Automated Compositional Verification part 1

Corina Păsăreanu
CMU Silicon Valley/ NASA Ames Research Center

outline

Part I

- assume-guarantee reasoning
- computing assumptions
- learning assumptions
- multiple components
- different assume-guarantee rules

Part II

- alphabet refinement
- assume-guarantee abstraction refinement
- reasoning about code
- related work
- conclusion

thanks

Dimitra Giannakopoulou and … many collaborators

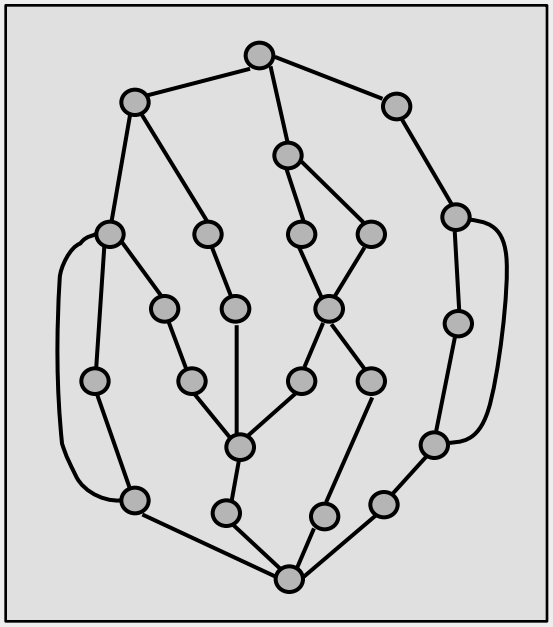
model checking

program / model

```
void add(Object o) {
 buffer[head] = o;
 head = (head+1)%size;
}

Object take() {
 …
 tail=(tail+1)%size;
 return buffer[tail];
}
```

model checker



property

always(ϕ **or** ψ)

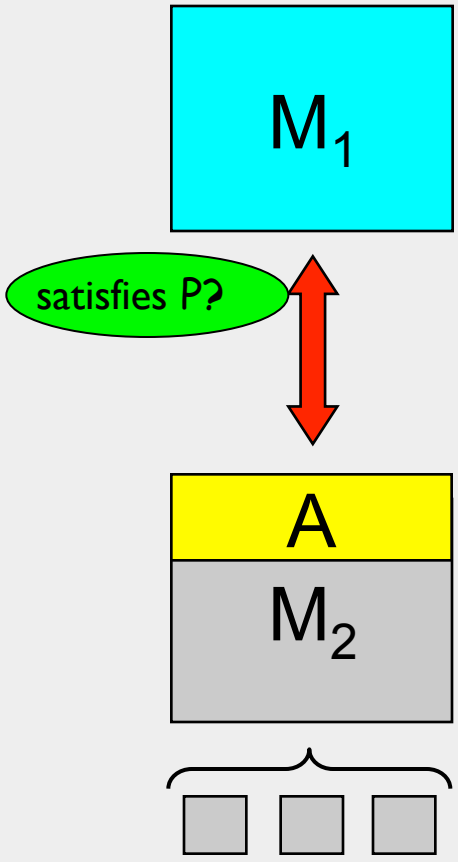
YES (property holds)

NO + counterexample:

```
Line 5: …
Line 12: …
…
Line 41:…
Line 47:…
```

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:

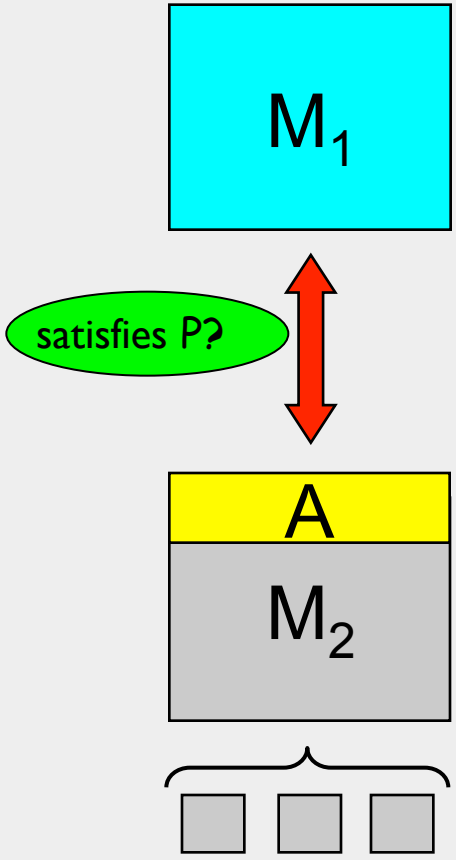
  Does M_1 satisfy P?

  – typically a component is designed to satisfy its requirements in specific contexts
- Assume-guarantee reasoning

  [Misra&Chandy 81, Jones 83, Pnueli 84]

  – introduces assumption A representing M_1's context

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):

$$\frac{1.\ \ \langle A \rangle\ M_1\ \langle P \rangle \qquad 2.\ \langle \mathit{true} \rangle\ M_2\ \langle A \rangle}{\langle \mathit{true} \rangle\ M_1 \,||\, M_2\ \langle P \rangle}$$

"discharge" the assumption

examples of assumptions

- no file “close” before “open”
- accesses to shared variable “X” must be protected by lock “L”
- (rover executive) whenever thread “A” reads variable “V”, no other thread can read “V” before thread “A” clears it first
- (spacecraft flight phases) a docking maneuver can only be invoked if the launch abort system has previously been jettisoned from the spacecraft

assume-guarantee reasoning

how do we compute assumptions?

formalisms

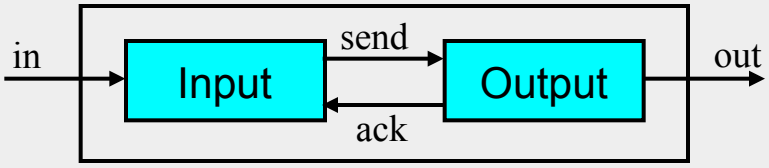
- components modeled as finite state machines (FSM)
  - FSMs assembled with parallel composition operator "||"
  - synchronizes shared actions, interleaves remaining actions
- alphabet of M: αM
- language of M: $\mathcal{L}(M)$
  - the set of all traces (with τ removed) in M
- projection
  - $t{\uparrow}\,\Sigma$ – the trace obtained from t by removing all actions not in Σ
  - $M{\uparrow}\,\Sigma$ – replace with τ all actions not in Σ

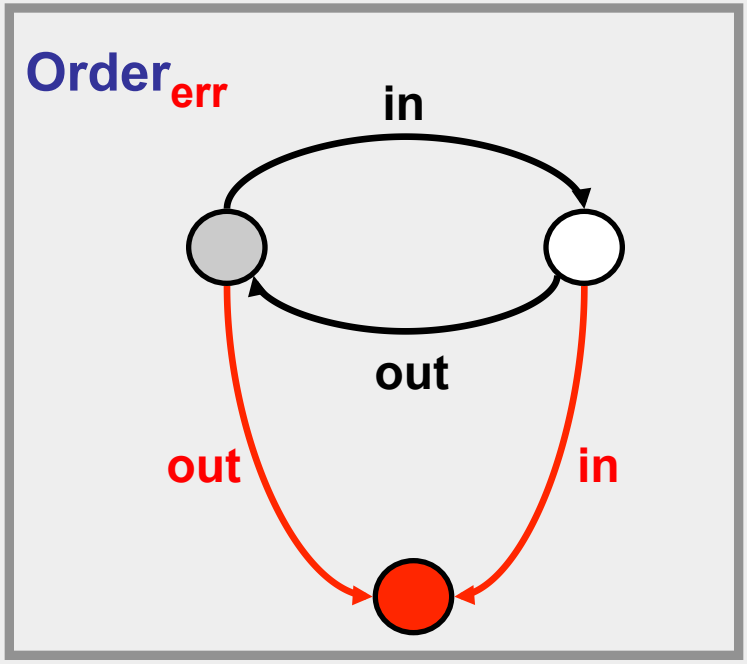
formalisms

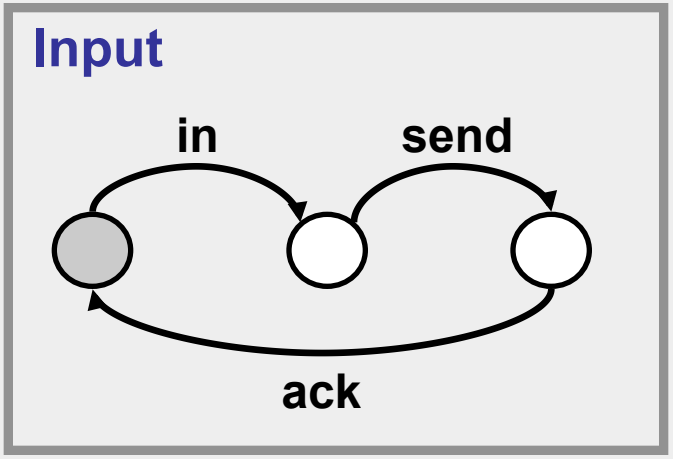
- a safety property P is a FSM
  - P describes all legal behaviors in terms of its alphabet
- property satisfaction: M $\models$ P iff $\mathcal{L}(M \uparrow \alpha P) \subseteq \mathcal{L}(P)$
- alternative definition
  - P_{err} – complement of P (determinize & complete P with "error" state)
  - bad behaviors lead to error
- M $\models$ 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})

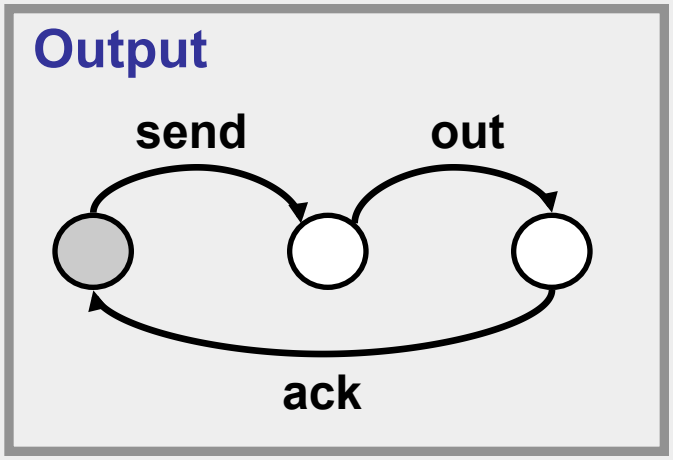
example

require in and out to alternate (property Order)

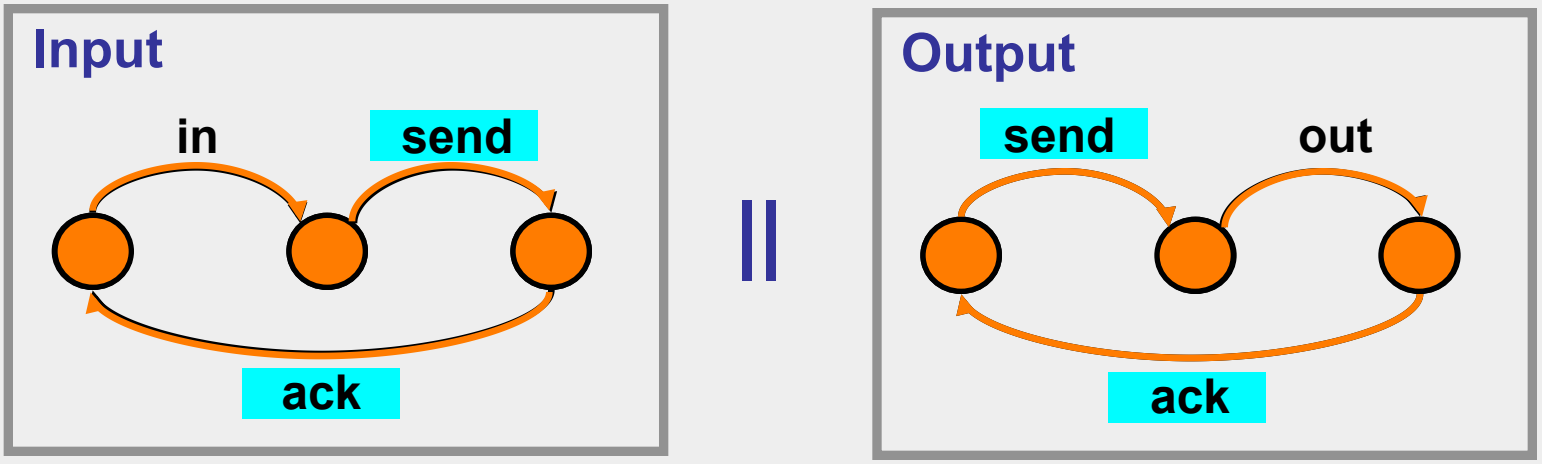




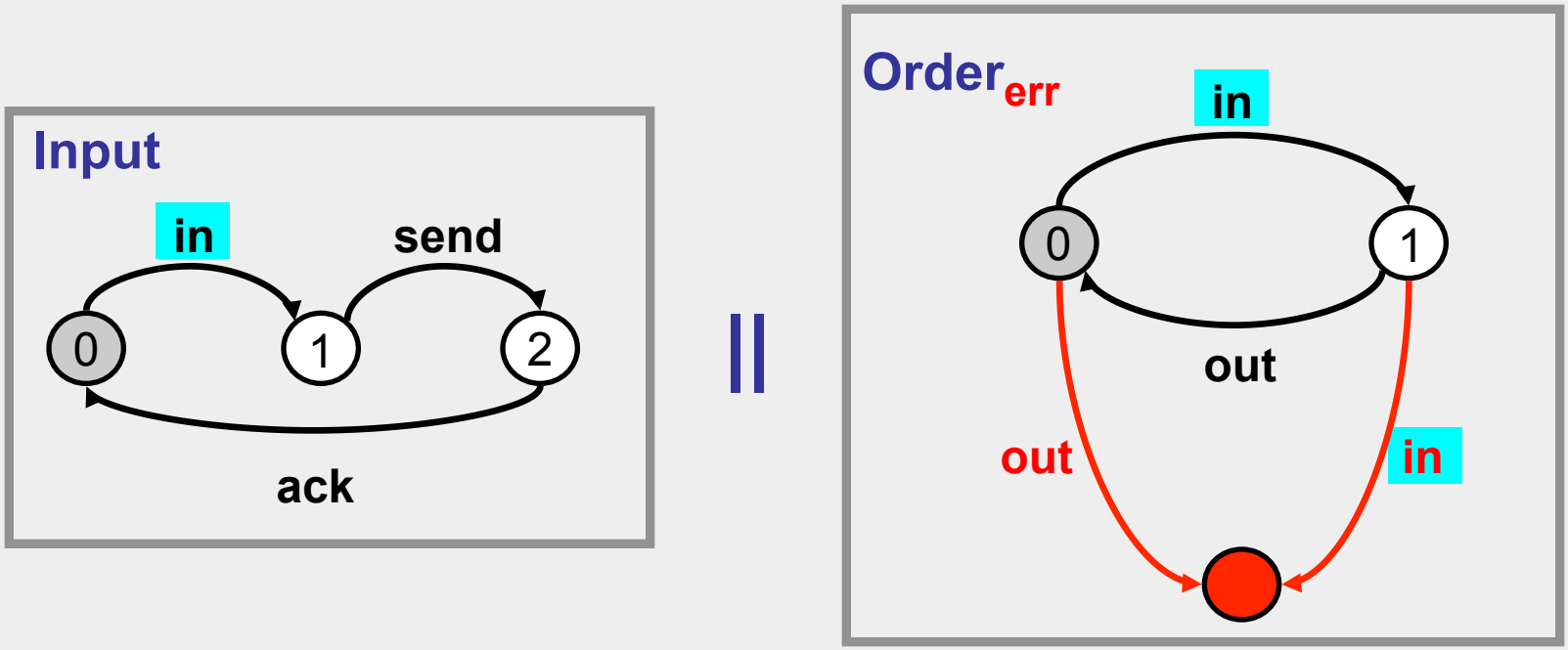




parallel composition



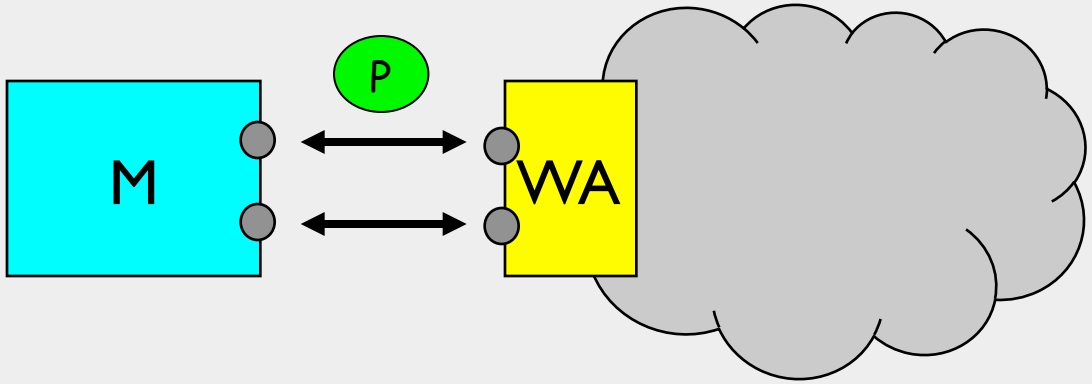
property satisfaction



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

cex. 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})

the weakest assumption

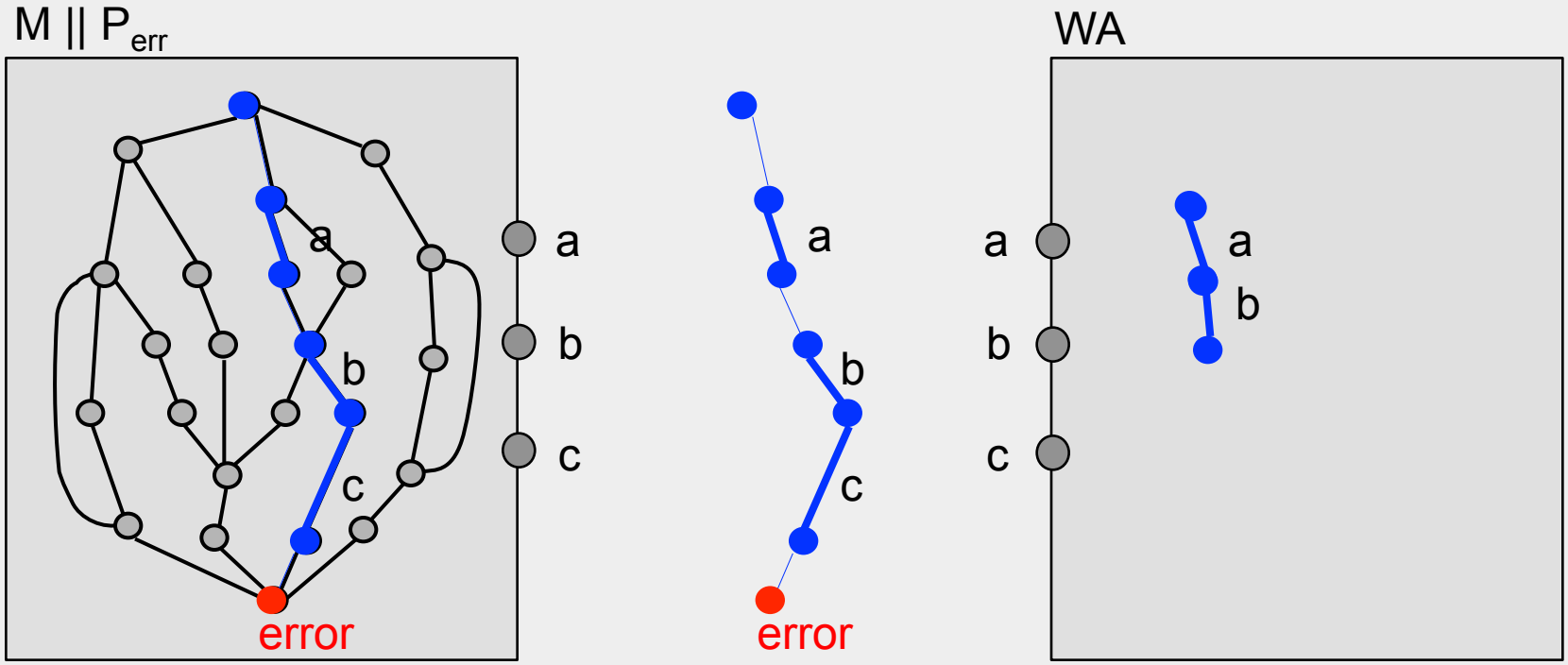


- given component M, property P, and the interface ∑ of M with its environment, generate the weakest environment assumption WA such that: ⟨WA⟩ M ⟨P⟩ holds

- weakest means that for all environments E:

$$\langle true \rangle \; M \;||\; E \; \langle P \rangle \;\; \textbf{IFF} \;\; \langle true \rangle \; E \; \langle WA \rangle$$

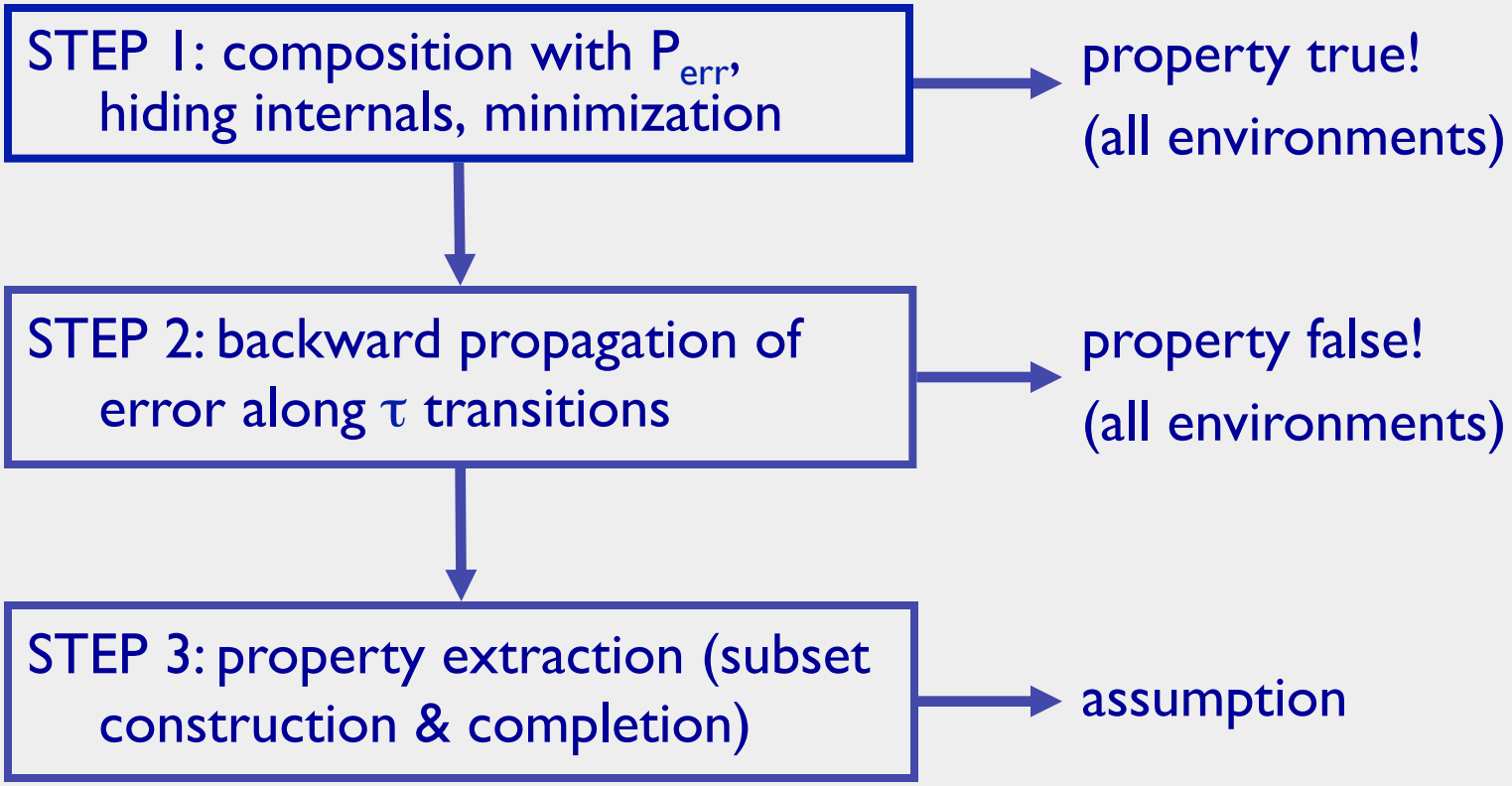
the weakest assumption



Weakest assumption

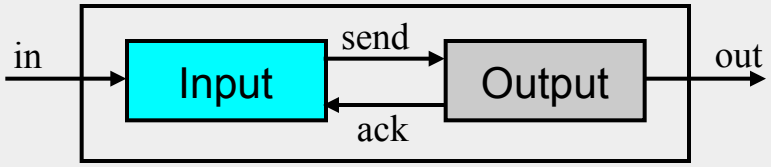
- Prevents component to go to error (safe)
- Should be as permissive as possible
- Uses only interface actions

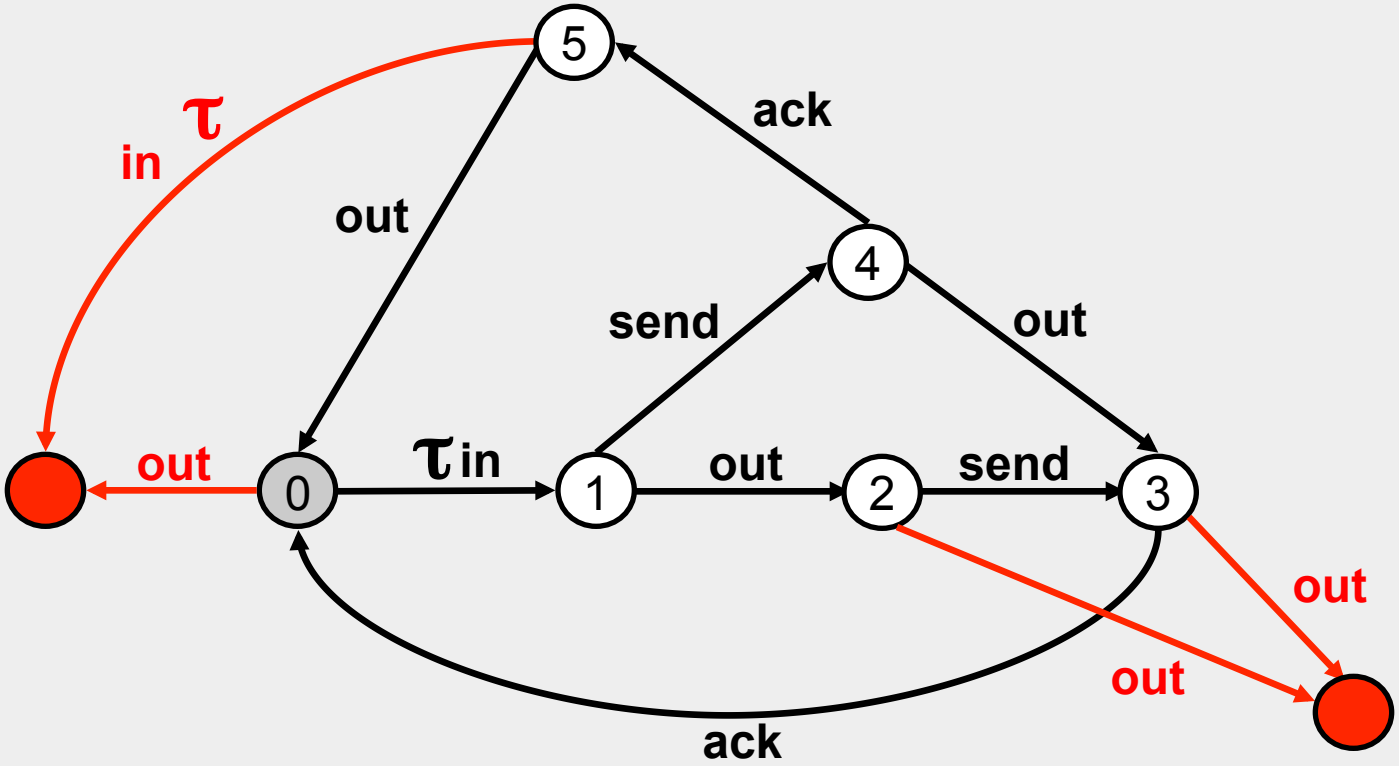
assumption generation [ASE' 02]



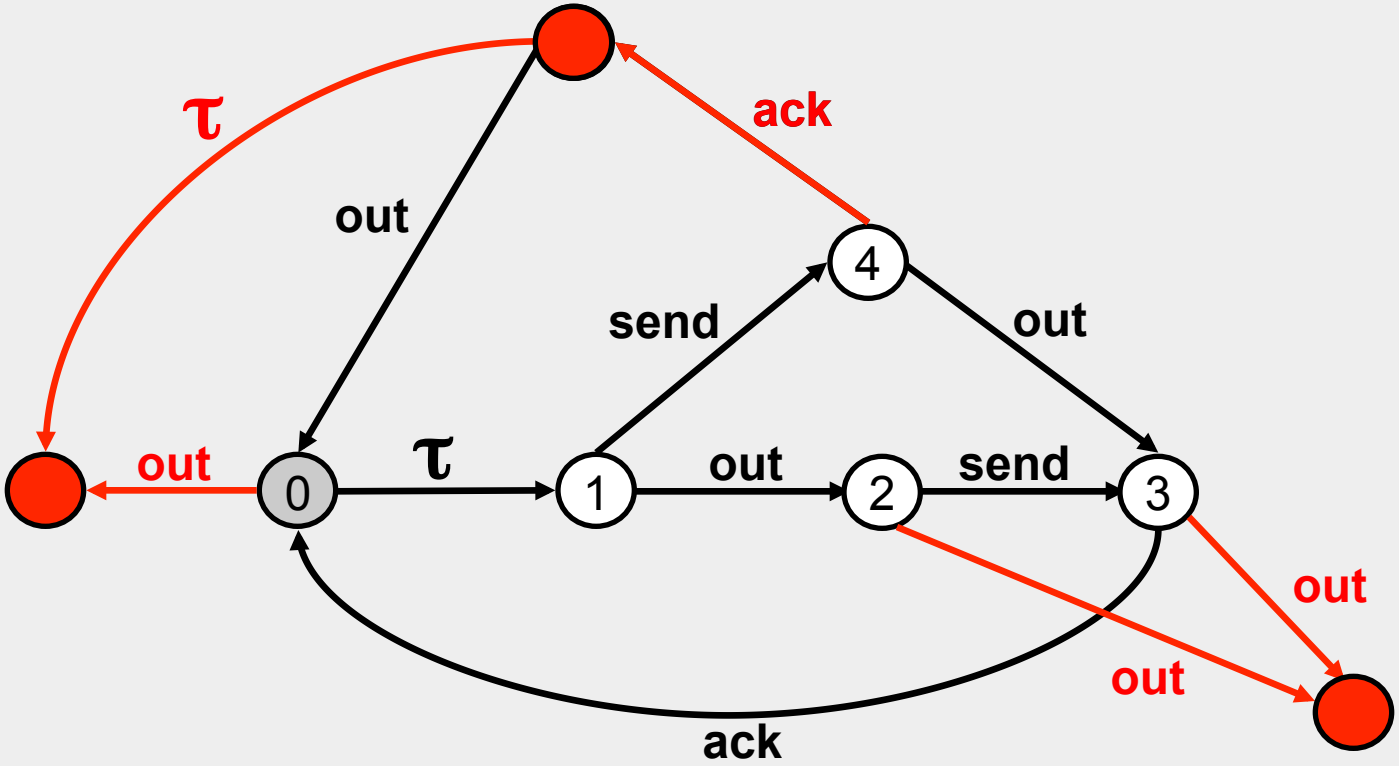
step 1: composition & hiding

Input || Order$_{err}$ \ {in}

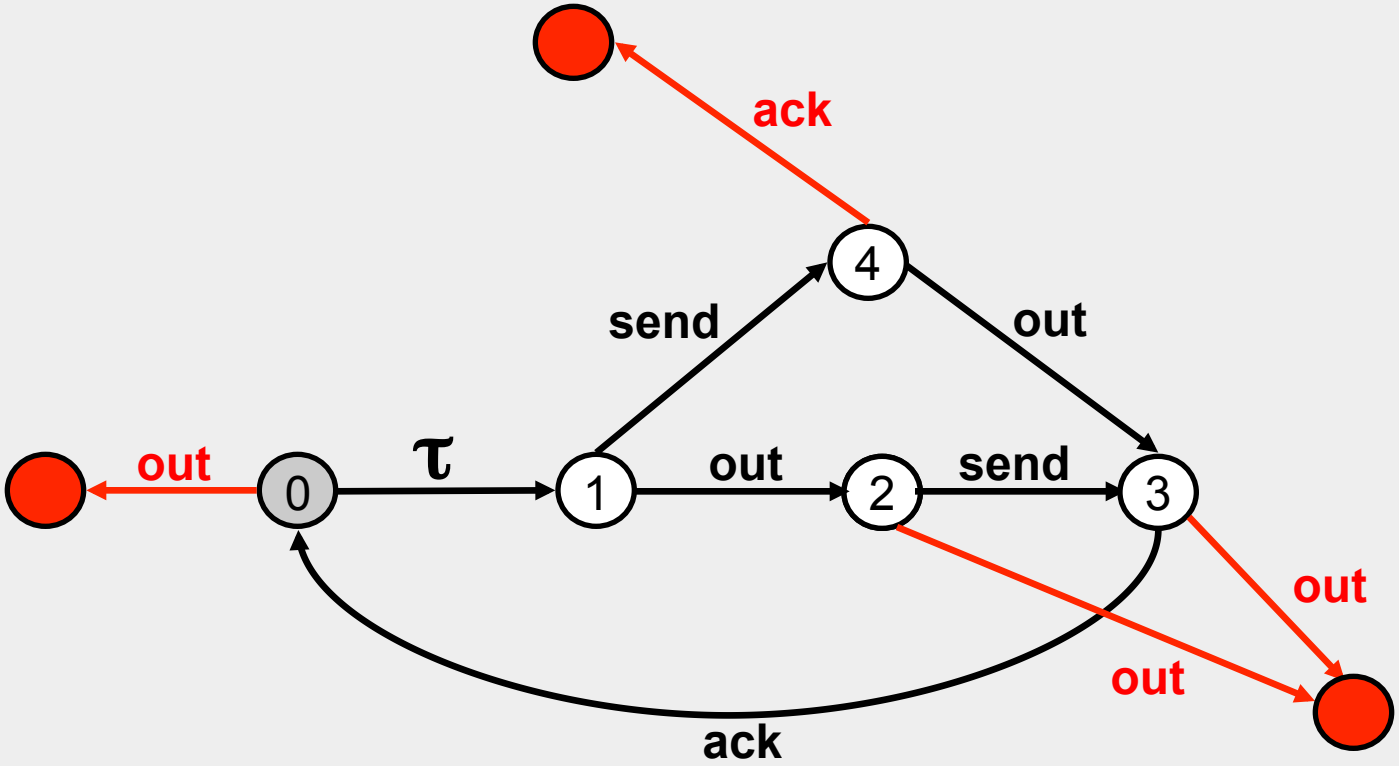




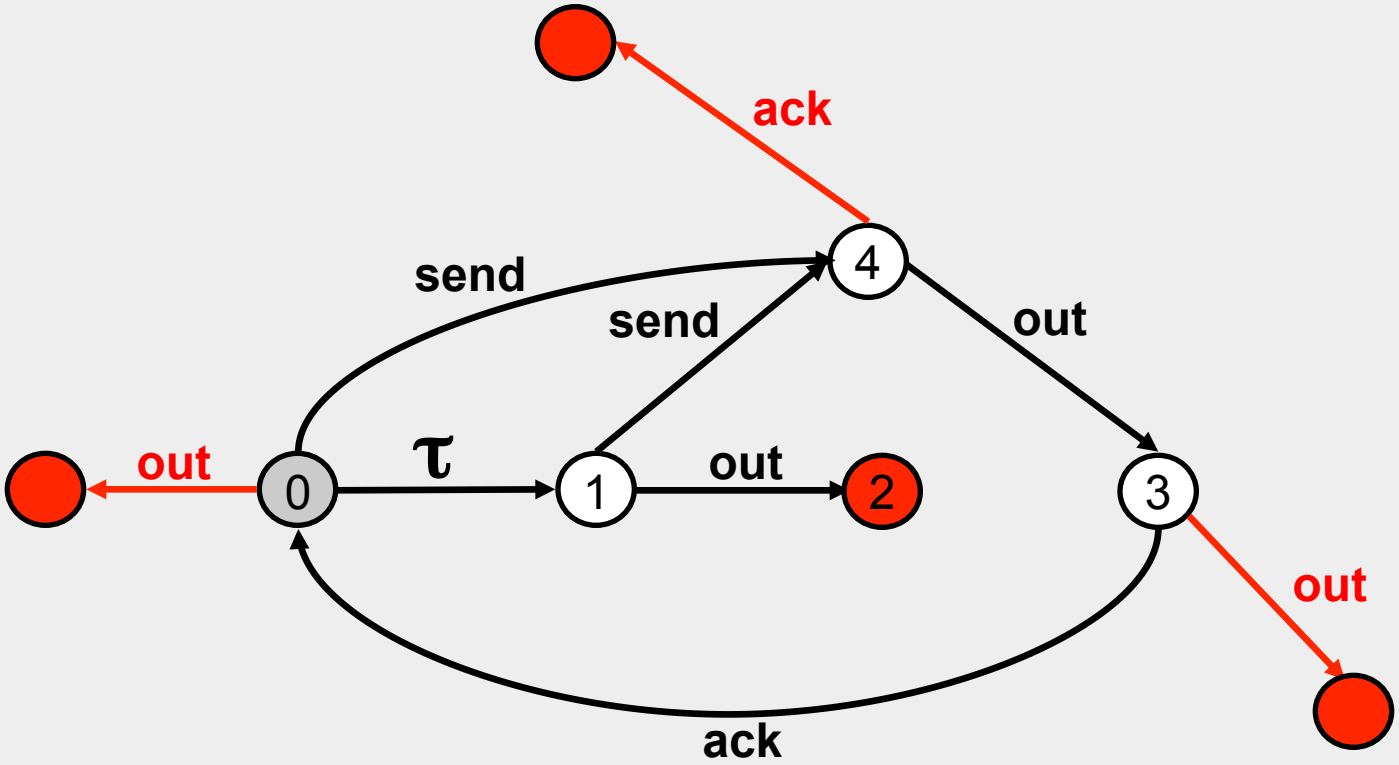
step 2: error propagation with τ



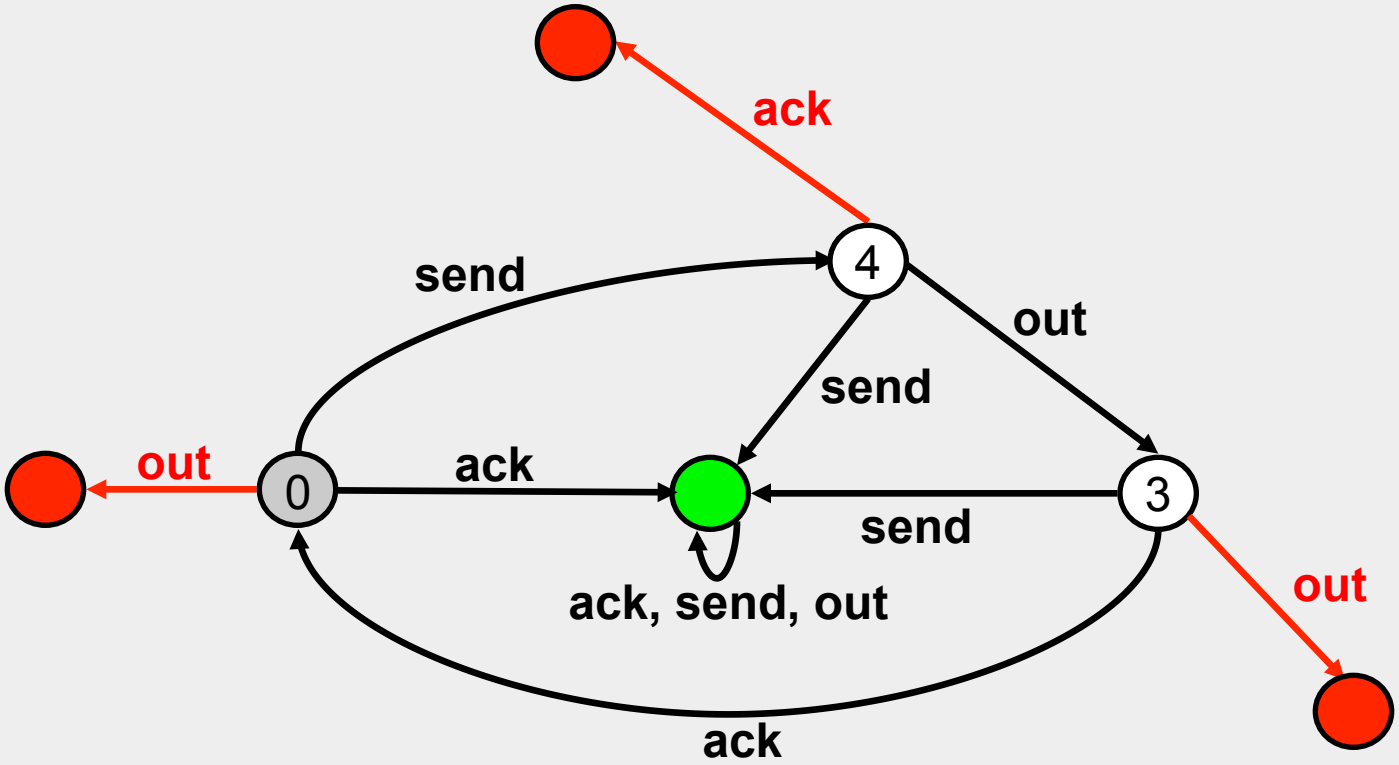
step 3: subset construction



step 3: subset construction



step 3: property construction



weakest assumption in AG reasoning

1. $\langle A \rangle\ M_1\ \langle P \rangle$
2. $\langle true \rangle\ M_2\ \langle A \rangle$

$\langle true \rangle\ M_1 \parallel M_2\ \langle P \rangle$

weakest assumption makes rule complete

$\langle WA \rangle\ M_1\ \langle P \rangle$ holds (WA could be *false*)
$\langle true \rangle\ M_2\ \langle WA \rangle$ holds implies $\langle true \rangle\ M_1 \parallel M_2\ \langle P \rangle$ holds
$\langle true \rangle\ M_2\ \langle WA \rangle$ not holds implies $\langle true \rangle\ M_1 \parallel M_2\ \langle P \rangle$ not holds

learning assumptions

iterative solution
intermediate results

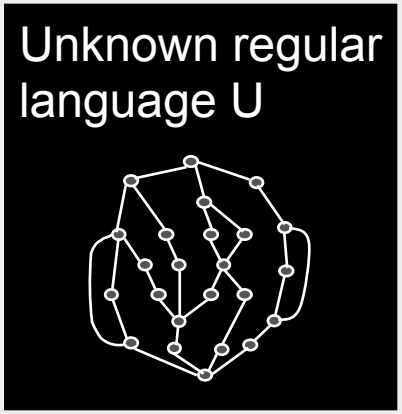
learning with L*

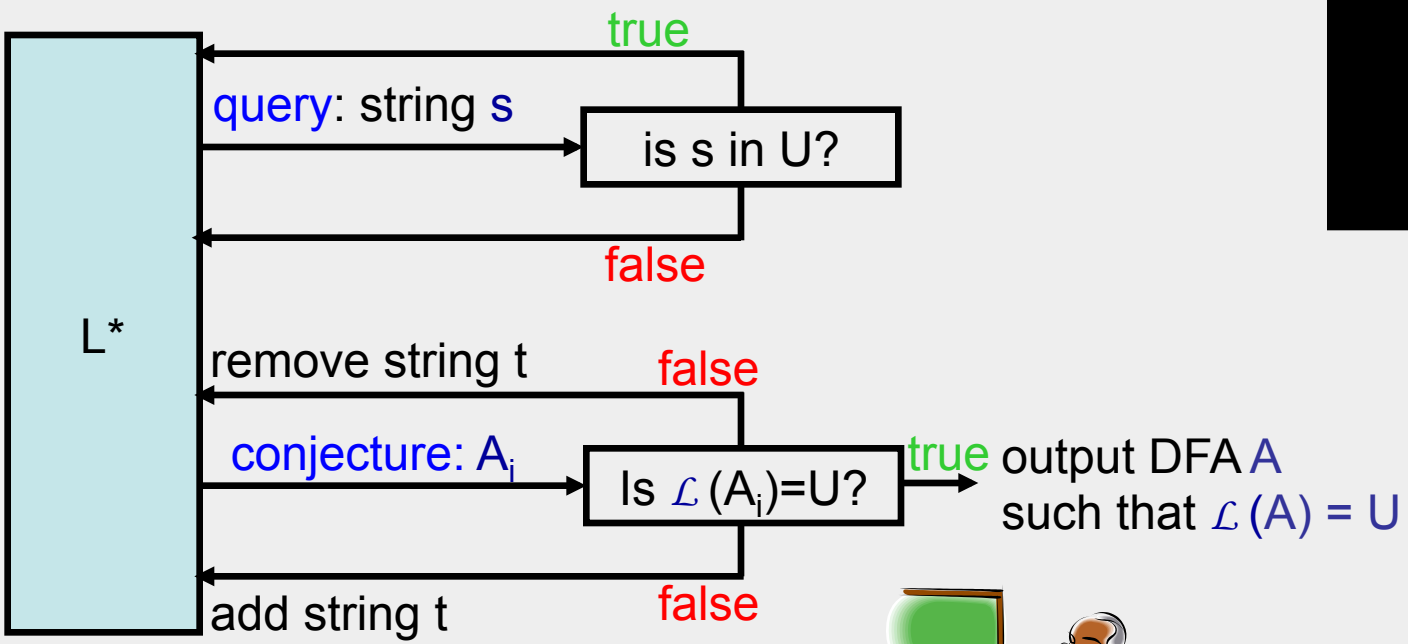
L* algorithm by Angluin, improved by Rivest & Schapire

Learns an unknown regular language U (over alphabet Σ)

Produces a DFA A such that $\mathcal{L}(A) = U$

Uses a teacher to answer two types of questions



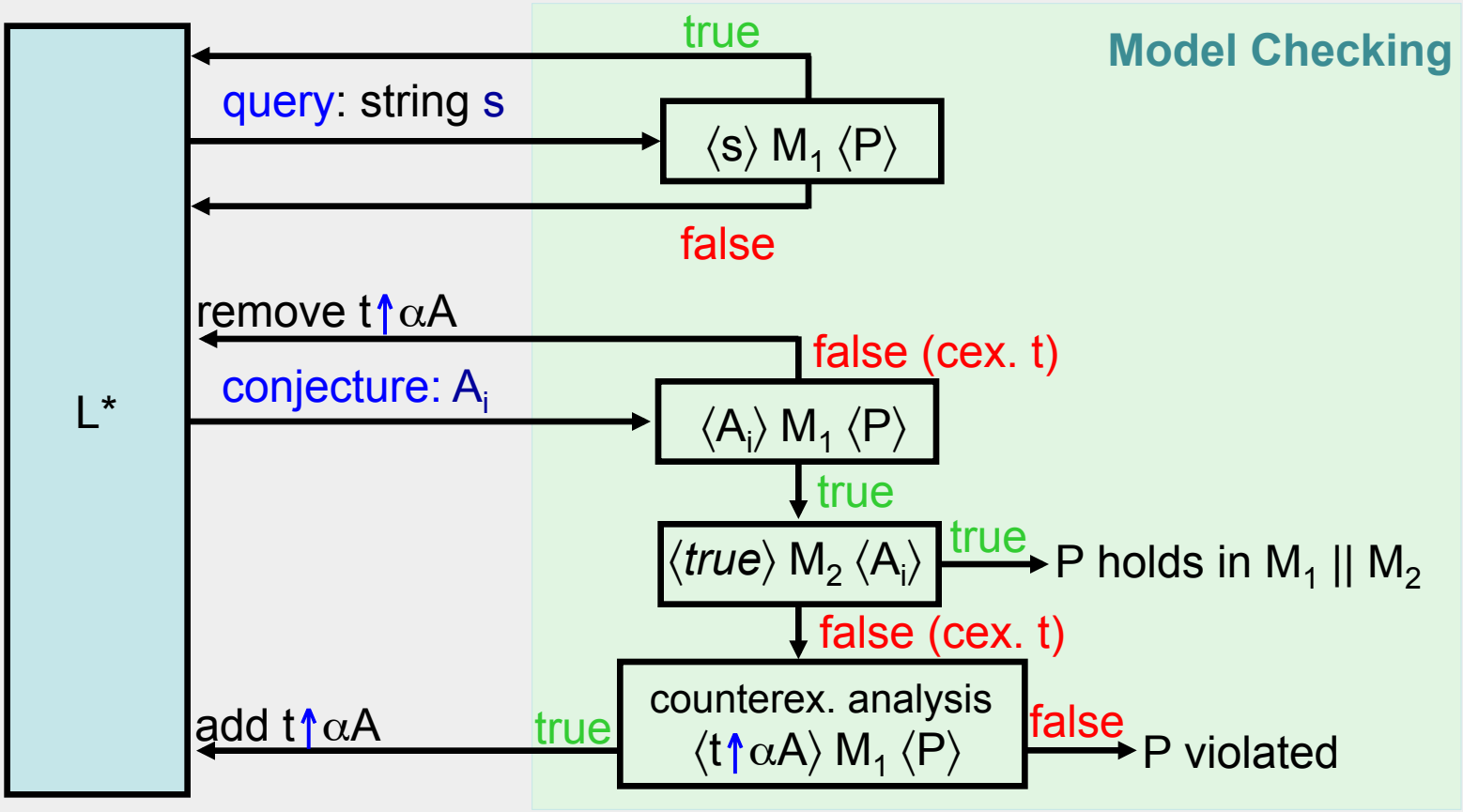


learning assumptions

Use L* to generate candidate assumptions

$\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

1.	$\langle A \rangle$	M_1	$\langle P \rangle$
2.	$\langle true \rangle$	M_2	$\langle A \rangle$
	$\langle true \rangle$	$M_1 \parallel M_2$	$\langle P \rangle$

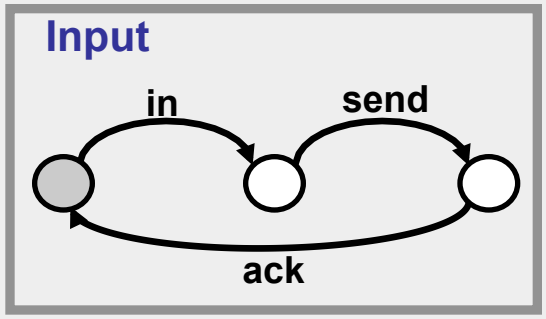


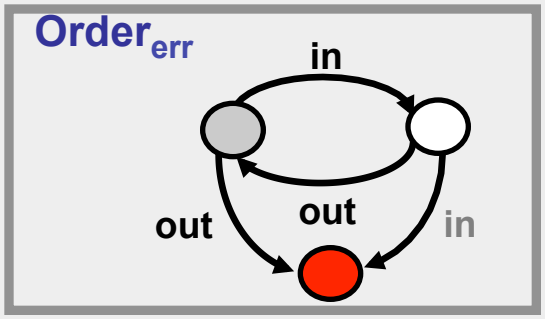
characteristics

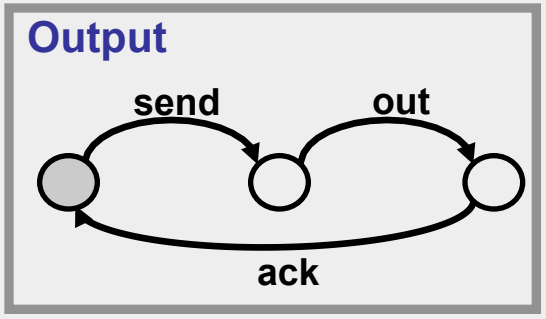
assumptions conjectured by L* are not comparable semantically

- 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





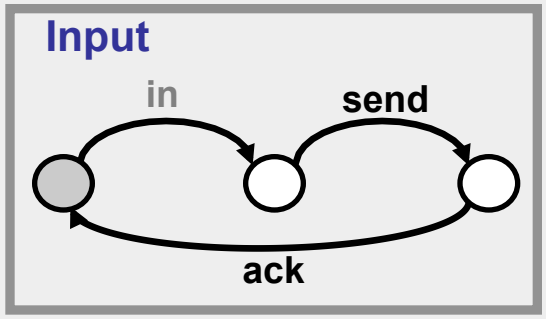


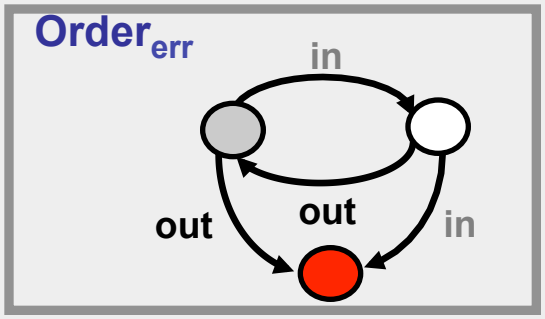
we check: ⟨true⟩ Input || Output ⟨Order⟩

M_1 = Input, M_2 = Output, P = Order

assumption alphabet: {send, out, ack}

queries

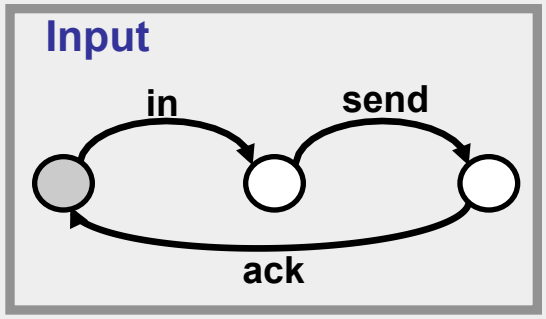


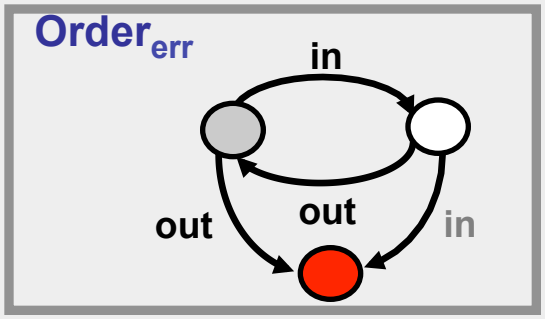


		E
	Table T	λ
S	λ	true
	out	false
S · Σ	ack	true
	out	false
	send	true
	out, ack	false
	out, out	false
	out, send	false

***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

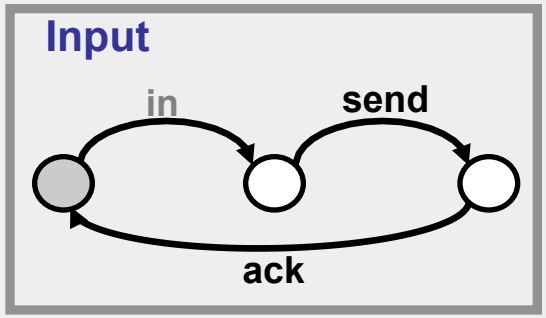
2 states – error state omitted

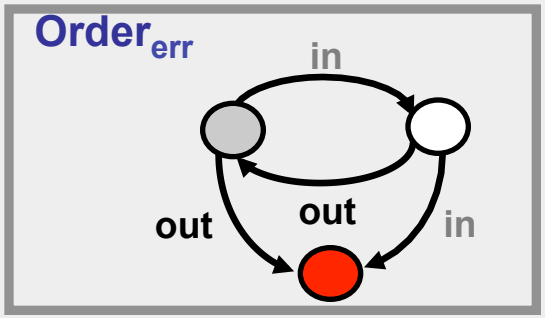
Assumption A_1

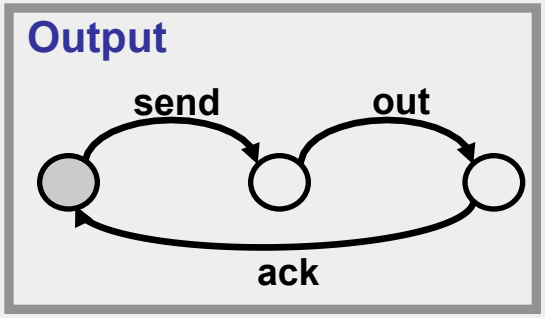
counterexamples add to *S*

S = set of prefixes
E = set of suffixes

conjectures







A_1:

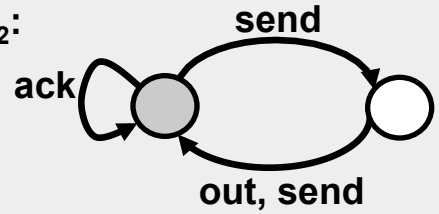
Oracle 1:
$\langle A_1 \rangle$ Input $\langle$Order$\rangle$

Counterexample:
c = $\langle$in,send,ack,in$\rangle$

Return to L*:
c$\uparrow \Sigma$ = $\langle$send,ack$\rangle$

Queries

A_2:



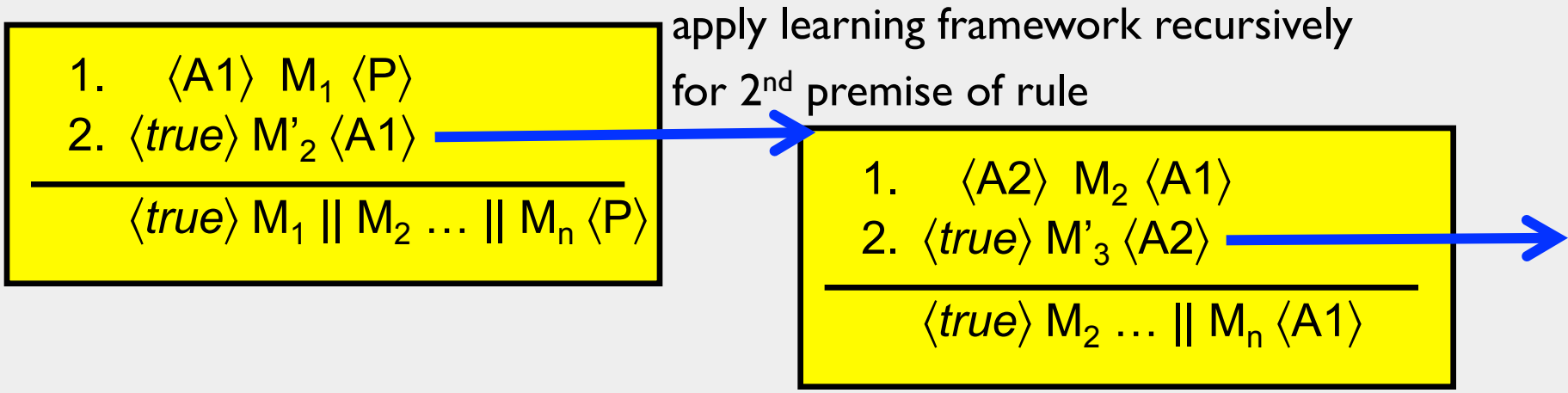
Oracle 1:
$\langle A_2 \rangle$ Input $\langle$Order$\rangle$
True

Oracle 2:
$\langle$true$\rangle$ Output $\langle A_2 \rangle$
True

property Order holds
on Input || Output

extension to *n* components

- Check if $M_1 \parallel M_2 \parallel \ldots \parallel M_n$ satisfies P
  - decompose it into M_1 and
  - $M'_2 = M_2 \parallel \ldots \parallel M_n$

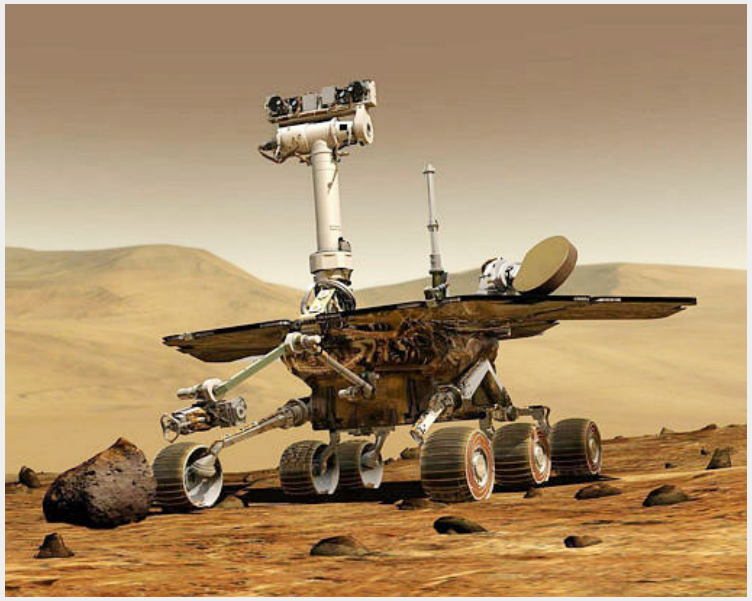


• • •

example

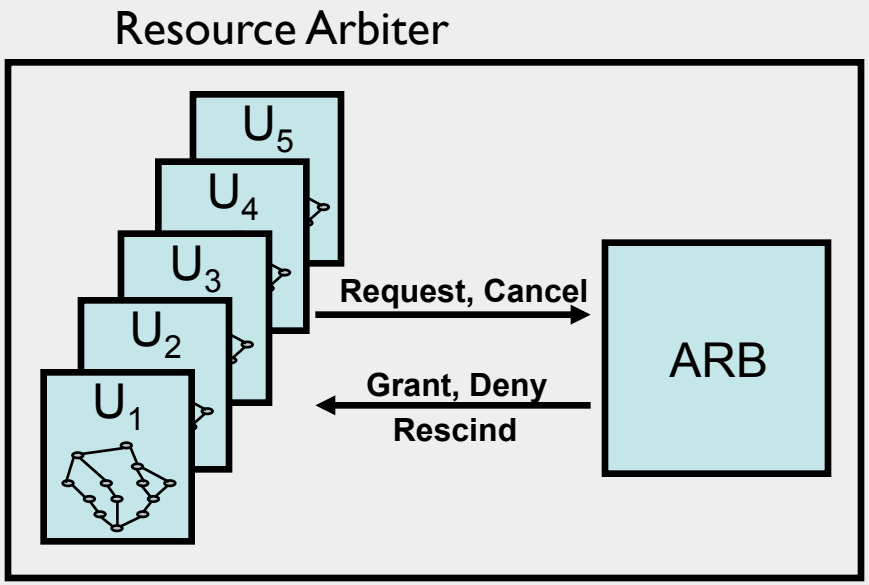
Mars Exploration Rover (MER) Resource Arbiter

- Local management of resource contention between resource consumers
- E.g. science instruments, communication systems
- k user threads and one server thread (arbiter)



Mutual exclusion between resources

- E.g. driving while capturing a camera image are mutually incompatible



recursive invocation

- Compute $A_1 \ldots A_5$ s.t.

$\langle A_1 \rangle\ U_1\ \langle P \rangle$ & $\langle true \rangle\ U_2 \parallel U_3 \parallel U_4 \parallel U_5 \parallel ARB\ \langle A_1 \rangle$

$\langle A_2 \rangle\ U_2\ \langle A_1 \rangle$ & $\langle true \rangle\ U_3 \parallel U_4 \parallel U_5 \parallel ARB\ \langle A_2 \rangle$

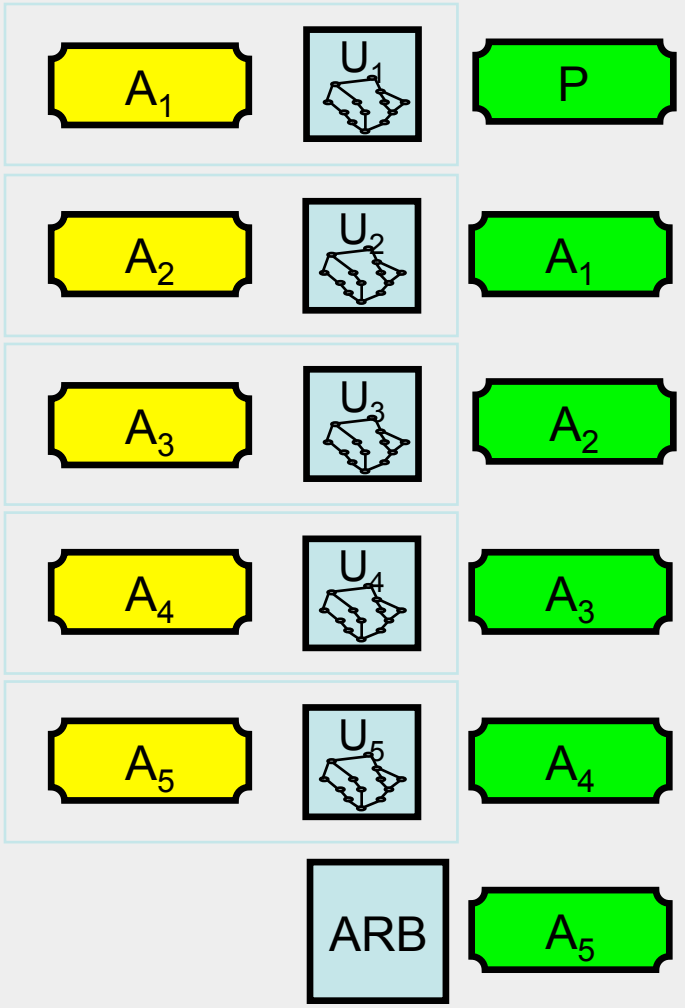
$\langle A_3 \rangle\ U_3\ \langle A_2 \rangle$ & $\langle true \rangle\ U_4 \parallel U_5 \parallel ARB\ \langle A_2 \rangle$

$\langle A_4 \rangle\ U_4\ \langle A_3 \rangle$ & $\langle true \rangle\ U_5 \parallel ARB\ \langle A_4 \rangle$

$\langle A_5 \rangle\ U_5\ \langle A_4 \rangle$ & $\langle true \rangle\ ARB\ \langle A_5 \rangle$

- Result: $\langle true \rangle\ U_1 \parallel .. \parallel U_5 \parallel ARB\ \langle P \rangle$ holds

- Compositional verification scaled to >5 users while monolithic verification ran out of memory [SPIN' 06]



symmetric rule

1. $\langle A_1 \rangle \; M_1 \; \langle P \rangle$
2. $\langle A_2 \rangle \; M_2 \; \langle P \rangle$
3. $P \models A_1 \wedge A_2$

$\langle \mathit{true} \rangle \; M_1 \parallel M_2 \; \langle P \rangle$

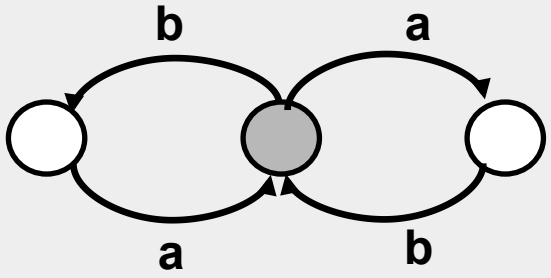
symmetric rule

1. $\langle A_1 \rangle \; M_1 \; \langle P \rangle$
2. $\langle A_2 \rangle \; M_2 \; \langle P \rangle$
3. $P \models A_1 \wedge A_2$

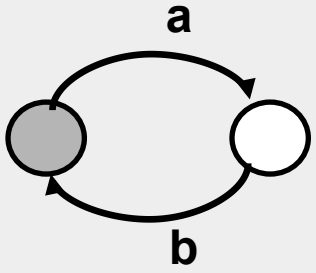
$\langle \mathit{true} \rangle \; M_1 \parallel M_2 \; \langle P \rangle$

Unsound!

M_1 , M_2



P, A_1 , A_2



symmetric rule [Misra&Chandi TSE'81]

1. $M_1 \models A_1 \rightarrow P$
2. $M_2 \models A_2 \rightarrow P$
3. $P \models A_1 \wedge A_2$

$\langle true \rangle\ M_1 \,||\, M_2\ \langle P \rangle$

Sound!

- Not sound if ➔ is interpreted as logical implication
- Use induction over time steps:
  - P holds initially in M
  - if assumption A holds up to the k-th step in any trace of M, then
  - guarantee P holds up to the k+1-th step in that trace, for all $k \geq 0$

symmetric rule [SAVCBS05]

1. $\langle A_1 \rangle \; M_1 \; \langle P \rangle$
2. $\langle A_2 \rangle \; M_2 \; \langle P \rangle$
3. $\mathcal{L}(coA_1 \,||\, coA_2) \subseteq \mathcal{L}(P)$

$\langle true \rangle \; M_1 \,||\, M_2 \; \langle P \rangle$

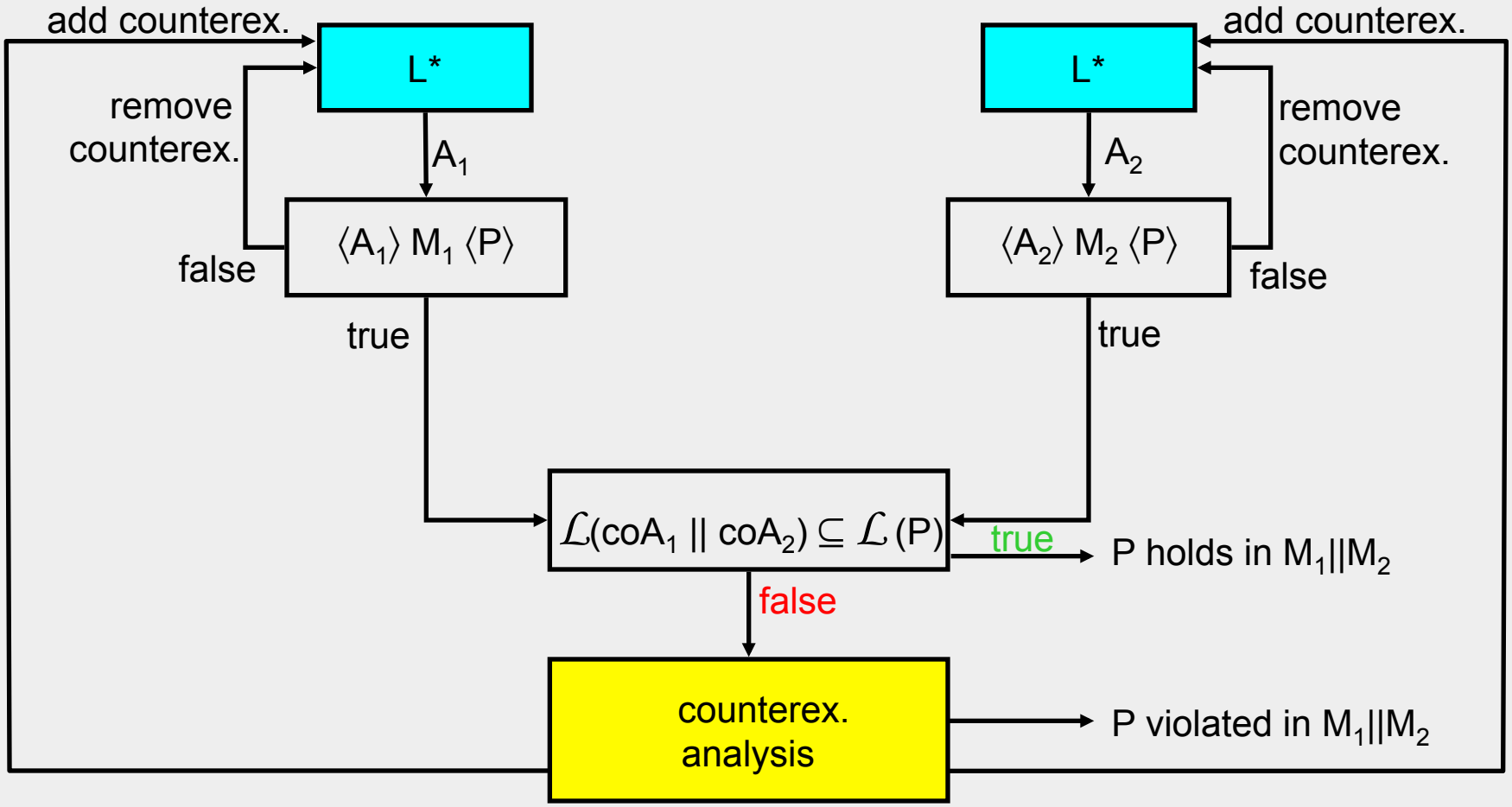
Common traces ruled out by assumptions satisfy P

Sound!

coA_i = complement of A_i, for i=1,2

$\alpha P \subseteq \alpha M_1 \cup \alpha M_2$; $\alpha A_i \subseteq (\alpha M_1 \cap \alpha M_2) \cup \alpha P$, for i =1,2

learning framework [SAVCBS05]



circular rule [Grumberg&Long Concur'91]

1. $\langle A_1 \rangle \; M_1 \; \langle P \rangle$
2. $\langle A_2 \rangle \; M_2 \; \langle A_1 \rangle$
3. $\langle \mathit{true} \rangle \; M_1 \; \langle A_2 \rangle$

$\langle \mathit{true} \rangle \; M_1 \,||\, M_2 \; \langle P \rangle$

- Similar to asymmetric rule
  - Applied recursively to 3 components
  - First and last component coincide
  - Hence learning framework similar

Automated Compositional Verification part 2

Corina Păsăreanu
CMU Silicon Valley/ NASA Ames Research Center

outline

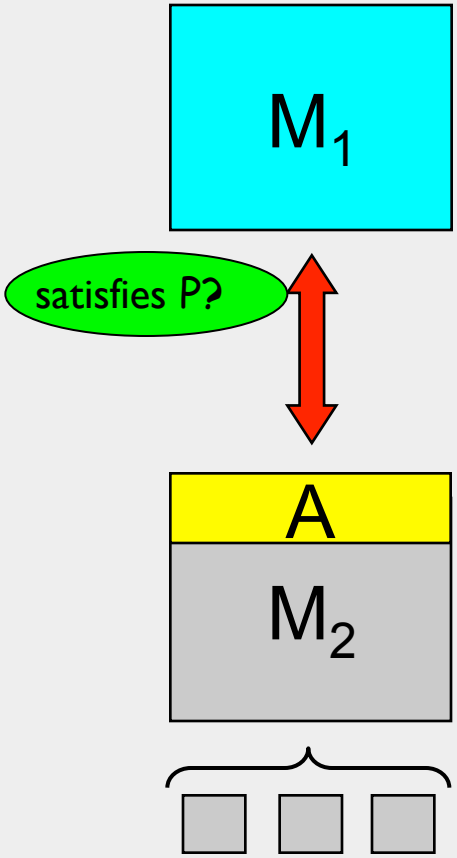
Part I

- assume-guarantee reasoning
- computing assumptions
- learning assumptions
- multiple components
- different assume-guarantee rules

Part II

- alphabet refinement
- assume-guarantee abstraction refinement
- reasoning about code
- related work
- conclusion

compositional verification

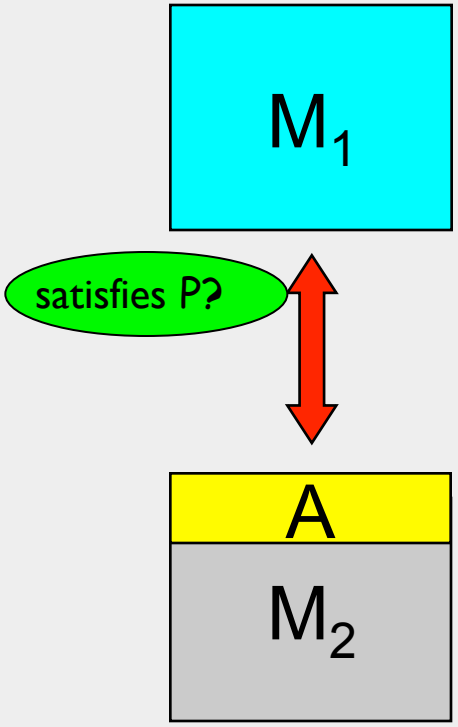


Does system made up of M_1 and M_2 satisfy property P?

- Check P on entire system: too many states!
- Use the natural decomposition of the system into its components to break-up the verification task
- Check components in isolation:

  Does M_1 satisfy P?
  - Typically a component is designed to satisfy its requirements in specific contexts / environments
- Assume-guarantee reasoning:
  - Introduces assumption A representing M_1's "context"

assume-guarantee reasoning



- Reason about triples:

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

  The formula 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

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

"discharge" the assumption

How do we come up with the assumption A?
(usually a difficult manual process)

Solution: synthesize A automatically

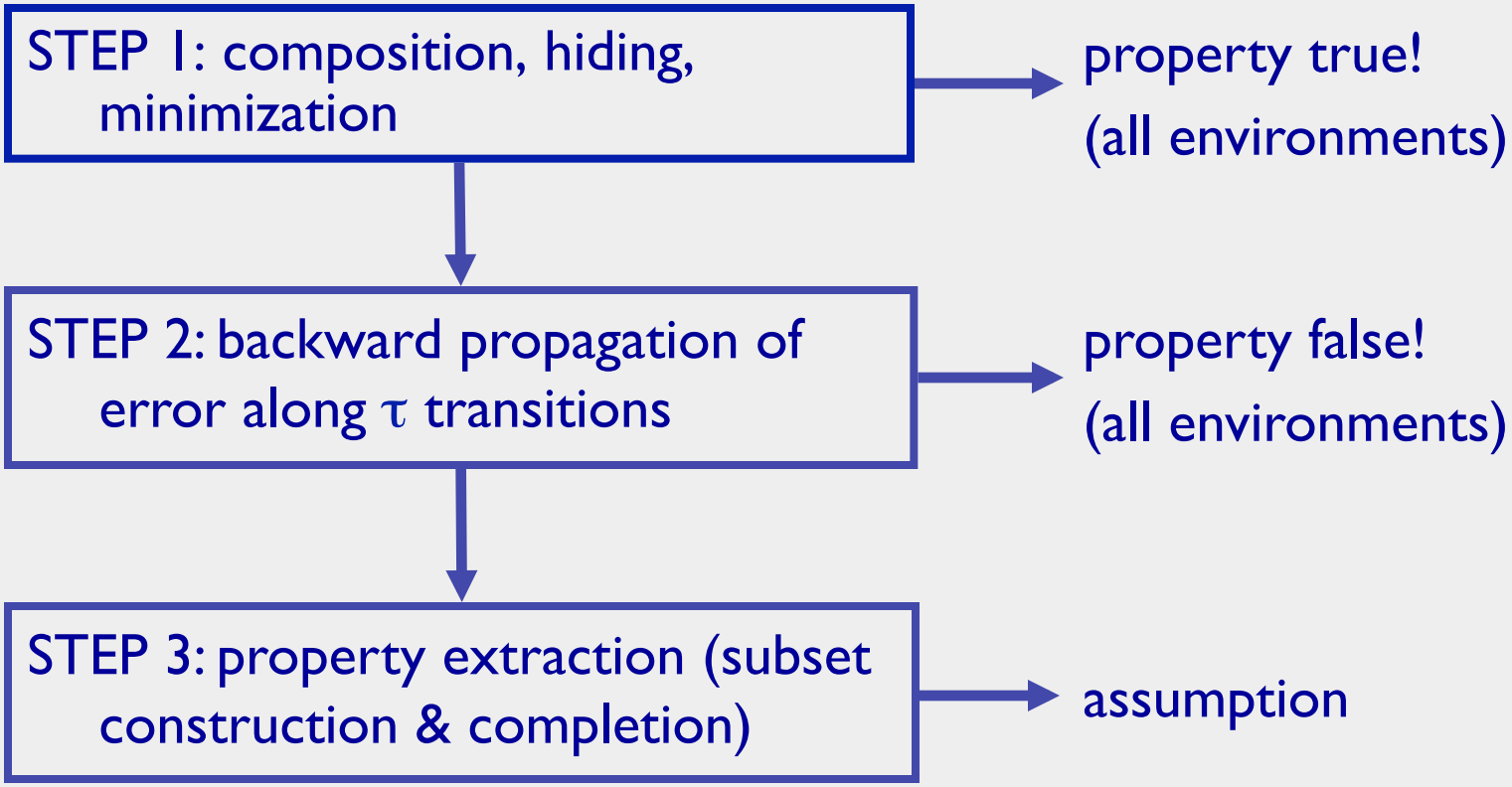
the weakest assumption

- 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:

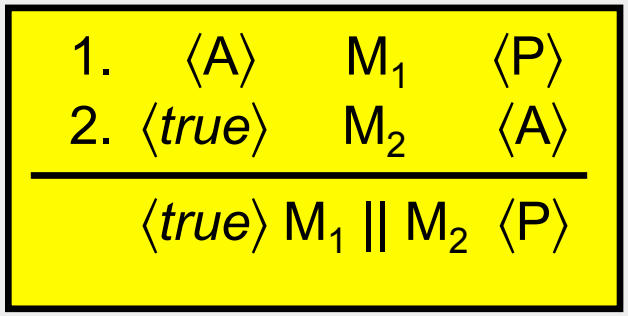
$$\langle \mathit{true} \rangle \; M \;||\; E \; \langle P \rangle \text{ IFF } \langle \mathit{true} \rangle \; E \; \langle WA \rangle$$

assumption generation [ASE' 02]



learning for assume-guarantee reasoning

- use an off-the-shelf learning algorithm to build assumption

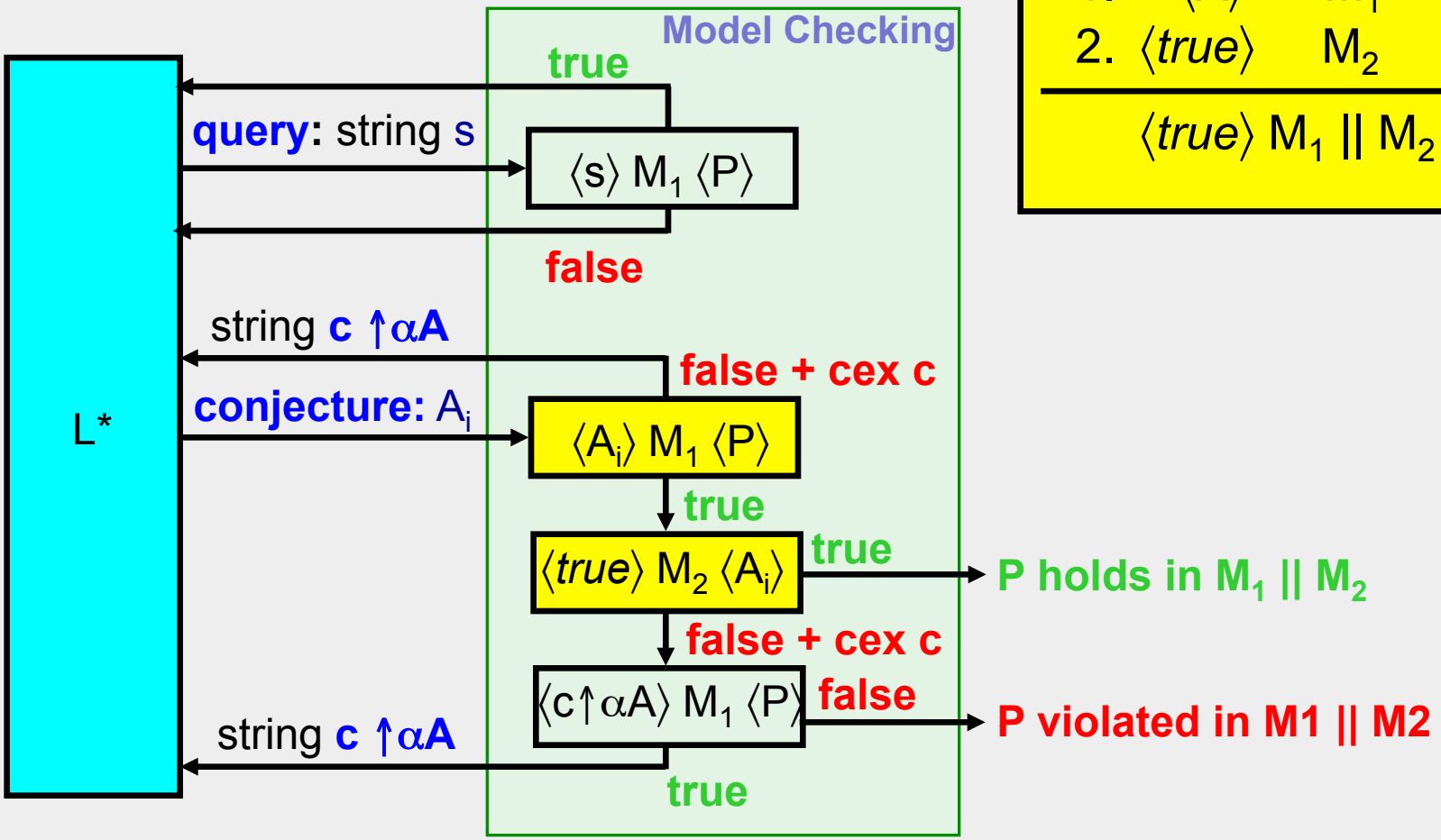


- process is *iterative*
- *a*ssumptions generated by querying the system, gradually refined
- queries answered by model checking
- refinement based on counterexamples

learning assumptions

- Use L* to generate candidate assumptions
- $\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$

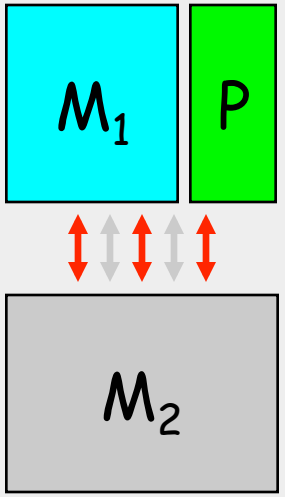
$$\frac{\begin{array}{lll} 1. & \langle A \rangle & M_1 \quad \langle P \rangle \\ 2. & \langle true \rangle & M_2 \quad \langle A \rangle \end{array}}{\langle true \rangle \; M_1 \,||\, M_2 \; \langle P \rangle}$$



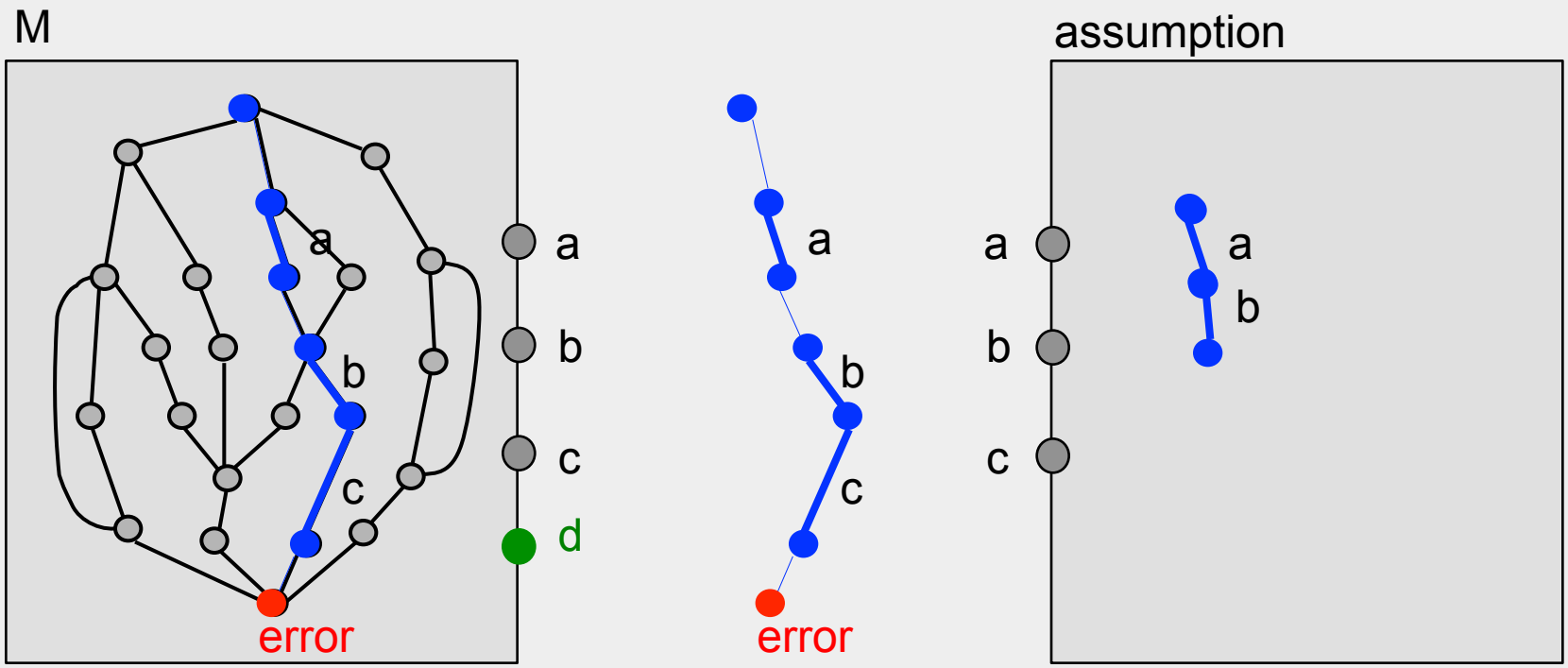
- guaranteed to terminate
- reaches weakest assumption or terminates earlier

assumption alphabet refinement

- rule ASYM
  - Assumption alphabet was fixed during learning
  - $\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$
- [SPIN'06]: A subset alphabet
  - may be sufficient to prove the desired property
  - may lead to smaller assumption



assumption alphabet refinement

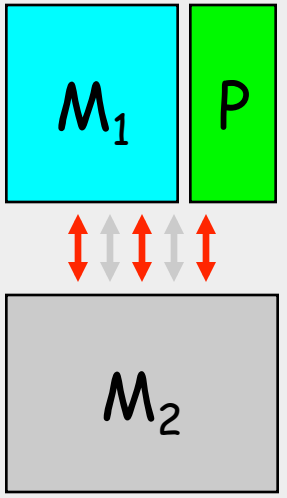


interface action d not relevant for the property

- e.g. may never appear on a path to error
- no need to include in alphabet assumption
- results in smaller assumption

assumption alphabet refinement [TACAS'07]

- How do we compute a good subset of the assumption alphabet?
- Solution: iterative alphabet refinement
  - Start with small alphabet
  - Apply learning framework
  - Add actions as necessary
  - Discovered by analysis of counterexamples from model checking



learning with alphabet refinement

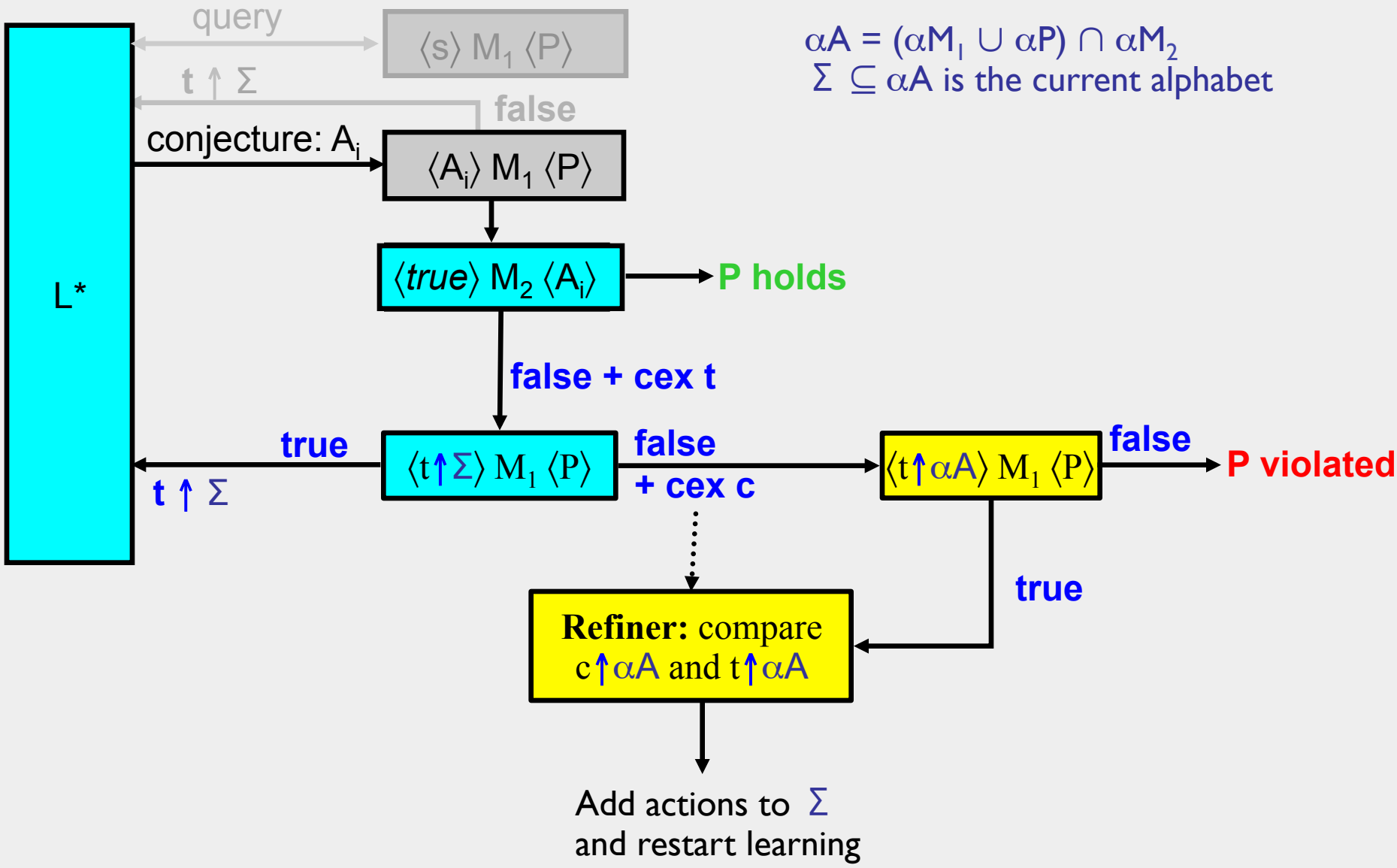
1. Initialize Σ to subset of alphabet $\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$
2. If learning with Σ returns true, return true and go to 4. (END)
3. If learning returns false (with counterexample c), perform

   extended counterexample analysis on c.

   If c is real, return false and go to 4. (END)

   If c is spurious, add more actions from αA to Σ and go to 2.
4. END

extended counterexample analysis



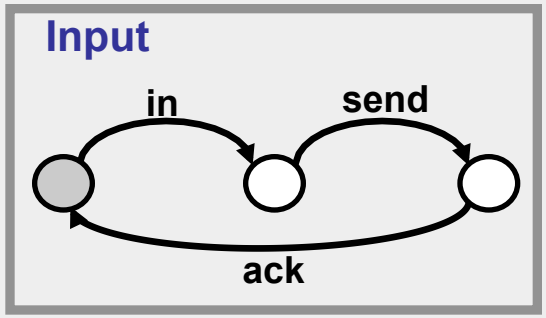
$\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$
$\Sigma \subseteq \alpha A$ is the current alphabet

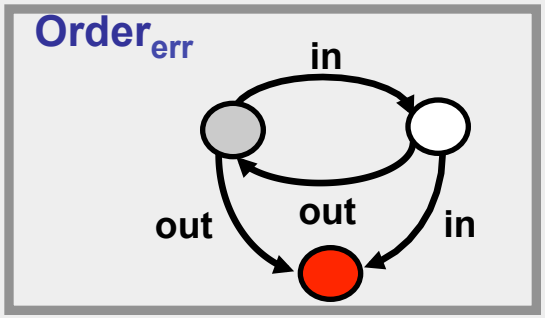
assumption under-approximation

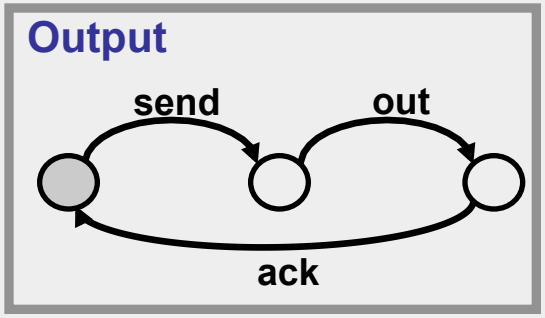
- let *A* be assumption resulted from L* with full interface alphabet αA
- during alphabet refinement:
  - assumptions *A'*, *A"* with alphabets $\alpha A' \subset \alpha A" \subset \alpha A$
  - are *under-approximations*, *i.e.*,
    $\mathcal{L}(A') \subseteq \mathcal{L}(A") \subseteq \mathcal{L}(A)$

- See also learning with optimal alphabet refinement
  - developed independently by Chaki & Strichman 07

alphabet refinement







$\sum$ = { out } αA = { send, out, ack }

⟨true⟩ Output ⟨A_i⟩ ⇒ false with t = ⟨send, out⟩

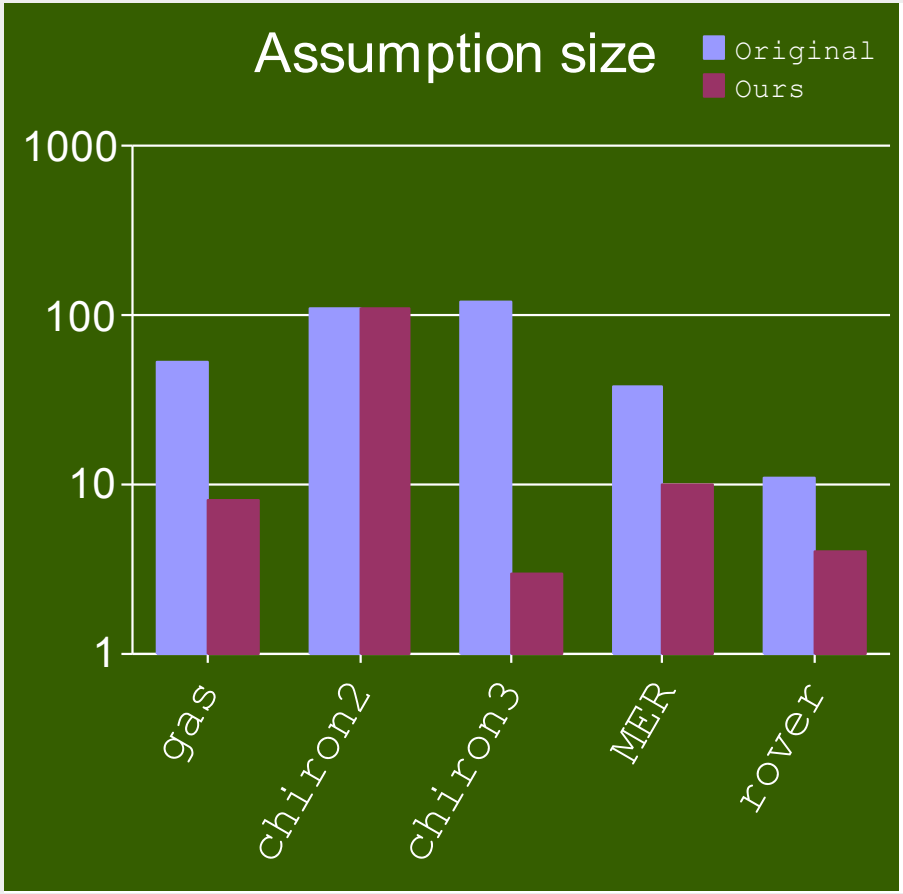
- $t\uparrow\Sigma$ = ⟨out⟩
- $t\uparrow\alpha A$ = ⟨send, out⟩

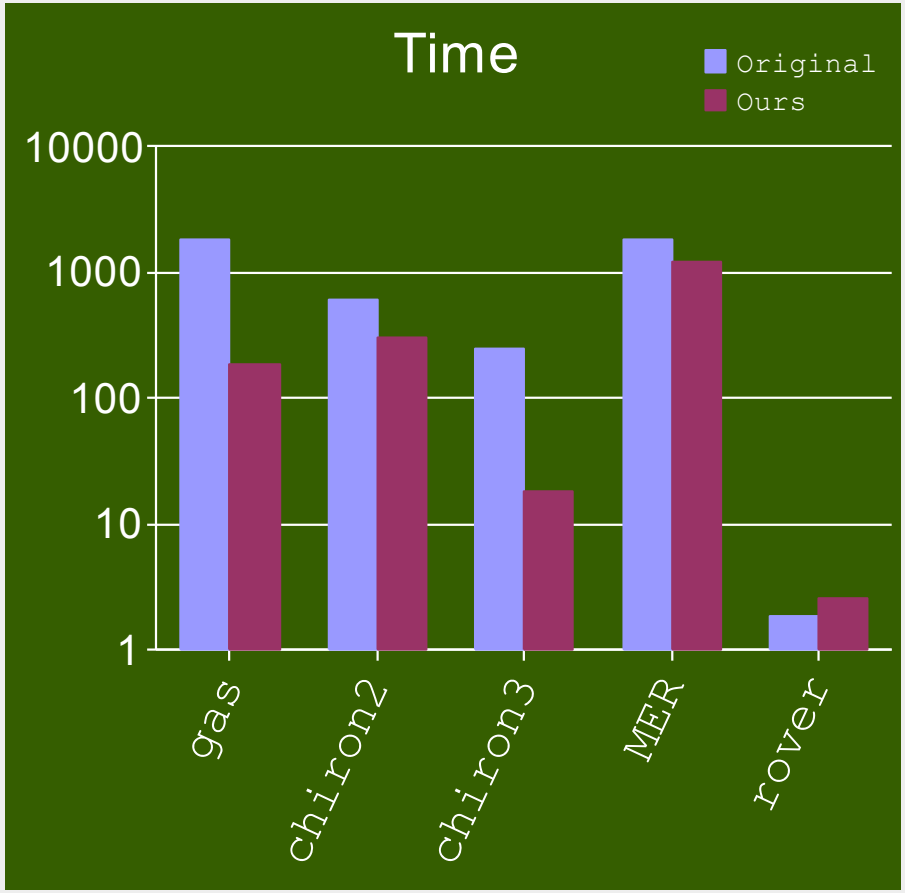
⟨$t\uparrow\Sigma$⟩ Input ⟨P⟩ ⇒ false with counterex. c = ⟨out ⟩

⟨$t\uparrow\alpha A$⟩ Input ⟨P⟩ ⇒ true **Not a real counterexample!**

compare ⟨out⟩ with ⟨send, out⟩ ⇒ add “send” to $\sum$

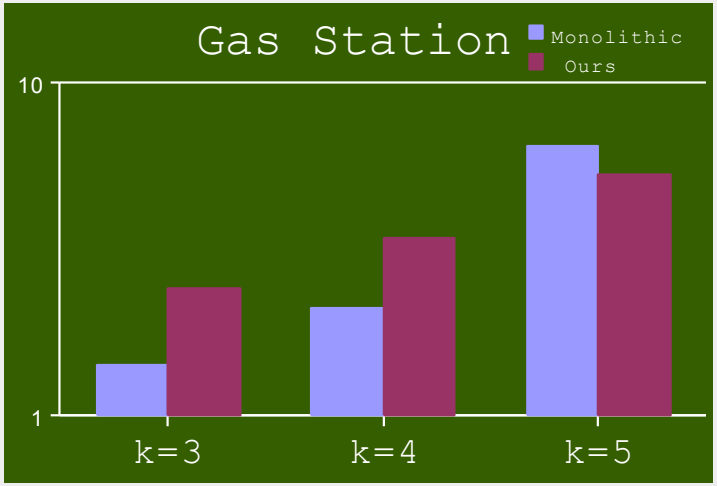
comparison with original learning

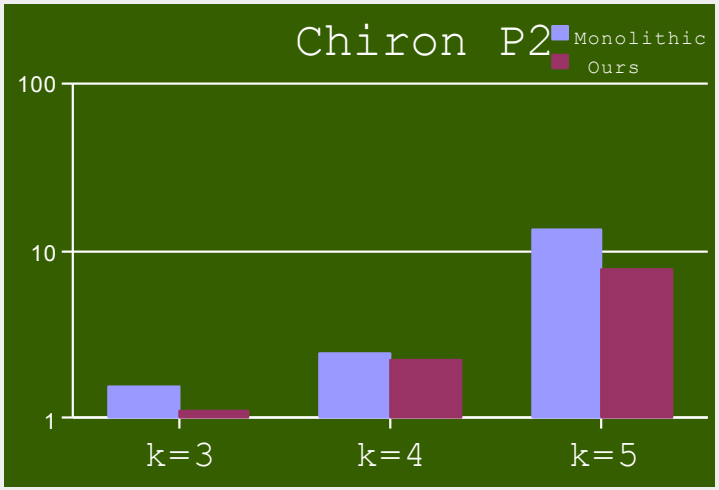


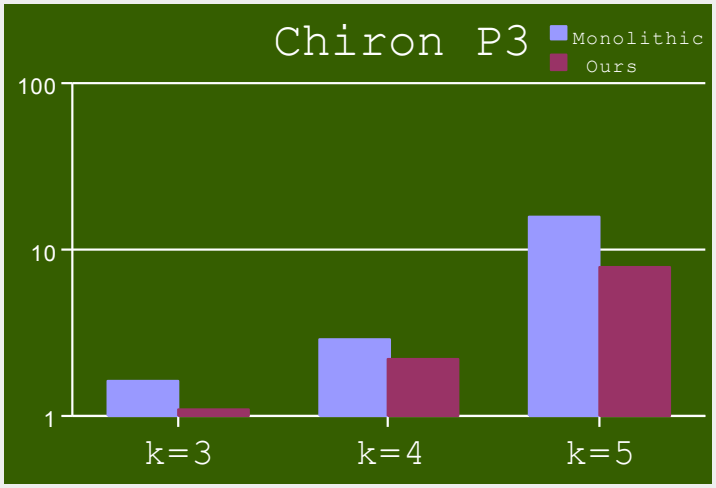


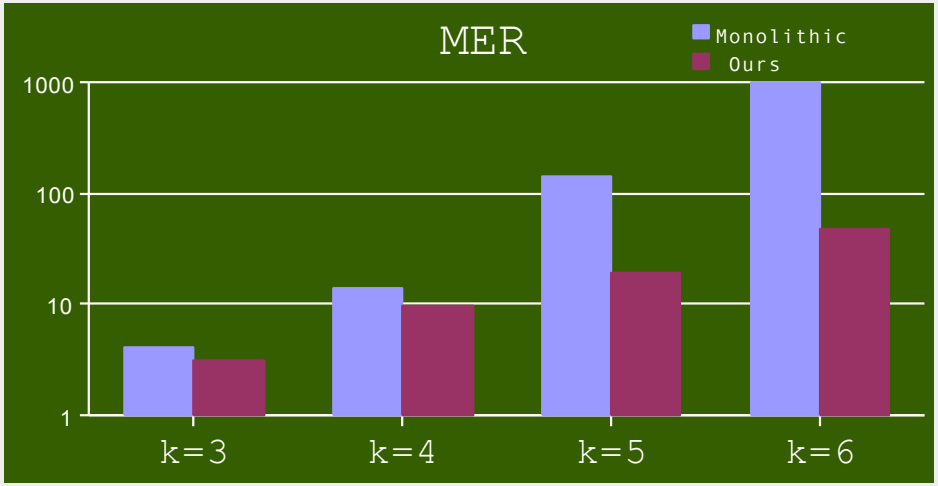
thanks Mihaela Bobaru

comparison with non-compositional (memory)









thanks Mihaela Bobaru

results

- Rule ASYM more effective than rules SYM and CIRC
- Recursive version of ASYM the most effective
  - When reasoning about more than two components
- Alphabet refinement improves learning based assume guarantee verification significantly
- Learning based assume guarantee reasoning
  - Can incur significant time penalties
  - Not always better than non-compositional (monolithic) verification
  - Sometimes, significantly better in terms of memory

abstraction

assume-guarantee abstraction refinement (AGAR)

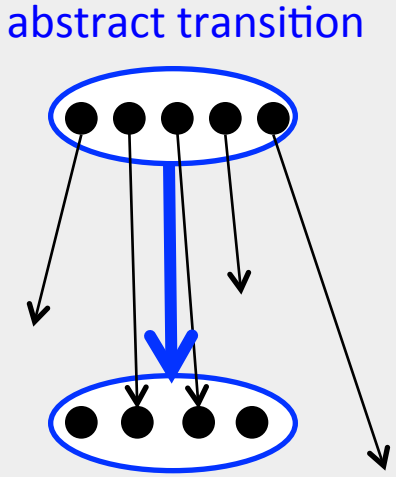
$$\frac{\begin{array}{lll} 1. & \langle A \rangle & M_1 \quad \langle P \rangle \\ 2. & \langle \mathit{true} \rangle & M_2 \quad \langle A \rangle \end{array}}{\langle \mathit{true} \rangle \; M_1 \parallel M_2 \; \langle P \rangle}$$

- Instead of learning A, build A as an *over-approximating abstraction* of M_2
- Why?
  - Use "more" information from M_1 and M_2
  - Nondeterministic assumptions can be exponentially smaller than deterministic ones
- [CAV'08]

assume-guarantee abstraction refinement (AGAR)

1. $\langle A \rangle \; M_1 \; \langle P \rangle$
2. $\langle true \rangle \; M_2 \; \langle A \rangle$

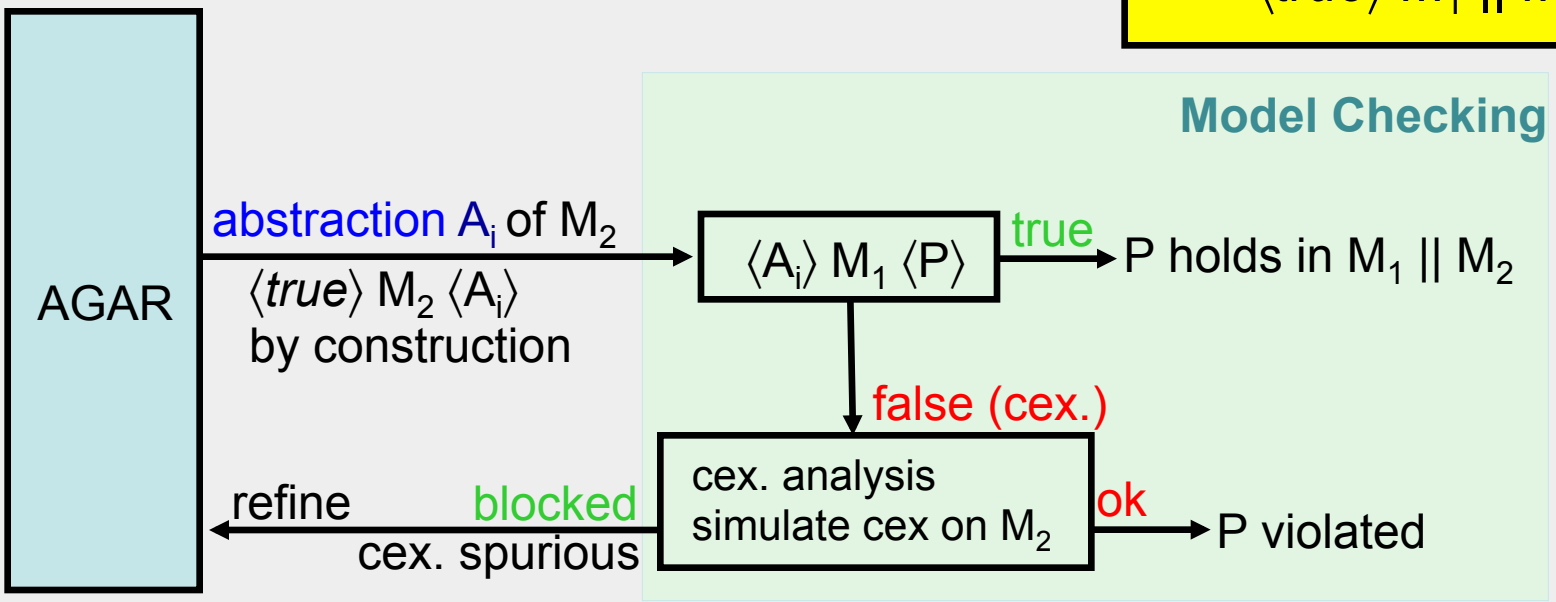
$\langle true \rangle \; M_1 \parallel M_2 \; \langle P \rangle$



- Existential abstraction
  - Maps concrete states in M_2 to abstract states in A
  - Add abstract transition in A if exists "corresponding" concrete transition in M_2
  - $\mathcal{L}(M_2 \uparrow \alpha A) \subseteq \mathcal{L}(A)$
- Premise 2: $\langle true \rangle \; M_2 \; \langle A \rangle$ holds by construction

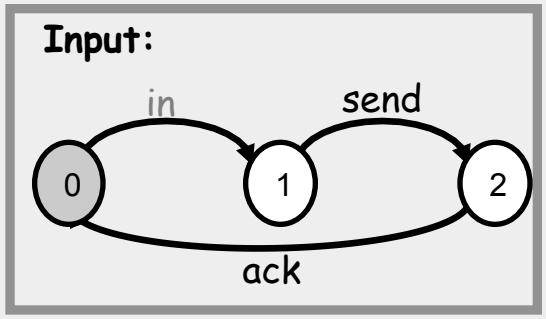
AGAR

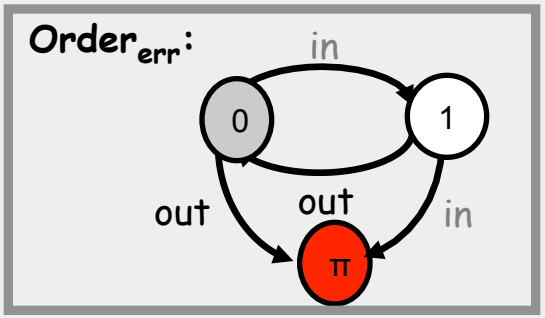
$$\frac{\begin{array}{lll} 1. & \langle A \rangle & M_1 \quad \langle P \rangle \\ 2. & \langle \mathit{true} \rangle & M_2 \quad \langle A \rangle \end{array}}{\langle \mathit{true} \rangle \; M_1 \,||\, M_2 \; \langle P \rangle}$$

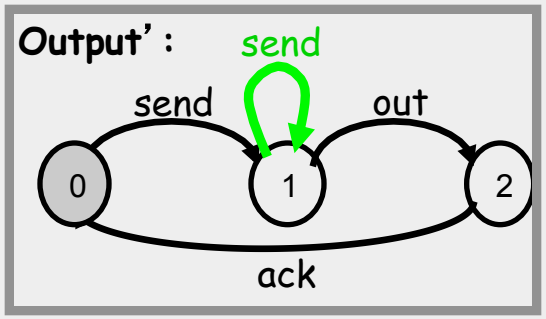


- Variant of CEGAR with differences:
  - use counterexample from M_1 to refine abstraction of M_2
  - A keeps information only about the interface (abstracts away the internals)
- Assumptions and number of iterations bounded by $|M_2|$

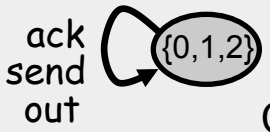
example: AGAR results







A_1:

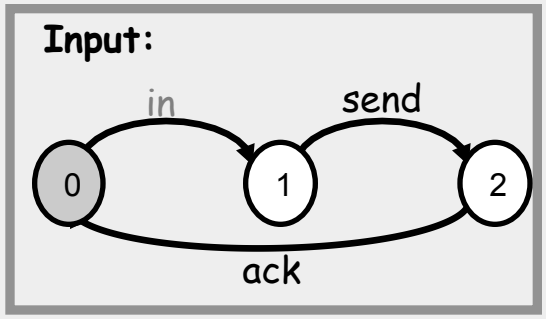


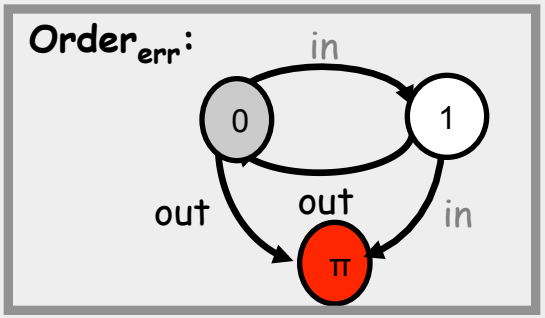
Check: $\langle A_1 \rangle\ M_1\ \langle P \rangle$

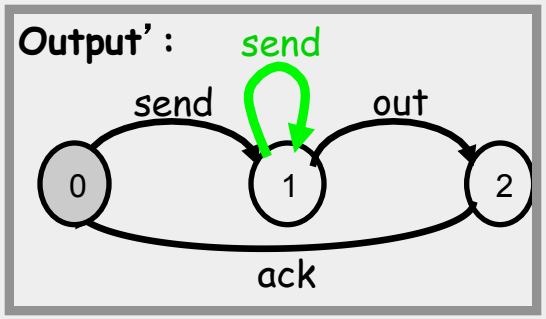
Abstract cex.:

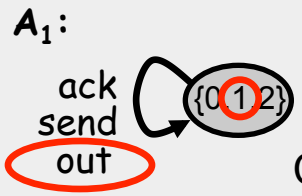
{0,1,2}, out, {0,1,2}

example: AGAR results

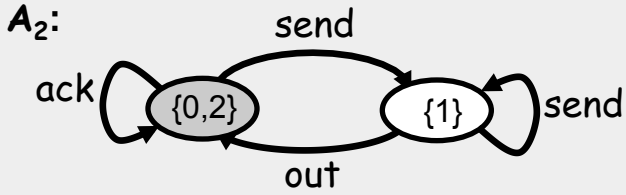




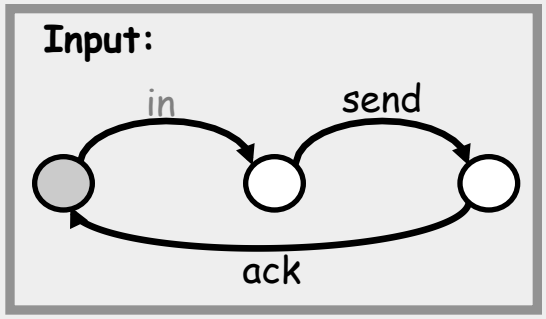


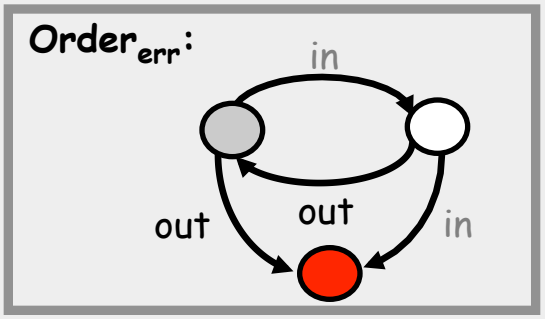


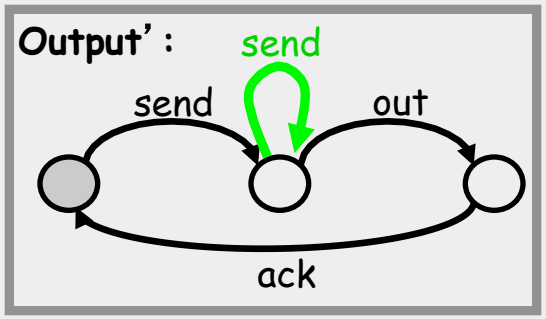
Check: $\langle A_1 \rangle M_1 \langle P \rangle$
Abstract cex.:
{0,1,2}, out, {0,1,2}

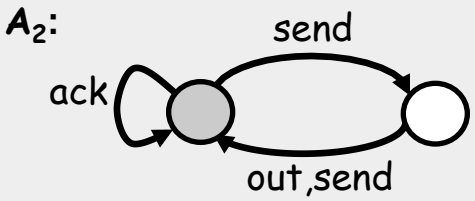


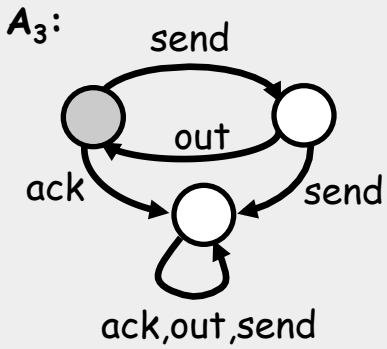
example: learning results

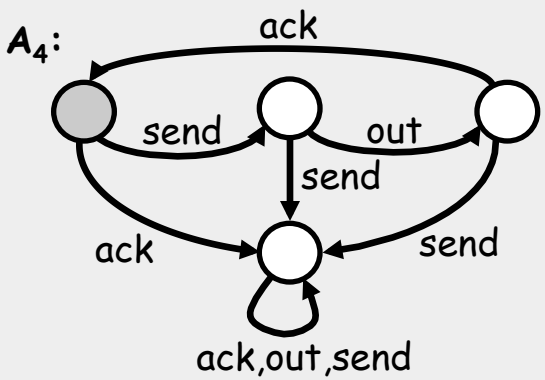












AGAR vs learning

Table 1. Comparison of AGAR and learning for 2 components, with and without alphabet refinement.

Case	k	No alpha. ref.						With alpha. ref.						Sizes	
		AGAR			Learning			AGAR			Learning				
		$\|A\|$	Mem.	Time	$\|A\|$	Mem.	Time	$\|A\|$	Mem.	Time	$\|A\|$	Mem.	Time	$\|M_1 \parallel P_{err}\|$	$\|M_2\|$
Gas Station	3	**16**	**4.11**	**3.33**	177	42.83	–	**5**	**2.99**	**2.09**	8	3.28	3.40	1960	643
	4	**19**	**37.43**	**23.12**	195	100.17	–	**5**	**22.79**	**12.80**	8	25.21	19.46	16464	1623
	5	**22**	**359.53**	**278.63**	45	206.61	–	**5**	216.07	**83.34**	8	**207.29**	188.98	134456	3447
Chiron, Property 2	2	10	**1.30**	**0.92**	9	**1.30**	1.69	10	1.30	**1.56**	**8**	**1.22**	5.17	237	102
	3	36	**2.59**	**5.94**	**21**	5.59	7.08	36	**2.44**	**10.23**	**20**	6.00	30.75	449	1122
	4	160	**8.71**	152.34	**39**	27.1	**32.05**	160	**8.22**	252.06	**38**	41.50	**180.82**	804	5559
	5	4	55.14	–	**111**	**569.23**	**676.02**	3	58.71	–	110	–	386.6	2030	129228
Chiron, Property 3	2	**4**	**1.07**	**0.50**	9	1.14	1.57	4	1.23	**0.62**	**3**	**1.06**	0.91	258	102
	3	**8**	**1.84**	**1.60**	25 n jmj	4.45	7.72	8	**2.00**	3.65	**3**	2.28	**1.12**	482	1122
	4	**16**	**4.01**	**18.75**	45	25.49	36.33	16	**5.08**	107.50	**3**	**7.30**	1.95	846	5559
	5	4	52.53	–	**122**	**134.21**	**271.30**	1	81.89	–	**3**	**163.45**	**19.43**	2084	129228
MER	2	**34**	**1.42**	11.38	40	6.75	**9.89**	**5**	**1.42**	5.02	6	1.89	**1.28**	143	1270
	3	**67**	**8.10**	**247.73**	335	133.34	–	9	11.09	180.13	**8**	**8.78**	**12.56**	6683	7138
	4	58	341.49	–	38	377.21	–	9	532.49	–	**10**	**489.51**	**1220.62**	307623	22886
Rover Exec.	2	**10**	**4.07**	**1.80**	11	2.70	2.35	**3**	2.62	**2.07**	4	2.46	3.30	544	41

AGAR vs learning

- No alphabet refinement:
  - 14 cases, better in 9 (assumption size), 12 (memory consumption), 10 (running time)
- With alphabet refinement:
  - 15 cases: better in 5, 7, 6, respectively

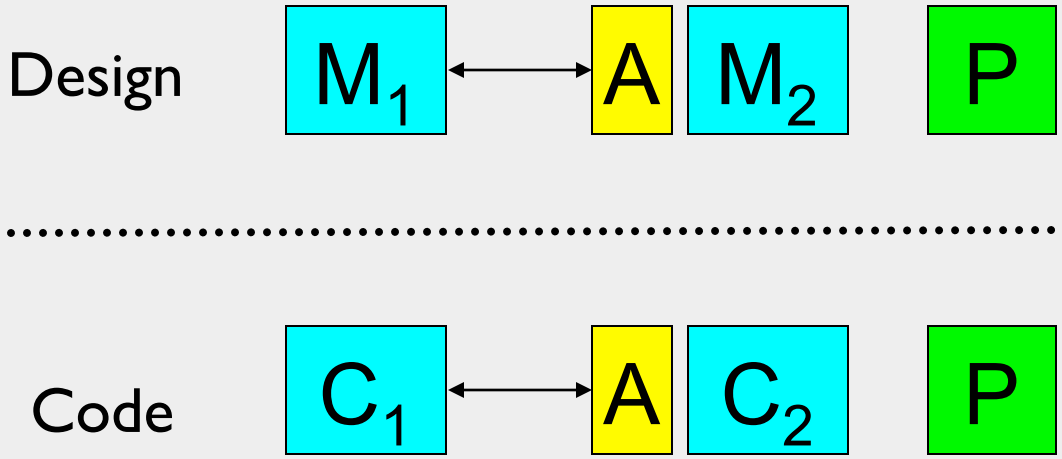
- Problem: unbalanced decompositions
  - Learning exercises more first component
  - AGAR dominated by second component

AGAR vs learning

Balanced decompositions

- No alphabet refinement:
  - 9 cases, better in 8 (assumption size), 9 (memory consumption), 9 (running time)
- With alphabet refinement:
  - 12 cases, better in 7, 7, and 6, respectively

reasoning about code



- Does $M_1 \,||\, M_2$ satisfy P? Model check; **build** assumption A
- Does $C_1 \,||\, C_2$ satisfy P? Model check; **use** assumption A

  [ICSE' 2004] – good results but may not scale

  Solution: replace model checking with testing! [IET Software 2009]

assumption generation for software components

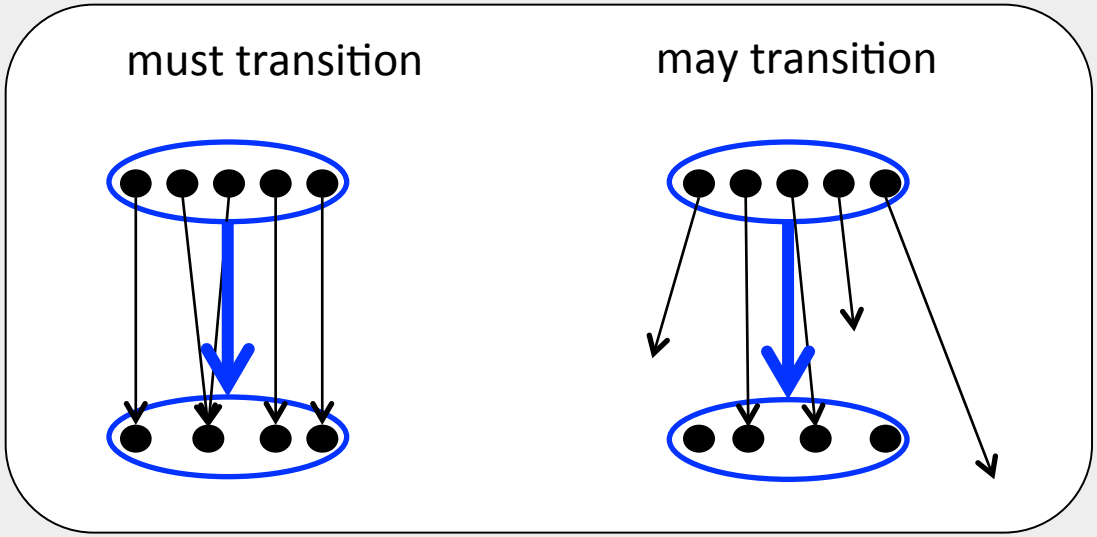
given component C compute weakest assumption (WA):

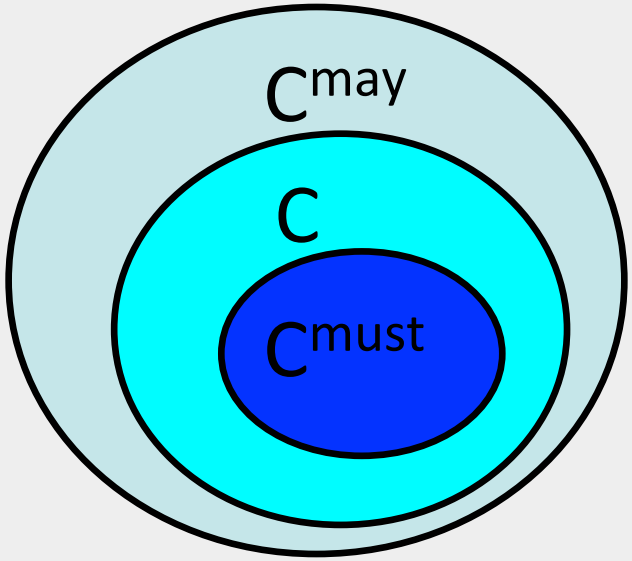
- safe: accept NO illegal sequence of calls
- permissive: accept ALL legal sequences of calls

C is not a model but an actual (infinite-state) implementation

may and must abstraction

- software component C may be *infinite state*
- apply predicate abstraction
- *may abstraction* produces a finite over-approximation
- *must abstraction* produces a finite under-approximation



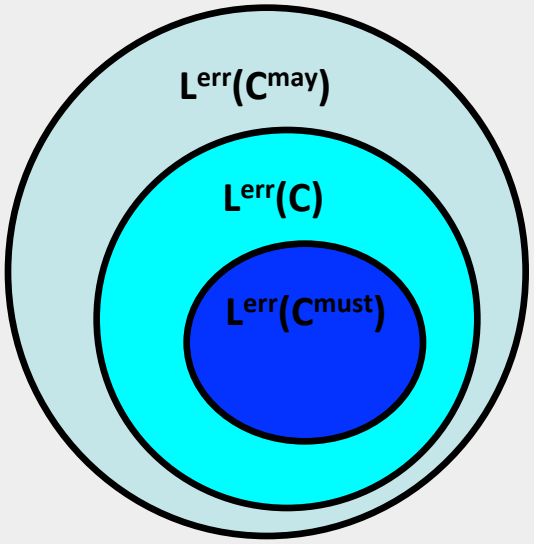


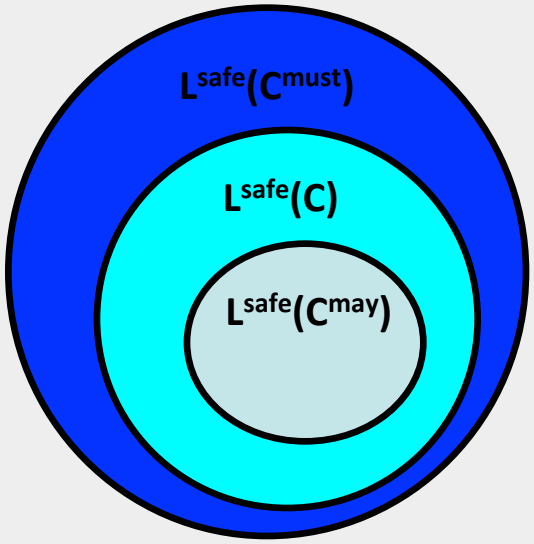
assumption generation for infinite-state components

- $L^{safe}(C)=\overline{L^{err}(C)}$

  Interface A is safe: $L(A) \subseteq L^{safe}(C)$

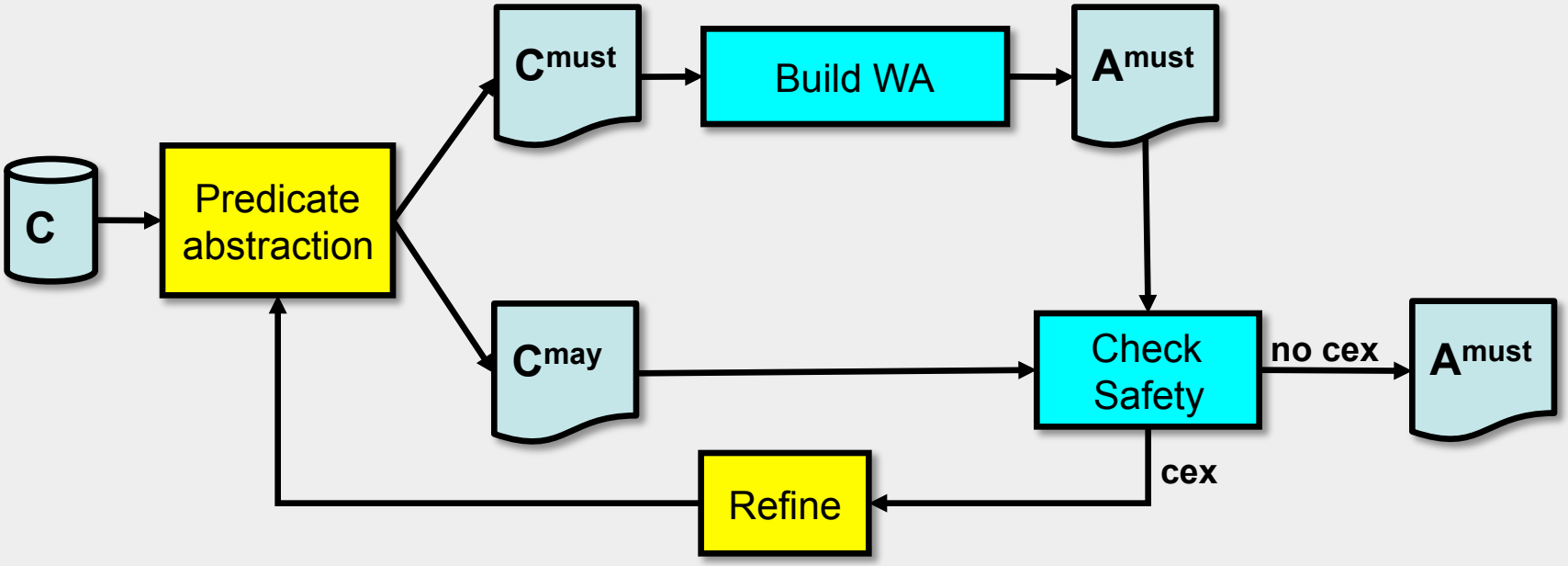
  Interface A is permissive: $L^{safe}(C) \subseteq L(A)$





*An interface **A** permissive w.r.t. **C**'s **must** abstraction and safe w.r.t **C**'s **may** abstraction is safe and permissive for C.*

assumption generation for infinite-state components



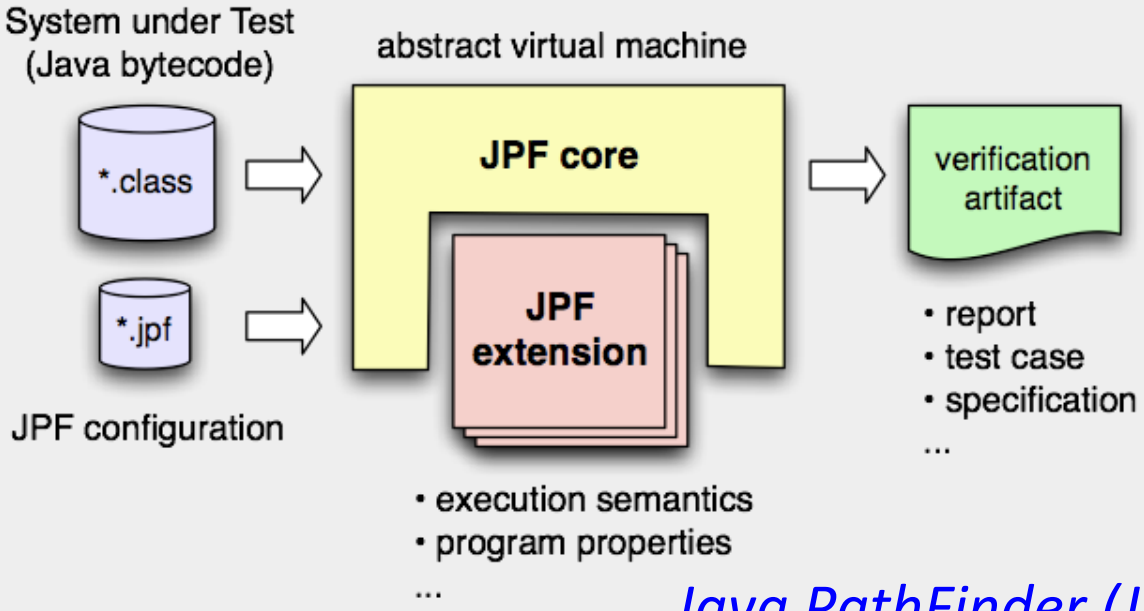
*An interface **A** permissive w.r.t. **C**'s **must** abstraction and safe w.r.t **C**'s **may** abstraction is safe and permissive for C.*

assumption generation for infinite-state components

- conceptually simple
- if concrete component is *deterministic*, so is the must abstraction
- can use learning for building WA
- expensive learning restarts; need of tighter integration of abstraction refinement with L*
- LearnReuse method [CAV'08]
- ARMC model checker: Java2SDK library classes, OpenSSL, NASA CEV model

interface generation in Java PathFinder

- [FASE'09]
- uses L* with heuristic for permissiveness check
- applies to JPF's state-chart extension
- recent work combines learning with symbolic execution [SAS'12]



Java PathFinder (JPF)

compositional verification for probabilistic systems

- labeled transition systems with both probabilistic and non-deterministic behavior
- verification of strong simulation conformance
- counterexamples are probabilistic tree structures
- assume-guarantee abstraction refinement [CAV'12]
- learning from probabilistic tree samples [LICS'12]

further reading

Learning Assumptions for Compositional Verification, *J. M. Cobleigh, D. Giannakopoulou, C. S. Pasareanu*, TACAS'03.

Component Verification with Automatically Generated Assumptions, *D. Giannakopoulou, C. S. Pasareanu, H. Barringer*, jASE'05.

Towards a Compositional SPIN, *C. S. Pasareanu, D. Giannakopoulou*, SPIN'06.

Learning to Divide-and-Conquer: Applying the L* Algorithm to Automate Assume-Guarantee Reasoning, *C. S. Pasareanu, D. Giannakopoulou, M. Gheorghiu Bobaru, J. M. Cobleigh, H. Barringer*, FMSD'08.

Assume Guarantee Testing for Software Components, *D. Giannakopoulou, C. Pasareanu, C. Blundell*, IET Software'08.

Automated Assume-Guarantee Reasoning by Abstraction Refinement, *M. Gheorghiu Bobaru, C. S. Pasareanu, D. Giannakopoulou*, CAV'08.

Assume Guarantee Verification for Interface Automata, *M. Emmi, D. Giannakopoulou, C. S. Pasareanu*, FM'08.

Learning Component Interfaces with May and Must Abstractions, *R. Singh, D. Giannakopoulou, C. Pasareanu*, CAV'10.

Learning Probabilistic Systems from Tree Samples, *A. Komuravelli, C. S. Pasareanu, E. M. Clarke*, LICS'12.

Assume-Guarantee Abstraction Refinement for Probabilistic Systems, *A. Komuravelli, C. S. Pasareanu, E. M. Clarke*, CAV'12.

Symbolic Learning of Component Interfaces, *D. Giannakopoulou, Z. Rakamaric, V. Raman*, SAS'12.

related work

interface automata:

L. de Alfaro, T. A. Henzinger: Interface Automata. ESEC/FSE 2001.

interfaces:

R. Alur, P. Cerný, P. Madhusudan, W. Nam: Synthesis of interface specifications for Java classes. POPL 2005.

T. A. Henzinger, R. Jhala, R. Majumdar: Permissive interfaces. ESEC/SIGSOFT FSE 2005.

learning for AG reasoning:

R. Alur, P. Madhusudan, W. Nam: Symbolic Compositional Verification by Learning Assumptions. CAV 2005.

S. Chaki, E. M. Clarke, N. Sinha, P. Thati: Automated Assume-Guarantee Reasoning for Simulation Conformance. CAV 2005.

J. M. Cobleigh, G. S. Avrunin, L. A. Clarke: Breaking up is hard to do: an investigation of decomposition for assume-guarantee reasoning. ISSTA 2006.

learning for separating automata:

A. Gupta, K. L. McMillan, Z. Fu: Automated Assumption Generation for Compositional Verification. CAV 2007.

Y-F Chen, A. Farzan, E. M. Clarke, Y-K Tsay, B-Y Wang: Learning Minimal Separating DFA's for Compositional Verification. TACAS 2009.

assume-guarantee reasoning for probabilistic systems: many papers from Oxford University

L. Feng, M. Z. Kwiatkowska, D. Parker: Compositional Verification of Probabilistic Systems Using Learning. QEST 2010.

what about shared-memory communication, thread-modular reasoning, Owicki-Gries method?

A. Cohen, K. S. Namjoshi: Local Proofs for Linear-Time Properties of Concurrent Programs. CAV 2008.

conclusion

- techniques for automatic assumption generation and compositional verification
  - finite state models and safety properties
  - learning and abstraction
- data abstraction to deal with software implementations
- promising in practice – when interfaces are small

future

- discovering good system decompositions
- parallelization for increased scalability
- beyond safety: liveness, timed and probabilistic reasoning
- run-time analysis
- make it practical!

thank you!