Automating Compositional Verification

Dimitra Giannakopoulou, NASA Ames

collaborators

- Corina Păsăreanu (CMU / NASA Ames)

- and talented students / visitors:

  Howard Barringer (Univ. of Manchester)

  Colin Blundell (UPenn)

  Jamieson Cobleigh (UMass, now MathWorks)

  Michael Emmi (UCLA)

  Mihaela Gheorgiu (Univ. of Toronto)

  Chang-Seo Park (UC Berkeley)

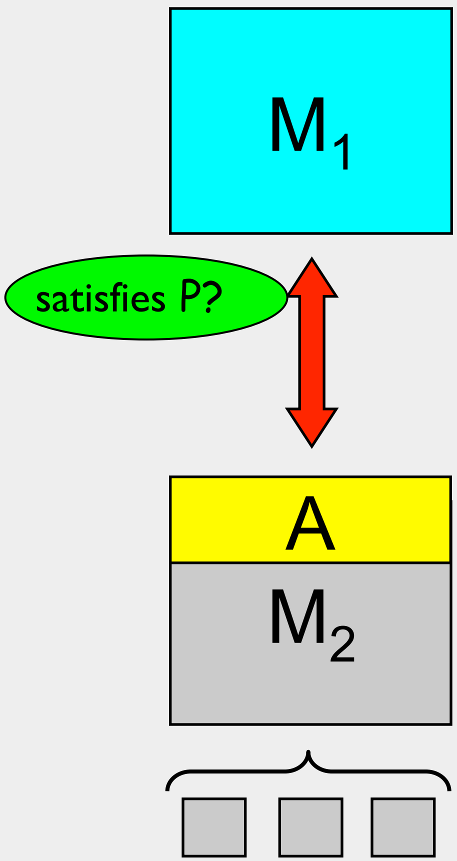
  Suzette Person (Univ. of Nebraska)

  Rishabh Singh (MIT)

state-explosion problem

compositional verification

does system made up of M_1 and M_2 satisfy property P?



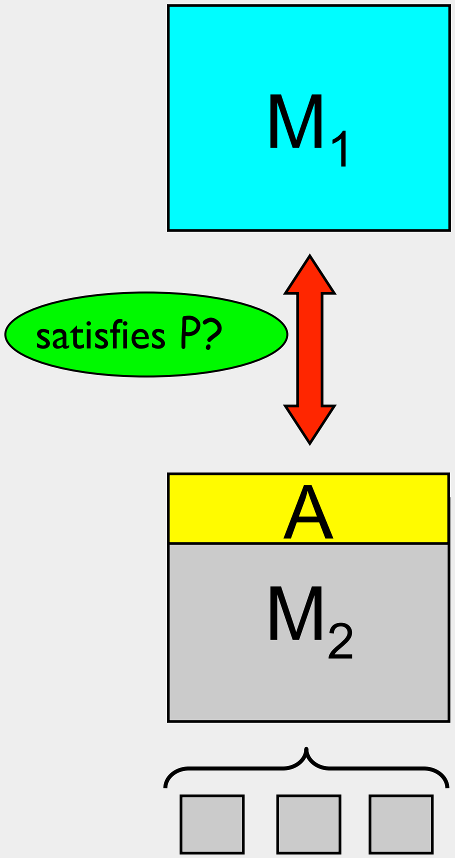
- check P on entire system: too many states!
- use system's natural decomposition into components to break-up the verification task
- check components in isolation:

does M_1 satisfy P?


“when we try to pick out anything by itself, we find it hitched to everything else in the universe”
John Muir

assume-guarantee reasoning

introduces assumptions / 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

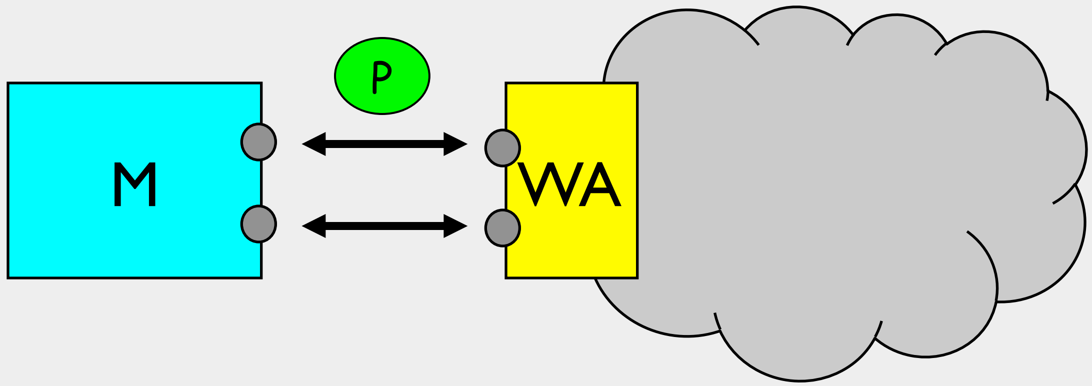
simplest assume-guarantee rule (ASYM):

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

"discharge" the assumption

how do we come up with the assumption?

the weakest assumption [ASE 2002]



- given component M, property P, and the interface ∑ 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⟩

weakest assumption in AG reasoning

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

weakest assumption makes rule complete

for all E, $\langle \mathit{true} \rangle$ M || E $\langle P \rangle$ IFF $\langle \mathit{true} \rangle$ E $\langle WA \rangle$

$\langle \mathit{true} \rangle$ M_1 || M_2 $\langle P \rangle$ IFF $\langle \mathit{true} \rangle$ M_2 $\langle WA \rangle$

in other words:

$\langle \mathit{true} \rangle$ M_2 $\langle WA \rangle$ holds implies $\langle \mathit{true} \rangle$ M_1 || M_2 $\langle P \rangle$ holds

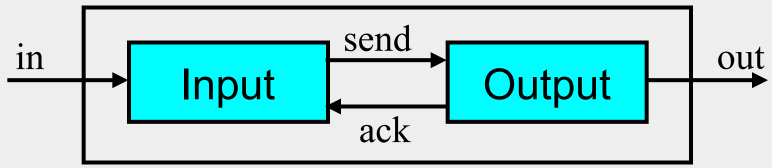
$\langle \mathit{true} \rangle$ M_2 $\langle WA \rangle$ not holds implies $\langle \mathit{true} \rangle$ M_1 || M_2 $\langle P \rangle$ not holds

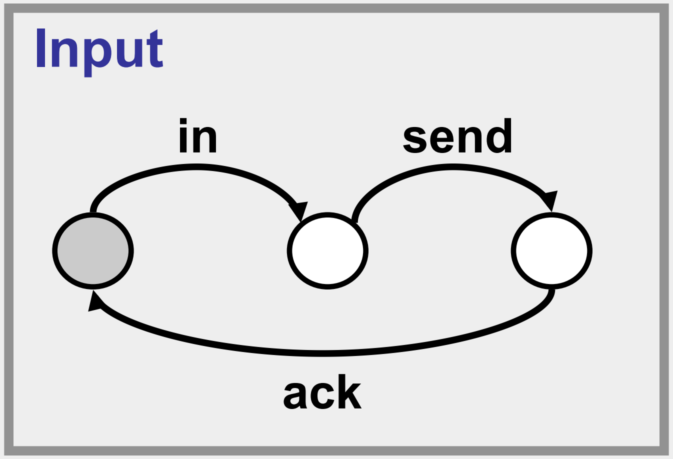
formalisms

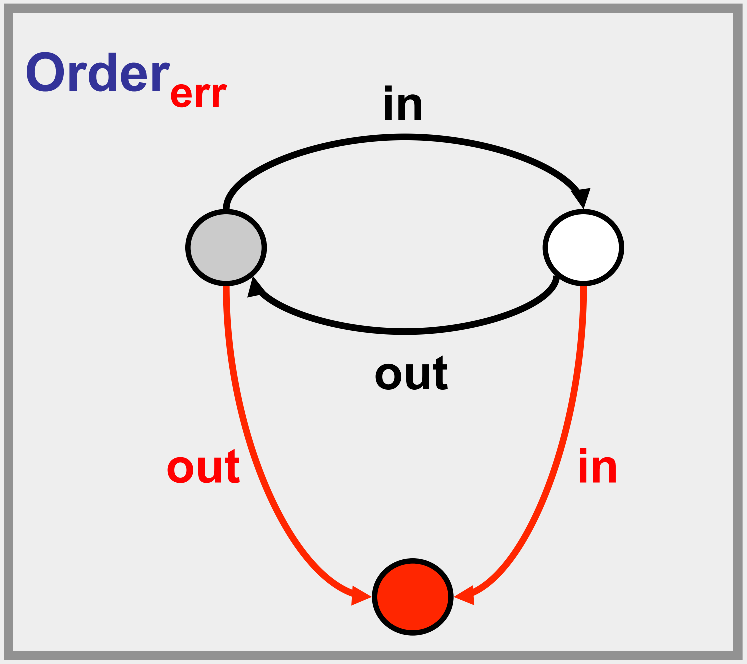
- components modeled as finite state machines (FSM)
  - FSMs assembled with parallel composition operator "||"
    - synchronizes shared actions, interleaves remaining actions
- a safety property P is a FSM
  - P describes all legal behaviors in terms of its alphabet
  - P_{err} – complement of P
    - determinize & complete P with an "error" state;
    - bad behaviors lead to error
  - component M satisfies P iff error state unreachable in ($M \| P_{err}$)
- assume-guarantee reasoning
  - assumptions and guarantees are FSMs
  - $\langle A \rangle M \langle P \rangle$ holds iff error state unreachable in ($A \| M \| P_{err}$)

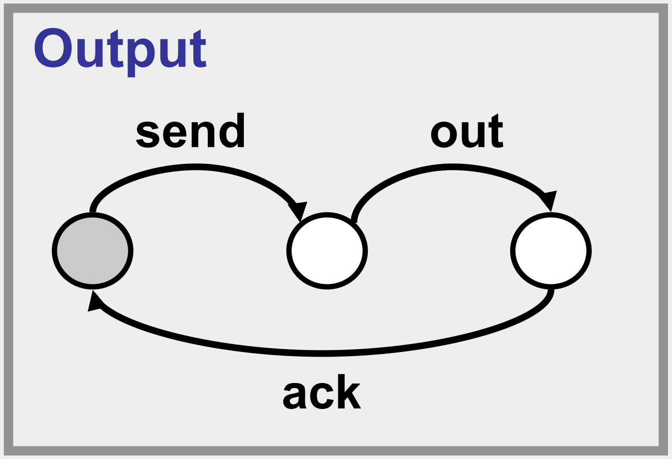
example

require in and out to alternate (property Order)

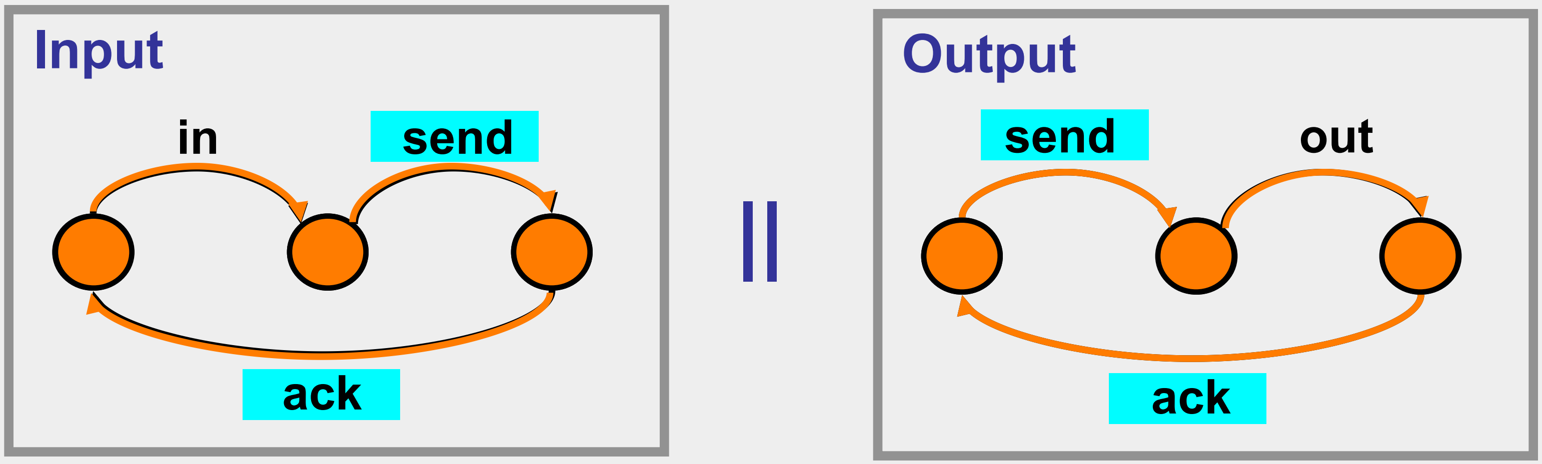




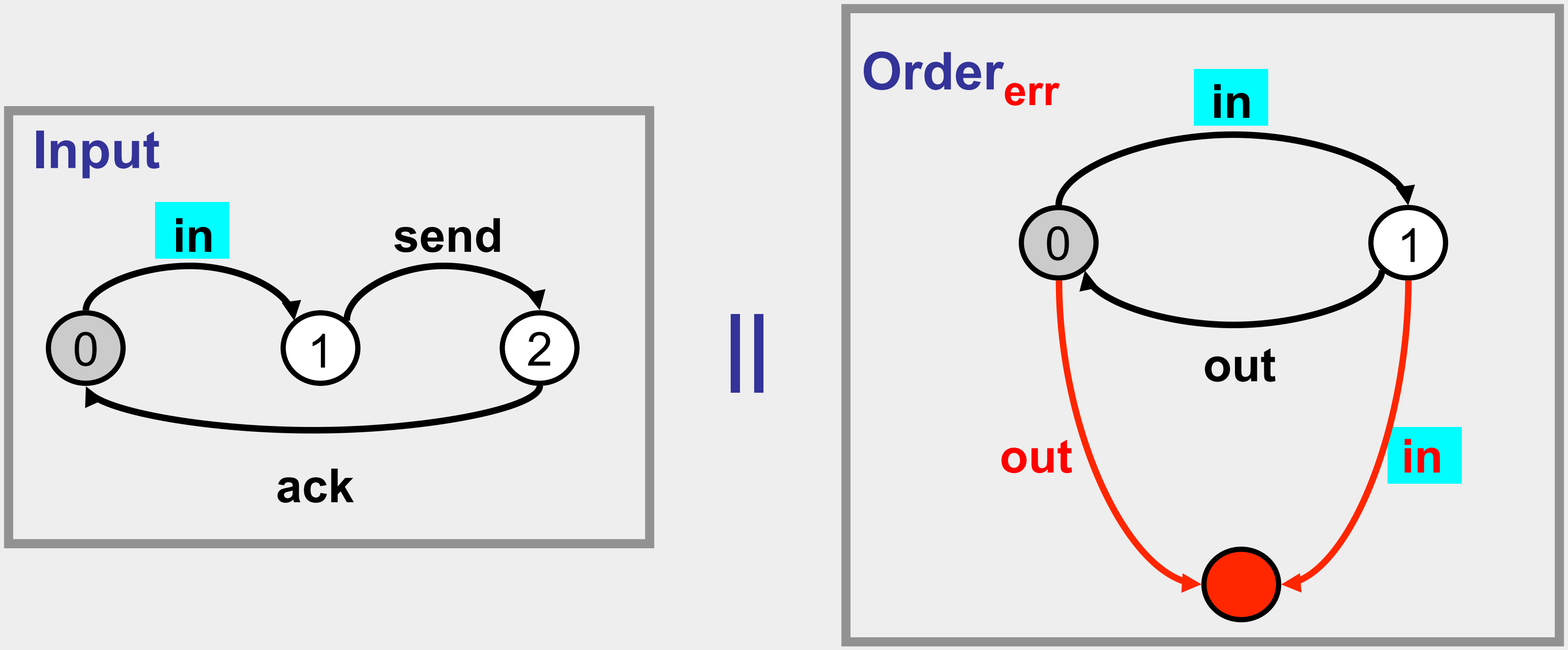




parallel composition



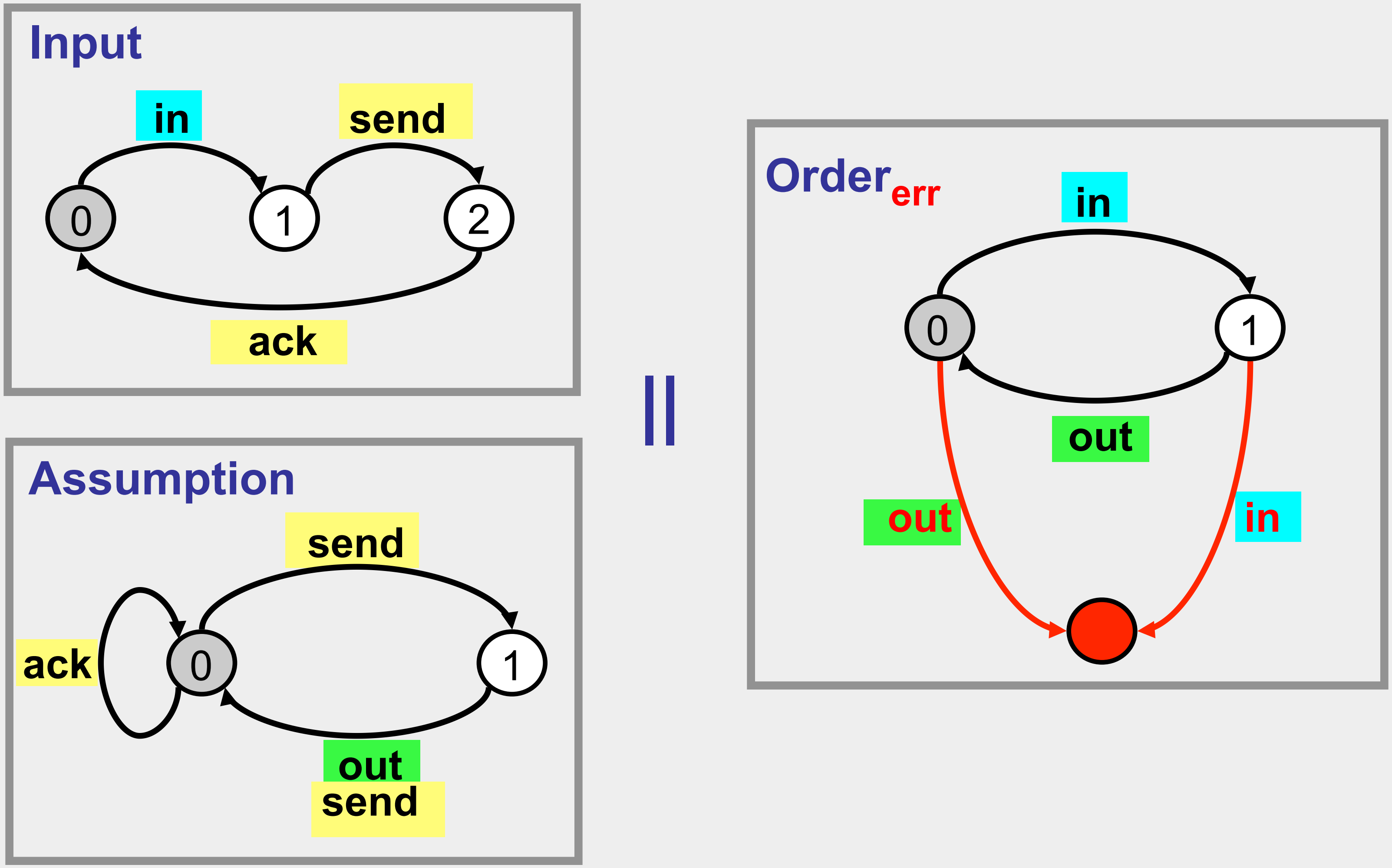
property satisfaction



crex. 1: (I_0, O_0) out (I_0, O_{error})

crex. 2: (I_0, O_0) in (I_1, O_1) send (I_2, O_1) out (I_2, O_0) out (I_2, O_{error})

assume-guarantee reasoning



crex 1: (I_0, A_0, O_0) out ✗

crex 2: (I_0, A_0, O_0) in (I_1, A_0, O_1) send (I_2, A_1, O_1) out (I_2, A_0, O_0) out ✗

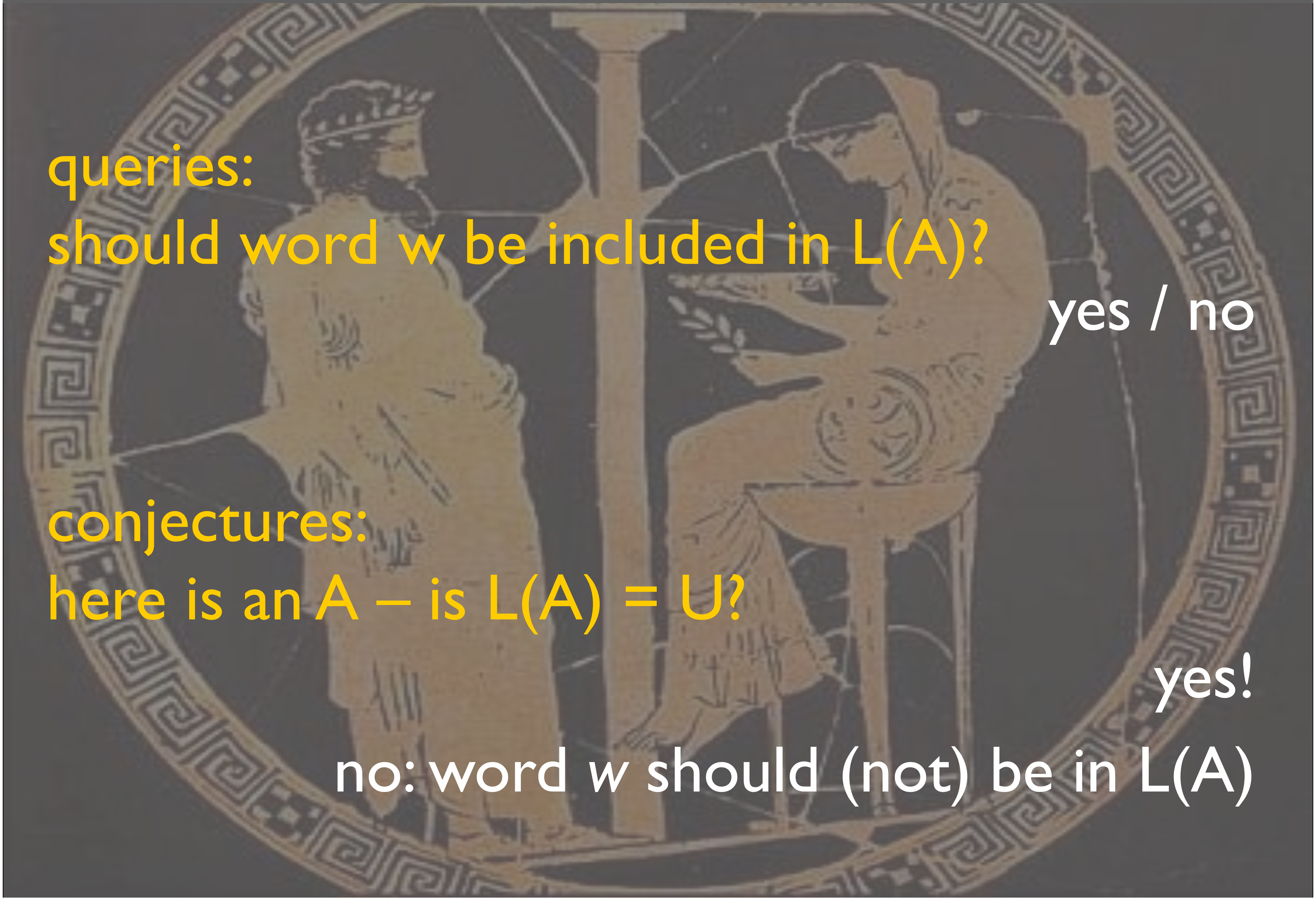
learning assumptions [TACAS 2003]

iterative solution +
intermediate results

L* learns unknown regular language U (over alphabet Σ) and produces minimal DFA A such that L(A) = U

(L originally proposed by Angluin)*

L* learner — the oracle



queries:
should word w be included in L(A)?

yes / no

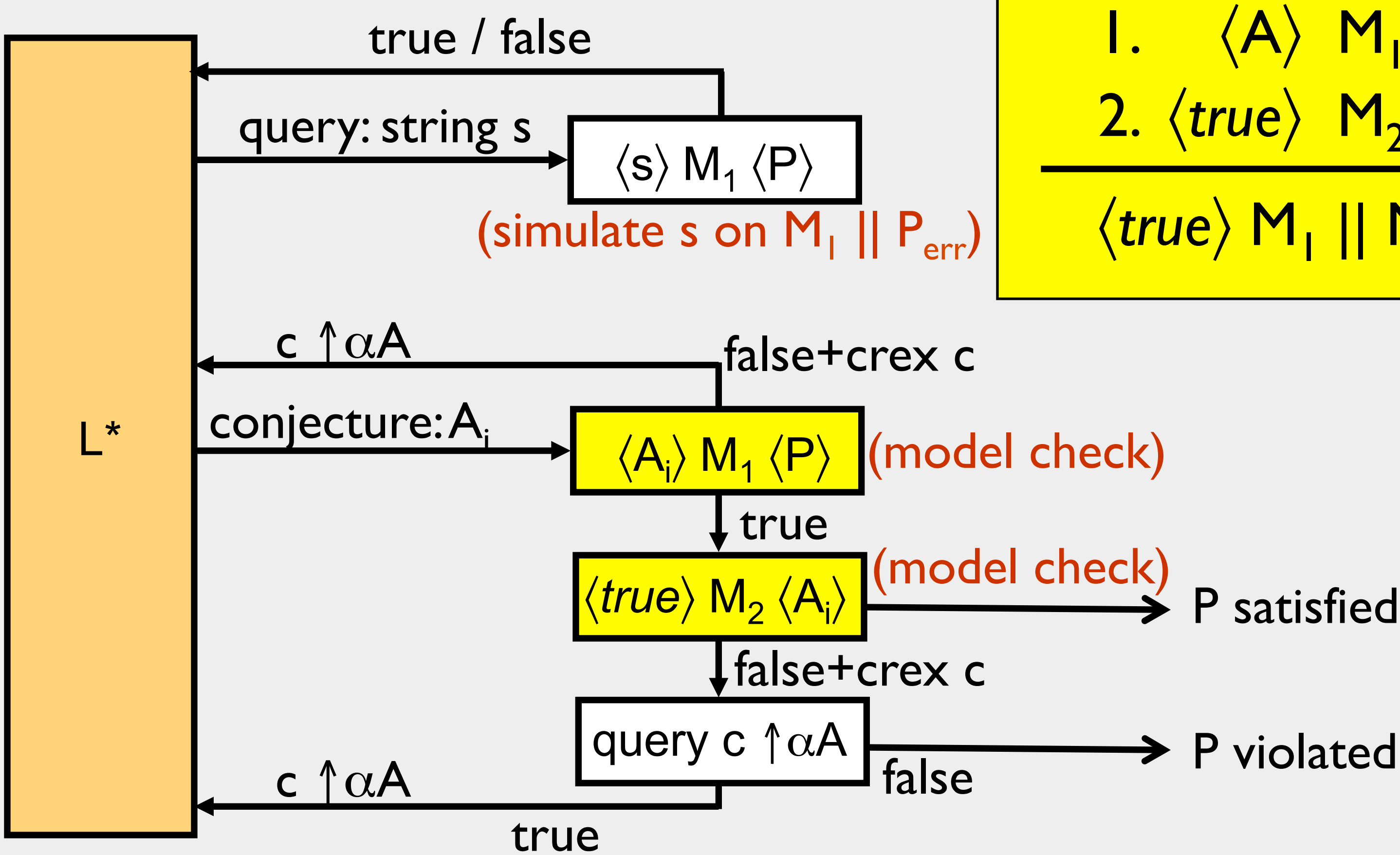
conjectures:
here is an A – is L(A) = U?

yes!

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

oracle for WA in assume-guarantee reasoning

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



$\langle WA\rangle\ M_1\ \langle P\rangle$ holds

$\langle \mathit{true}\rangle\ M_2\ \langle WA\rangle$ holds implies $\langle \mathit{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle$ holds

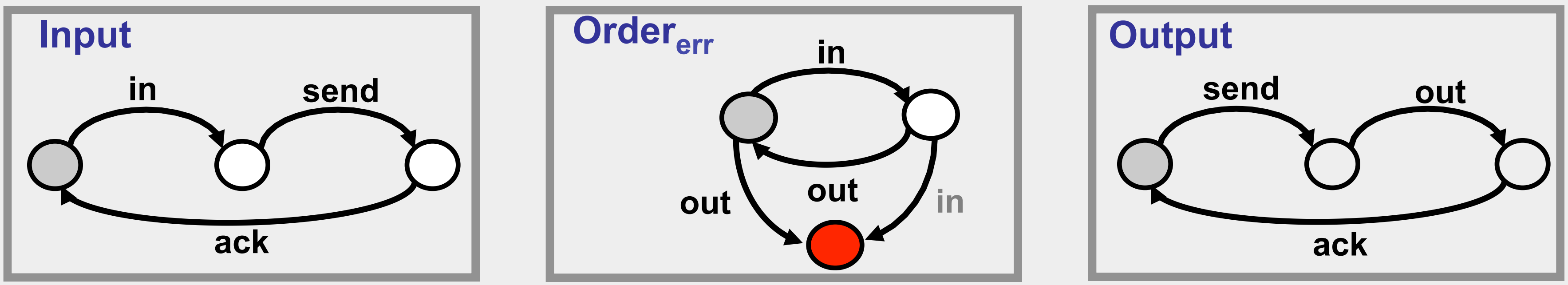
$\langle \mathit{true}\rangle\ M_2\ \langle WA\rangle$ does not hold implies $\langle \mathit{true}\rangle\ M_1 \parallel M_2\ \langle P\rangle$ does not hold

characteristics

assumptions conjectured by L* are not comparable semantically

- terminates with *minimal* automaton A for U
- generates DFA candidates A_i: $|A_1| < |A_2| < \ldots < |A|$
- produces at most n candidates, where $n = |A|$
- # queries: $O(kn^2 + n \log m)$,
  - m is size of largest counterexample, k is size of alphabet
- for assume-guarantee reasoning, may terminate early with a smaller assumption than the weakest

example

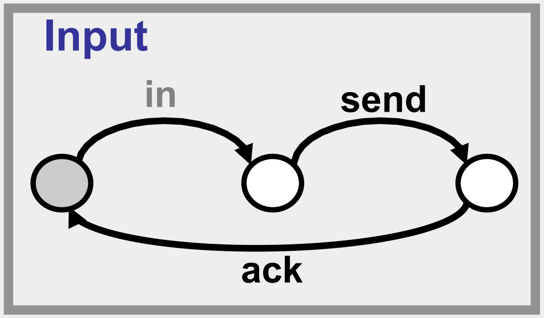


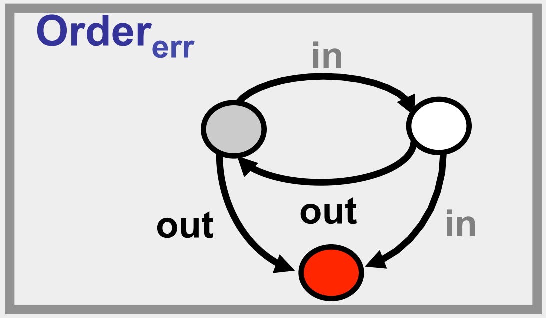
we check: ⟨true⟩ Input || Output ⟨Order⟩

M_1 = Input, M_2 = Output, P = Order

assumption alphabet: {send, out, ack}

queries

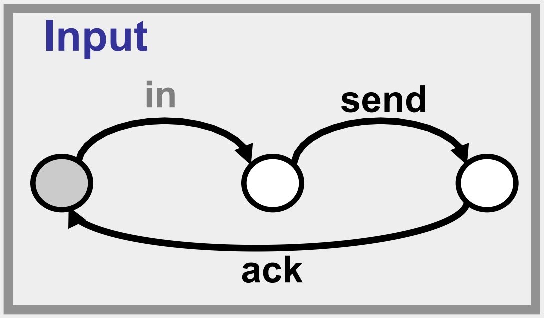


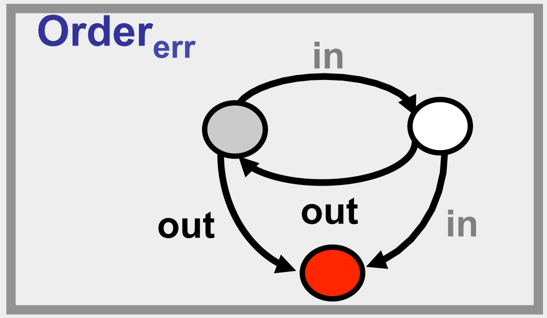


		E
	Table T	λ
S	λ	true
	out	false
S · Σ	ack	true
	out	false
	send	true
	out, ack	false
	out, out	false
	out, send	false

S = set of prefixes
E = set of suffixes

candidate construction

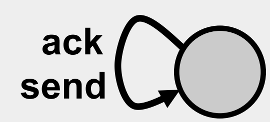




		E
	Table T	λ
S	λ	true
	out	false
S · Σ	ack	true
	out	false
	send	true
	out, ack	false
	out, out	false
	out, send	false

2 states – error state omitted

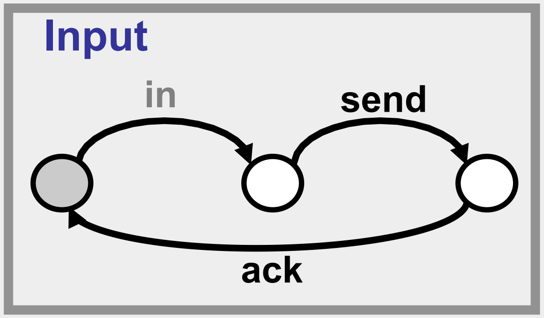
Assumption A_1

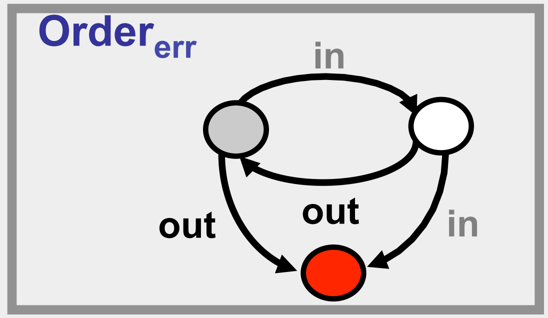


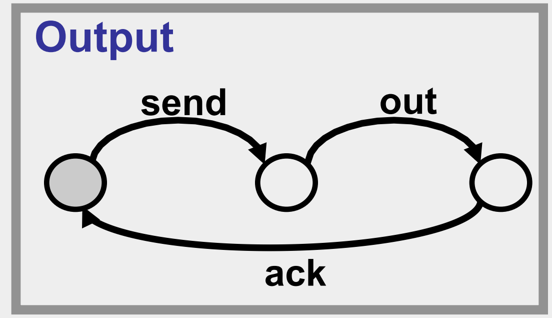
counterexamples add to *S*

S = set of prefixes
E = set of suffixes

conjectures







A_1:

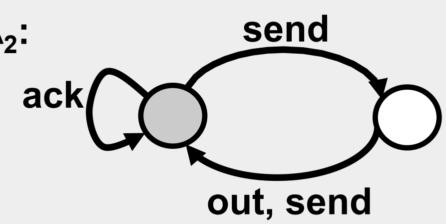
Oracle 1:
⟨A_1⟩ Input ⟨Order⟩

Counterexample:
c = ⟨in,send,ack,in⟩

Return to L*:
c↑ Σ = ⟨send,ack⟩

Queries

A_2:



Oracle 1:
⟨A_2⟩ Input ⟨Order⟩
True

Oracle 2:
⟨true⟩ Output ⟨A_2⟩
True

property Order holds
on Input || Output

more than 2 components [TACAS03, FMSD09]

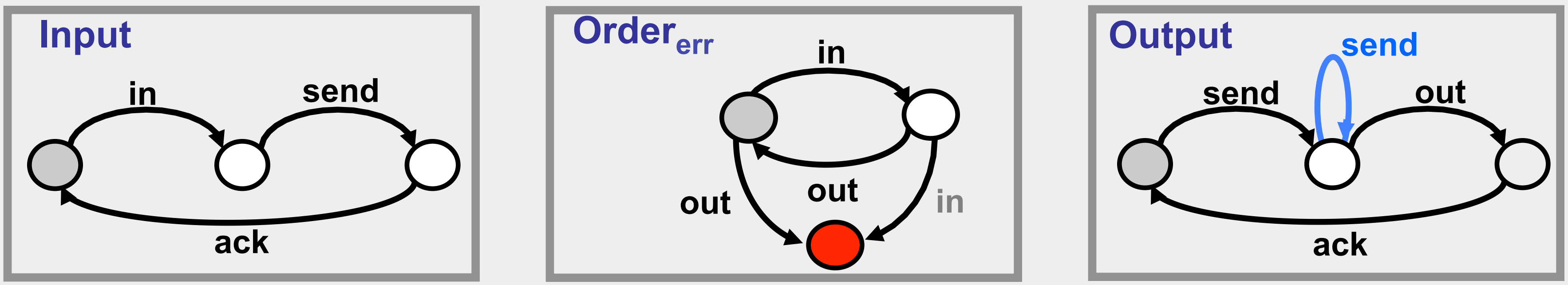
1. $\langle A_1 \rangle\ M_1\ \langle P \rangle$
2. $\langle \mathit{true} \rangle\ M_2 \parallel M_3\ \langle A_1 \rangle$

$$\langle \mathit{true} \rangle\ M_1 \parallel (M_2 \parallel M_3)\ \langle P \rangle$$

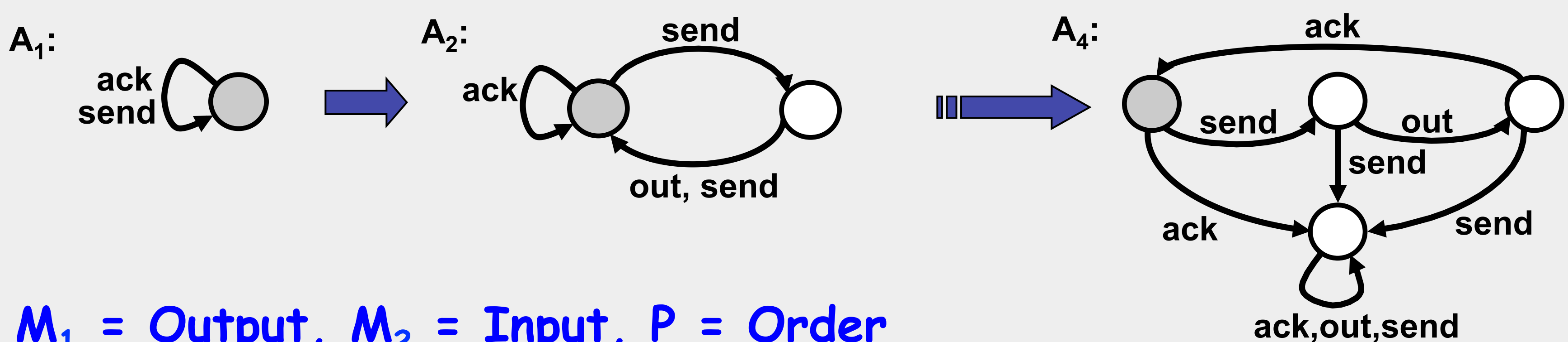
1. $\langle A_2 \rangle\ M_2\ \langle A_1 \rangle$
2. $\langle \mathit{true} \rangle\ M_3\ \langle A_2 \rangle$

$$\langle \mathit{true} \rangle\ M_2 \parallel M_3\ \langle A_1 \rangle$$

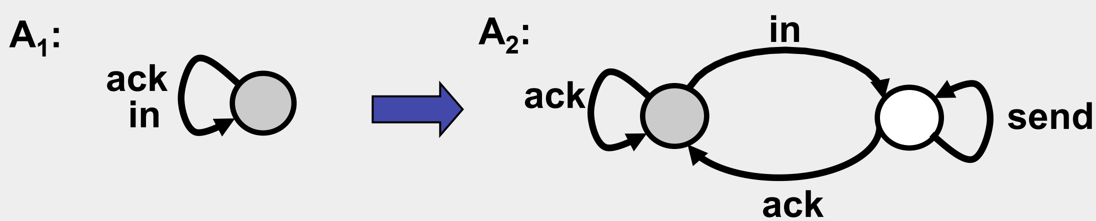
symmetric rules: motivation



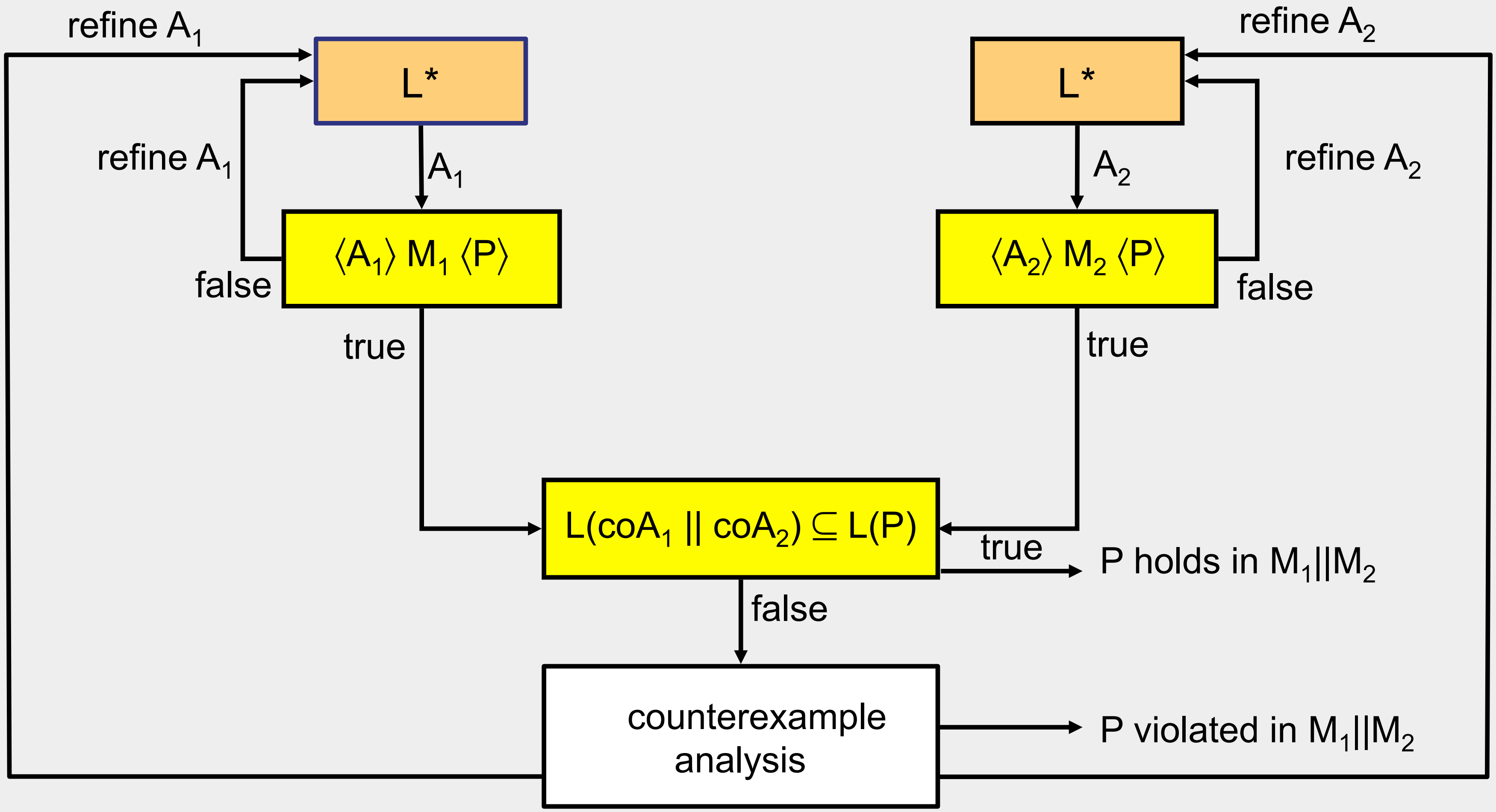
M_1 = Input, M_2 = Output, P = Order



M_1 = Output, M_2 = Input, P = Order



symmetric learning framework [SAVCBS05]



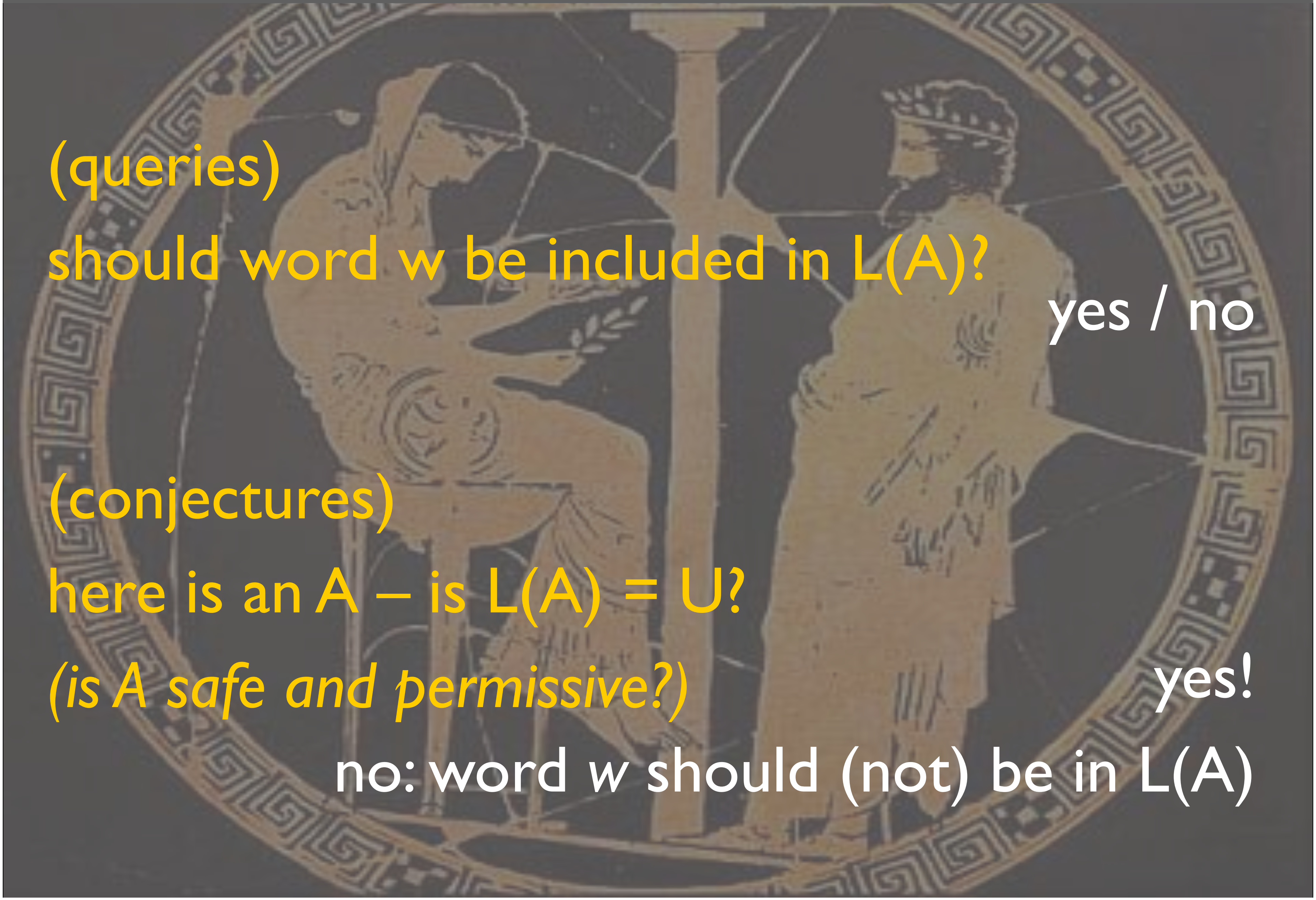
interfaces

- beyond syntactic interfaces (*open* file before *close*)
- document implicit assumptions

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

- safe & permissive interface = weakest assumption

L* learner | the oracle



(queries)

should word w be included in L(A)?

yes / no

(conjectures)

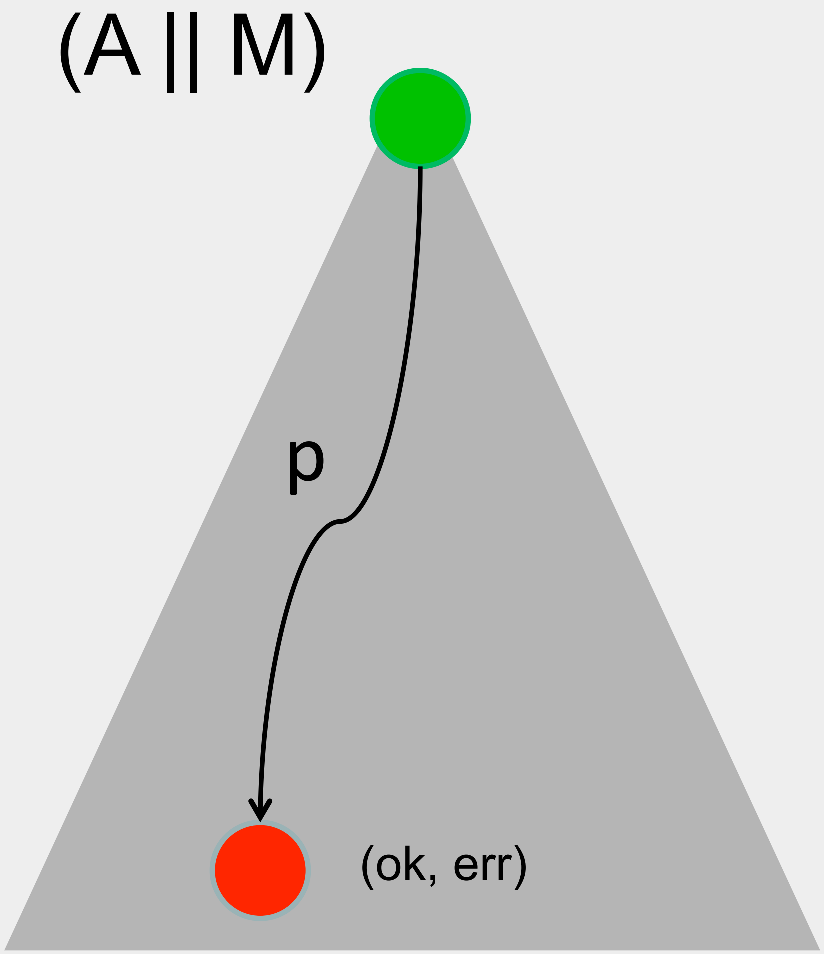
here is an A – is L(A) = U?

(is A safe and permissive?)

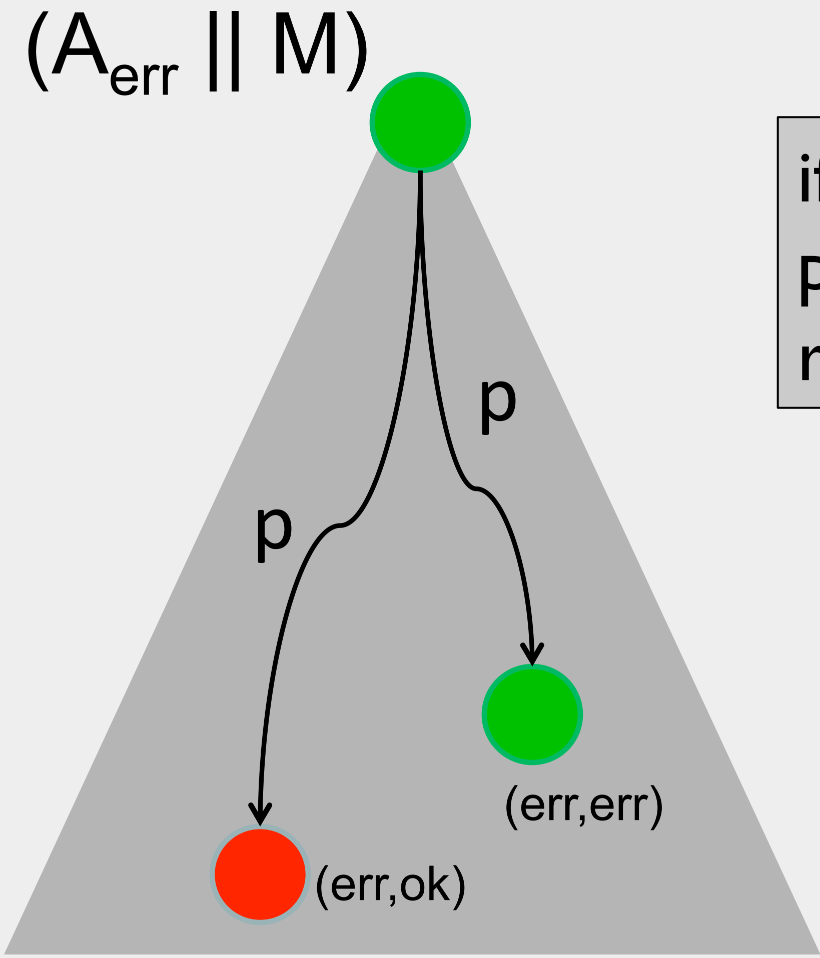
yes!

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

checkSafe(interface A, FSM M)



checkPermissive(interface A, FSM M)



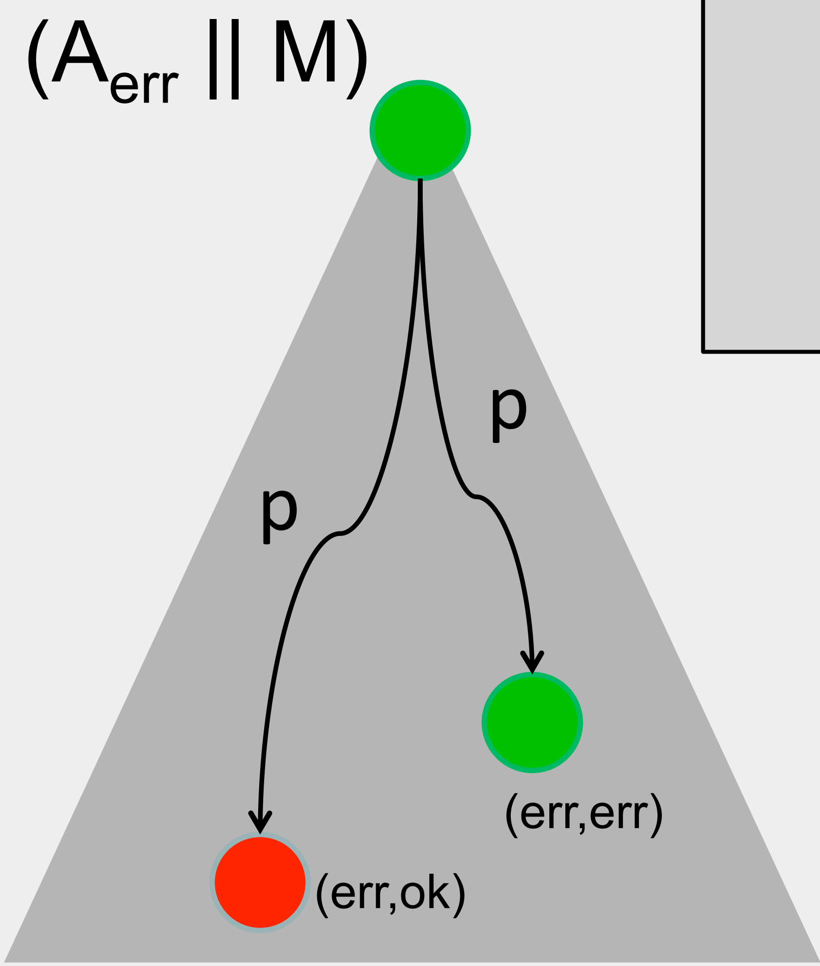
if M is non-deterministic, permissiveness check requires subset construction

ASE 2002

Alur et al, 2005

Henzinger et al, 2005

permissiveness heuristics [FASE 2009]



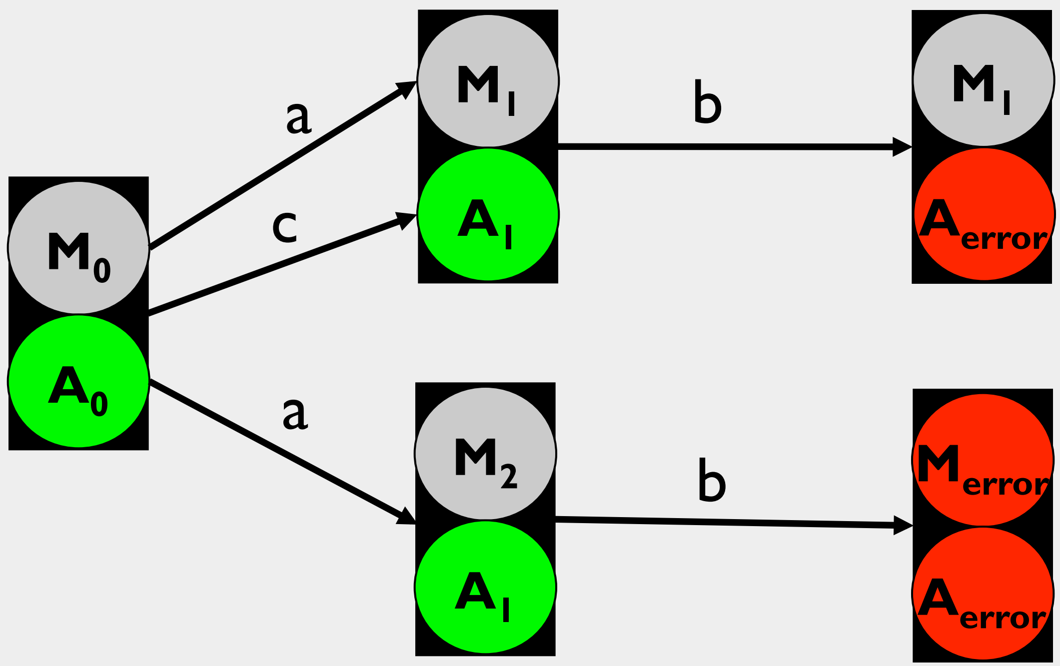
model check for (err, ok)

- reached (err, ok) by "p"
- query "p"
- no ("p" should not be in A)
- backtrack & continue search…

resolves non-determinism

dynamically & selectively;

remember, it's a heuristic



JavaPathfinder

UML statecharts

assume-guarantee reasoning

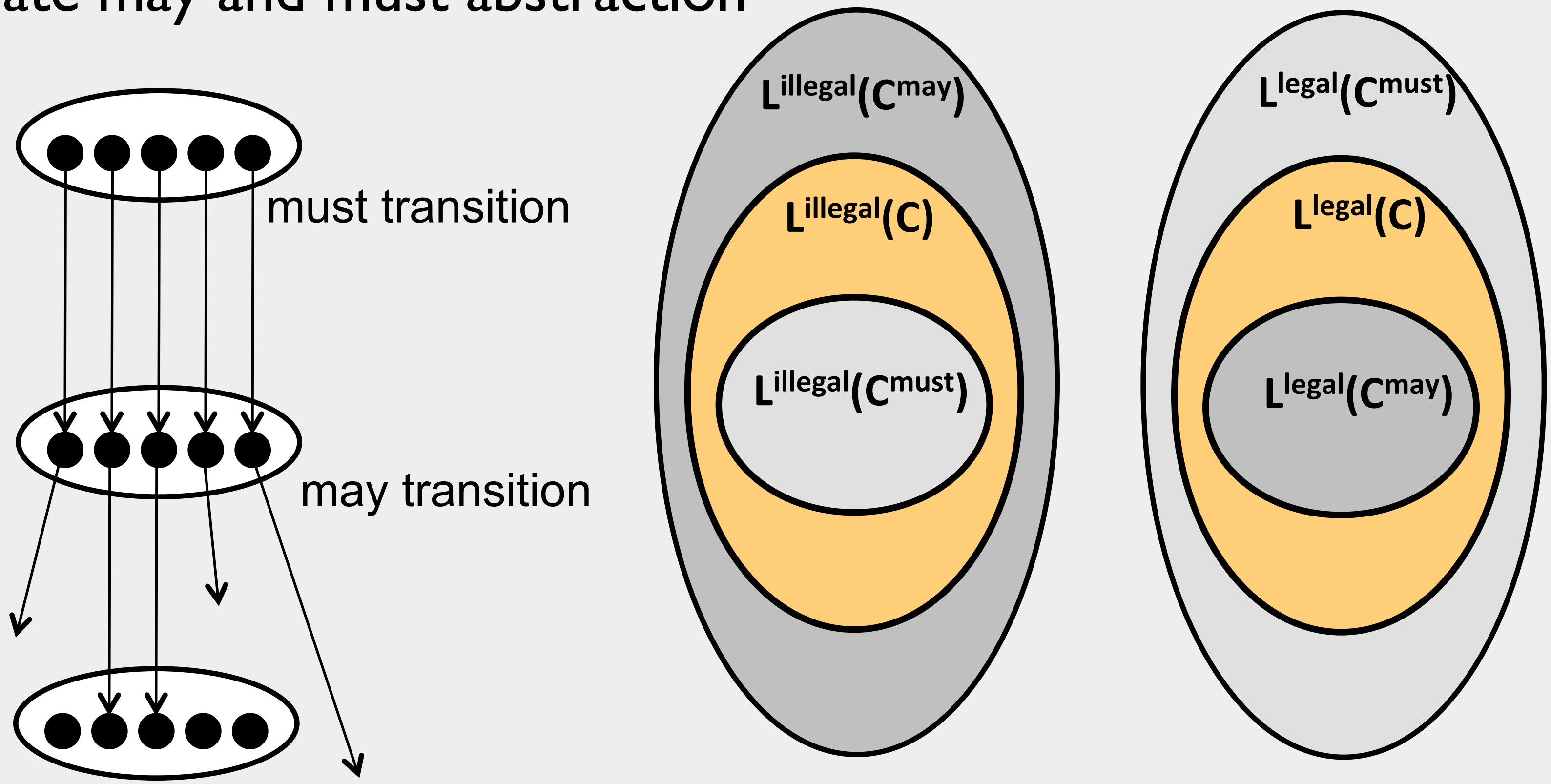
interface generation / discharge

jpf-cv

http://babelfish.arc.nasa.gov/trac/jpf

infinite components [CAV 2010]

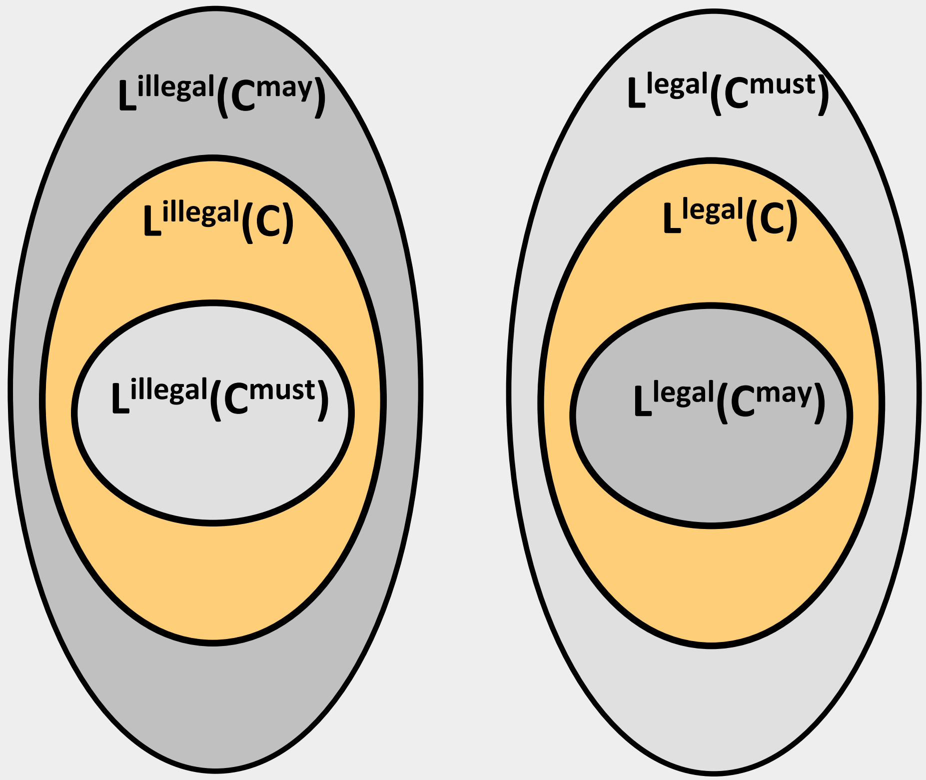
- use predicate abstraction (e.g., x ≥ 0, x < 0)
- generate may and must abstraction



an interface safe w.r.t. C^{may} and permissive w.r.t. C^{must}
is safe and permissive w.r.t. concrete component C

Query(σ, C)

if checkSafe(σ,C^{must}) != null
    return "no"
cex = checkSafe(σ,C^{may})
if cex == null
    return "yes"
Preds = Preds U Refine(cex)
Query(σ, C)



If concrete component is deterministic, so is the must abstraction…

ARMC model checker: Java2SDK library classes, OpenSSL, NASA CEV model

related work

- assume-guarantee reasoning for code (ICSE 2004, SAVCBS 2005, IET Software 2009)
- learning with alphabet refinement (TACAS 2007; also Chaki et al.)
- learning assumptions for interface automata (FM 2008)
- assume-guarantee abstraction refinement (CAV 2008)

- compositional verification in symbolic setting (Alur et al. 05)
- minimal assumptions as separating automata for languages $L(M_2)$ and $L(M_1) \cap L(coP)$ (Gupta et al. 07, Chen et al. 09)
- learning omega-regular languages for liveness (Farzan et al. 08)
- learning non-deterministic automata (Bollig et al. 09)
- learning Boolean functions (Chen et al. 10)
- assumption generation in probabilistic setting (Feng et al. 10)

summary and food for thought…

- techniques are generic
- better applied at design level
- not a panacea…
  - perform well when alphabets & assumptions are small

- what makes a system amenable to compositional techniques?
- design for compositional verification; combine with other design approaches
- how can we make it practical for real systems? what types of interfaces are useful in practice?
- discovering good system decompositions
- liveness, timed & probabilistic systems, non functional properties
- multi core / parallelization?

thank you!

invoke a model checker

within a model checker?

permissiveness check

MC: model check for (M_i, A_{error})

reached (err, ok) by trace t

if (memoized(t) == no) *// t is spurious*

backtrack and continue search

else *// memoized(t) == yes or t not in memoized*

model checker produces t

if (query(t) == yes)

return t to L* *// not permissive*

else restart at MC

conjecture : Oracle 1

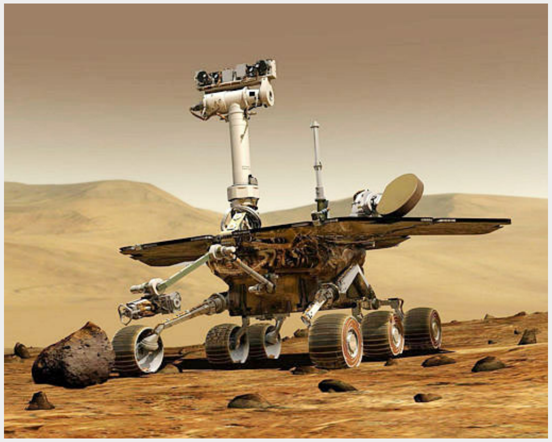
1. cex = checkSafe(A, C^{may})
2. if cex == null
3. invoke Oracle2
4. If Query(cex, C) == “false”
5. return cex to L*
6. else
7. goto 1

conjecture : Oracle 2

1. cex = checkPermissive(A, C^{must})
2. if cex == null
3.   return A
4. If Query(cex, C) == “true”
5.   return cex to L*
6. else
7. goto 1

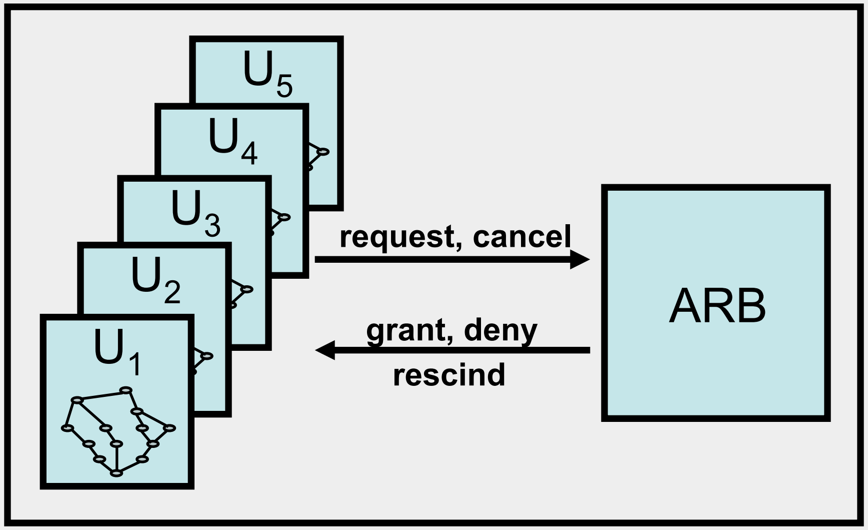
example 1: Mars Exploration Rover

- tools: LTSA, SPIN
- model derived from JPL's Mars Exploration Rover (MER) Resource Arbiter
  - local management of resource contention between resource consumers (e.g. science instruments, communication systems)
  - consists of k user threads and one server thread (arbiter)



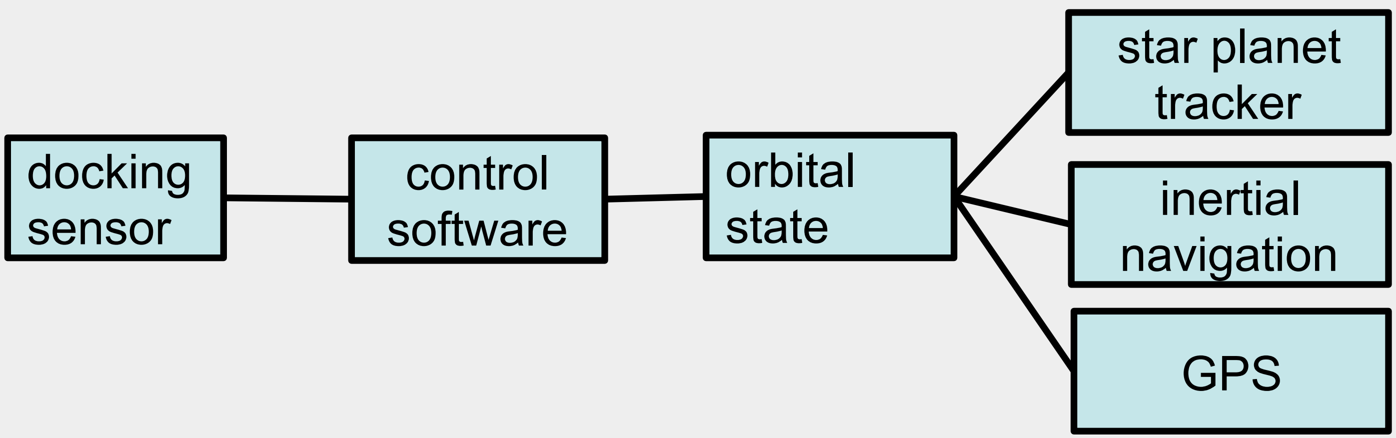
- checked mutual exclusion between resources (e.g. driving while capturing a camera image are incompatible)
- compositional verification scaled to >5 users vs. monolithic verification ran out of memory [SPIN'06]

Resource Arbiter



example 2: autonomous rendezvous & docking

- tool: LTSA
- consists of control software, state estimator, and 4 types of sensors
- input provided as UML state-charts, properties of type:
  - "you need at least two operational sensors to proceed to next mode"
- 3 bugs detected
- scaling achieved with compositional verification:
  - monolithic verification runs out of memory after > 13M states
  - compositional verification terminates successfully in secs. Largest state-space explored is less than 60K states, as opposed to > 13M.



example 3: K9 Rover Executive

- tools: LTSA, JavaPathfinder
- model of NASA Ames K9 Rover Executive
  - executes flexible plans for autonomy
  - consists of Executive thread and ExecCondChecker thread for monitoring state conditions
  - checked for specific shared variable: if Executive reads its value, ExecCondChecker should not read the variable before the Executive clears it
- generated assumption of 6 states for model in LTSA [TACAS 2003]
- used generated assumption to check 8K lines of JAVA code translated from 10K lines of C++ code using the JavaPathfinder model checker [ICSE 2004]
- reduced memory used by JavaPathfinder > 3 times
- used generated assumption to perform assume-guarantee testing of C++ code using Eagle runtime monitoring framework [SAVCBS 2005, IET Software 2009]