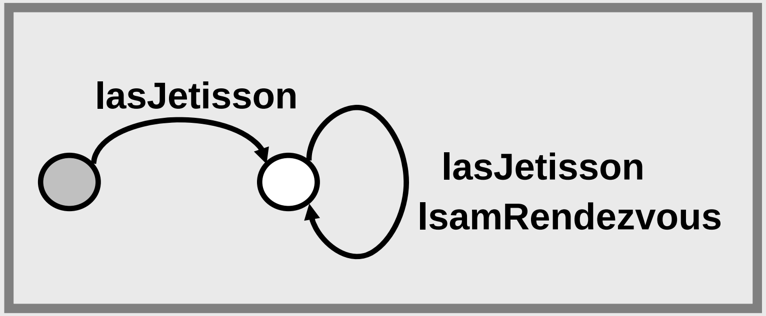
crew exploration vehicle

- model of the *Ascent* and *EarthOrbit* flight phases of a spacecraft
- properties:
  - *“An event lsamRendezvous, 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”*

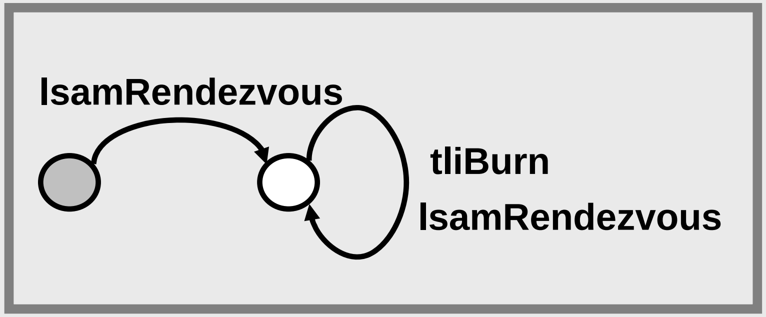


results

Assumption 1:



Assumption 2:



Generated interface assumptions encode Flight Rules in terms of events

JPF run configurations

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

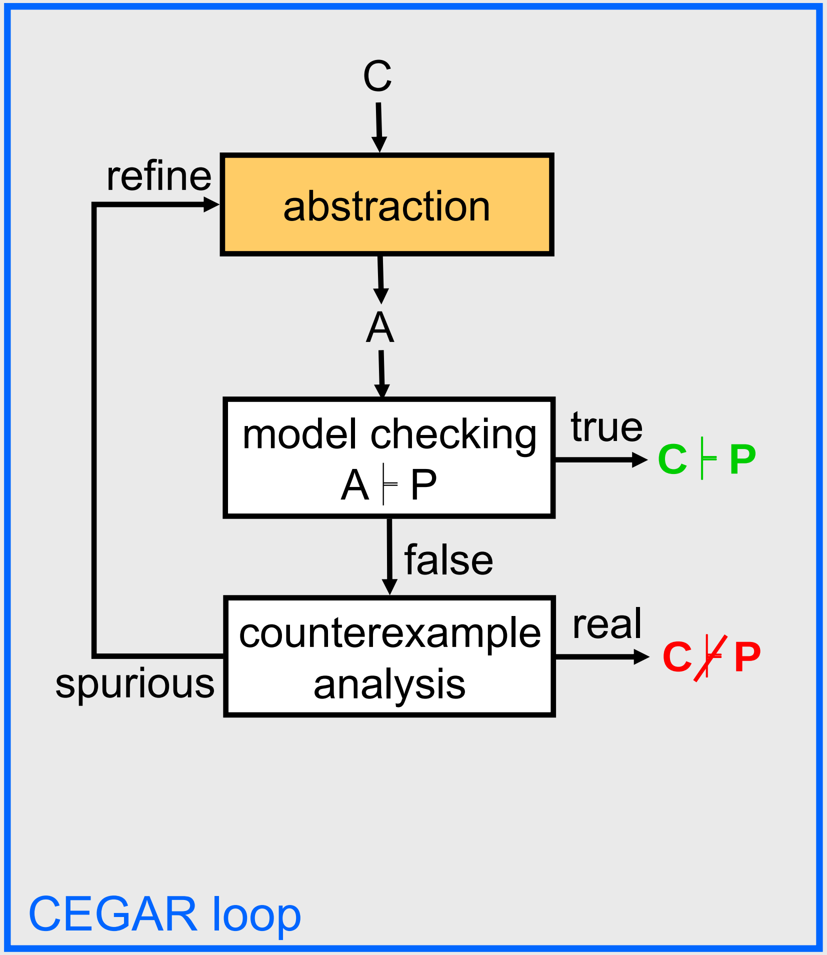
- arguments for property 1:
  +jpf.listener=.tools.ChoiceTracker:.cv.AssertionFilteringListener
  +assertionFilter.include=tliBurn
  +assumption.alphabet=tliBurn,lsamRendezvous
  +assumption.outputFile=examples/jpfESAS/script/generatedAssumption1
  +jpf.report.console.property_violation=error
  +vm.store_steps=true

  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 $C_1 \parallel C_2 \parallel \dots \parallel C_n \models P$
  - build finite-state abstractions: $A_1, A_2, \dots A_n$
  - minimize: $M_1, M_2, \dots M_n$
  - analyze: $M_1 \parallel M_2 \parallel \dots \parallel M_n \models P$?
  - refine based on counterexamples
- permissive interfaces – Henzinger et al. 05
  - uses CEGAR to compute interfaces
- new result at CAV'08

assume-guarantee abstraction refinement (AGAR)

$$\frac{\begin{array}{lccc} 1. & \langle A \rangle & M_1 & \langle P \rangle \\ 2. & \langle \mathit{true} \rangle & M_2 & \langle A \rangle \end{array}}{\langle \mathit{true} \rangle\ M_1 \parallel M_2\ \langle P \rangle}$$

- build A as an abstraction of M_2
- $\langle \mathit{true} \rangle\ M_2\ \langle A \rangle$ holds by construction
- check Premise 1: $\langle A \rangle\ M_1\ \langle P \rangle$
- obtained counterexamples are analyzed and used to refine A
- variant of CEGAR (Counter-example Guided Abstraction Refinement) with differences:
  - use counterexample from one component (M_1) to refine abstraction of the other component (M_2)
  - 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 $L(M_1) \cap L(coP)$
- 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!