Assume-Guarantee Reasoning for Deadlock

Sagar Chaki, Nishant Sinha
November 15, 2006

Overview

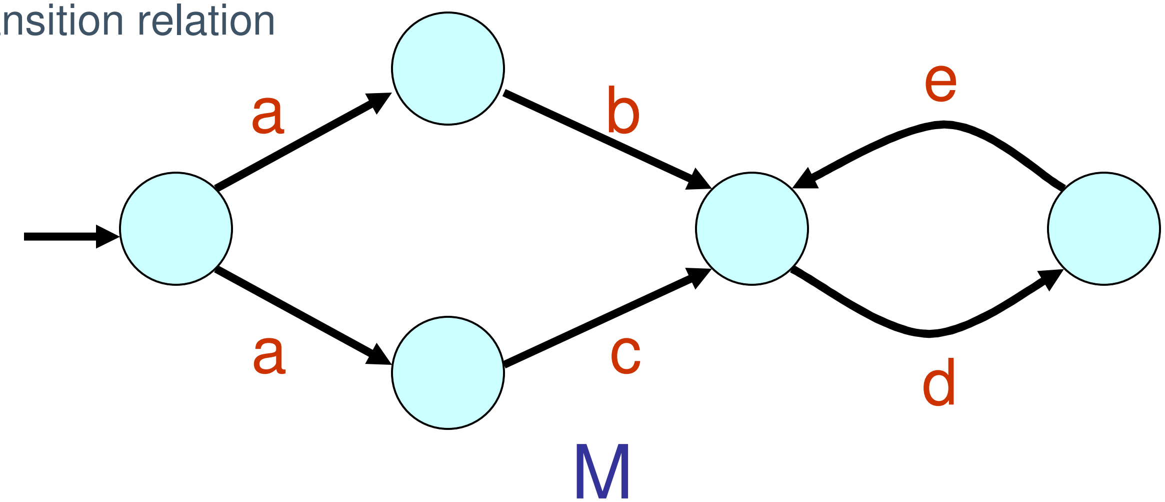
We present a framework that uses **learning** and **automated Assume-Guarantee (AG)** reasoning to detect **deadlocks**

- Concurrent systems with blocking message-passing communication
- Develop a notion of regular failure languages
- Propose a new kind of Failure Automata that accept such languages
- Develop an algorithm L^F to learn deterministic FA that accept an unknown regular failure language
- Use L^F to learn appropriate assumptions for deadlock detection
- Present experimental results

Finite LTS

$M = (Q , I , \Sigma , T)$

- $Q \equiv$ non-empty set of states
- $I \in Q \equiv$ initial state
- $\Sigma \equiv$ set of actions $\equiv$ alphabet
- $T \subseteq Q \times \Sigma \times Q \equiv$ transition relation



$\Sigma(M) = \{a,b,c,d,e,f\}$

Operational Semantics

Components handshake (synchronize) over shared actions

- Else proceed independently (asynchronously)
- CSP semantics

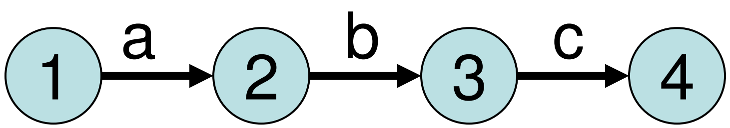
Composition of M_1 & $M_2 \equiv M_1 \,||\, M_2$

- State of $M_1 \,||\, M_2$ is of the form (s_1,s_2) where s_i is a state of M_i

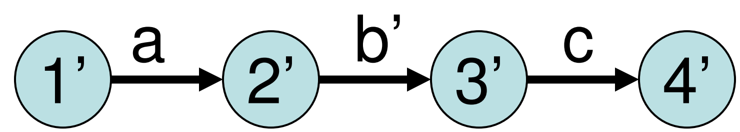
$$\frac{s_1 \xrightarrow{a} s'_1 \quad a \notin \Sigma(M_2)}{(s_1,s_2) \xrightarrow{a} (s'_1,s_2)} \qquad \frac{s_2 \xrightarrow{a} s'_2 \quad a \notin \Sigma(M_1)}{(s_1,s_2) \xrightarrow{a} (s_1,s'_2)}$$

$$\frac{s_1 \xrightarrow{a} s'_1 \qquad s_2 \xrightarrow{a} s'_2}{(s_1,s_2) \xrightarrow{a} (s'_1,s'_2)}$$

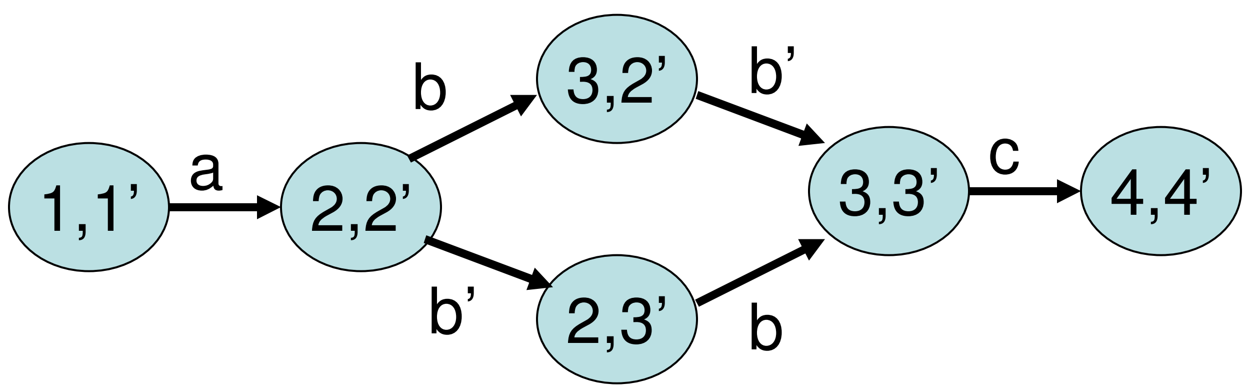
Example



M_1 $\Sigma = \{a,b,c\}$

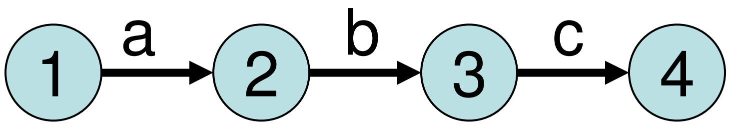


M_2 $\Sigma = \{a,b',c\}$

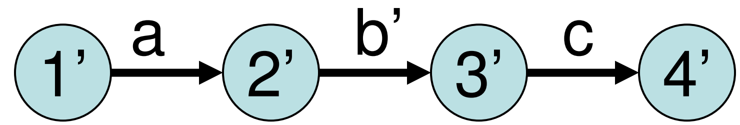


$M_1 \parallel M_2$

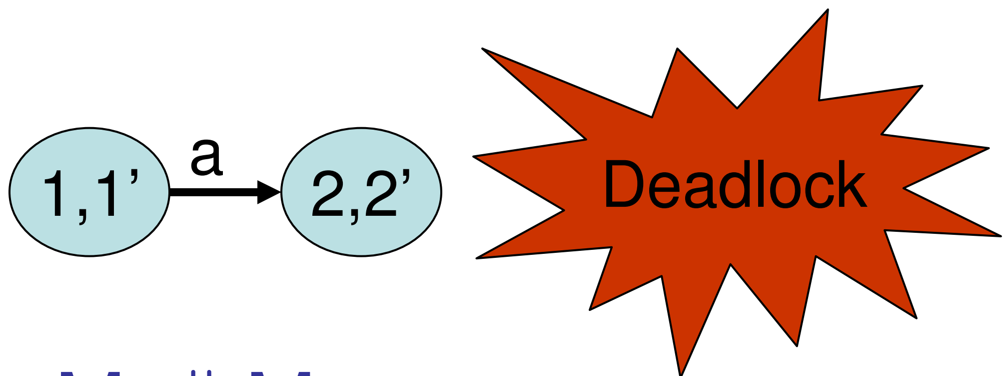
Deadlock



$M_1 \quad \Sigma = \{a,b,b',c\}$

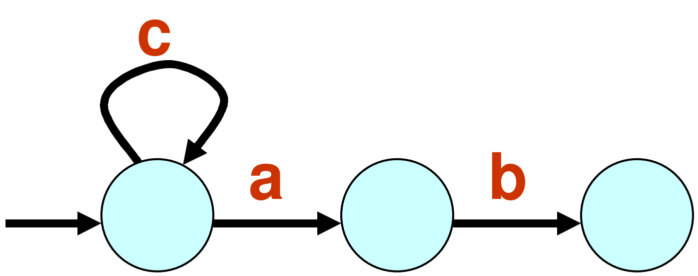


$M_2 \quad \Sigma = \{a,b,b',c\}$

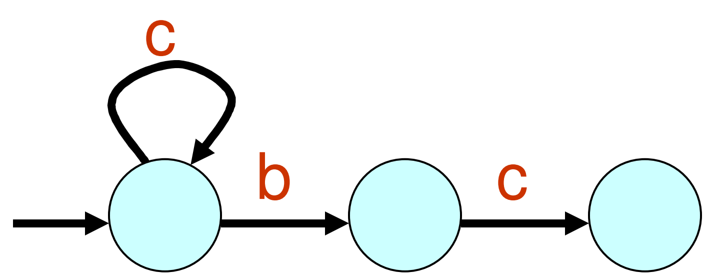


$M_1 \parallel M_2$

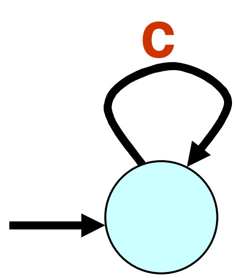
Deadlock and Composition



M_1

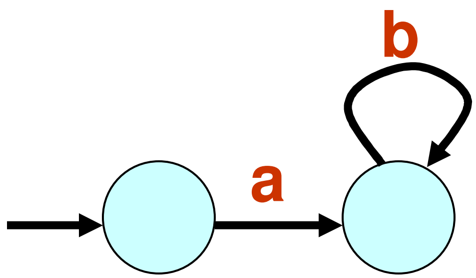


M_2

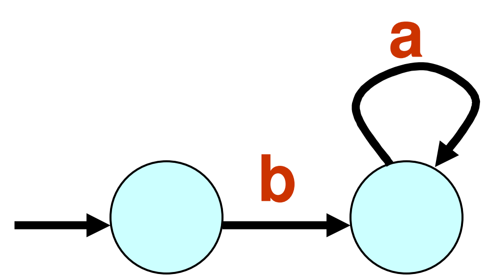


$M_1 \,||\, M_2$

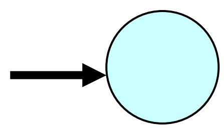
Deadlock and Composition



M_1



M_1



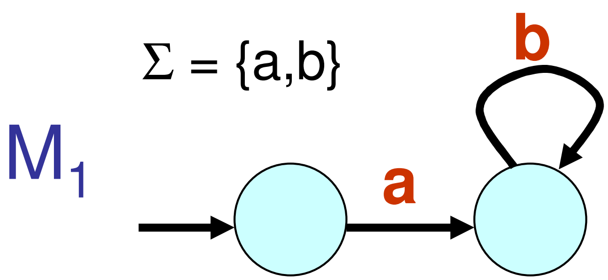
$M_1 \mid\mid M_2$

Failures & Failure Languages

Trace $\in \Sigma^*$ = sequence of actions

Refusal $\subseteq \Sigma$ = set of actions

Failure $\in \Sigma^* \times 2^\Sigma$ = a trace, followed by a refusal

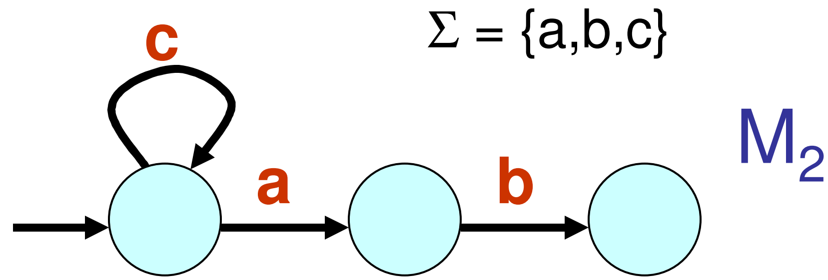


$L(M_1)$ =

{ λ,{b} a,{a}

ab,{a} abb,{a} …}

Downward closed



$L(M_2)$ =

{ λ,{b} c,{b} cc,{b} …

a,{a,c} ab,{a,b,c} }

Deadlock

AG Rule for Deadlock

Consider the following (idea for a) non-circular proof rule

$$\frac{M_1 \parallel A \text{ does not deadlock} \qquad M_2 \leq A}{M_1 \parallel M_2 \text{ does not deadlock}} \quad \text{AG-NC}$$

We are interested in the largest A that satisfies the first premise.

- Under what conditions is such a language uniquely defined?
- What kind of automata accept such languages?
- Can we learn such automata efficiently?

Downward Closed Failure Languages

A failure language L is downward closed if

$$\forall\, t \in \Sigma^*, \forall\, R, R' \in 2^{\Sigma}, (t,R) \in L \wedge R' \subseteq R \Rightarrow (t,R') \in L$$

There is always an unique maximal downward closed A that satisfies the first premise of AG-NC

Clearly, languages accepted by LTSs are downward closed.

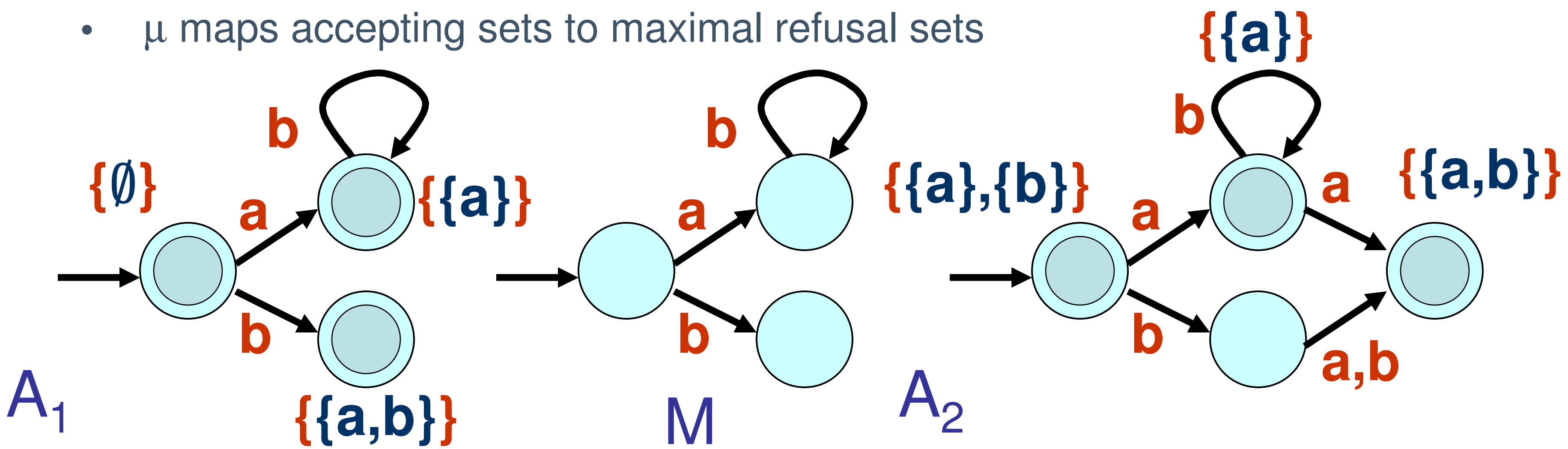
However, the class of languages accepted by LTSs is simply too restricted.

We need automata with more general accepting conditions

Failure Automata (FLA)

A = (Q , I , Σ , T, F, μ)

- Q, I, Σ, T defined as for LTSs
- F ⊆ Q is a set of final or accepting states
- μ maps accepting sets to maximal refusal sets



L(A_1) = L(M)

L(A_2) = maximal A for M

Some Results

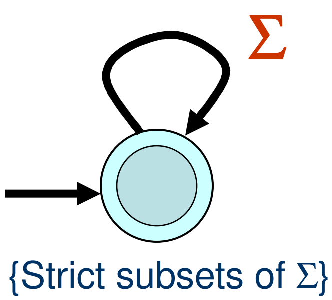
A failure language is regular iff it is accepted by some FLA

- Deterministic FLA have the same accepting power as FLA in general
- Every regular failure language is accepted by a unique minimal DFLA

The maximal language satisfying premise #1 is unique and regular

- Hence accepted by an unique minimal DFLA

Deadlock can be expressed as a regular failure language containment problem: M does not deadlock iff $L(M) \subseteq$ No-DL where No-DL = $(\Sigma^* \times 2^\Sigma) - (\Sigma^* \times \{\Sigma\})$ is the set of all non-deadlocking failures



$$\frac{L(M_1 \parallel A) \subseteq \text{No-DL} \qquad L(M_2) \subseteq L(A)}{L(M_1 \parallel M_2) \subseteq \text{No-DL}}$$

AG-NC

Sound and Complete

Next Steps

We develop a learning algorithm L^F that can learn any unknown regular failure language U

L^F uses a minimally adequate teacher (MAT) that can answer two kinds of queries

- Membership: Given a failure f, does f belong to U?
- Candidate: Given a DFLA C, is L(C) = U? If not, the MAT also returns a counterexample failure in the symmetric difference of L(C) and U

We use L^F to learn the maximal A

- MAT will be implemented via model checking

In case of a deadlock we return a counterexample witness

The Algorithm L^F

Maintains an observation table whose rows are labeled with traces and columns with **failures**. Iteratively does the following:

1) Build the table using membership queries

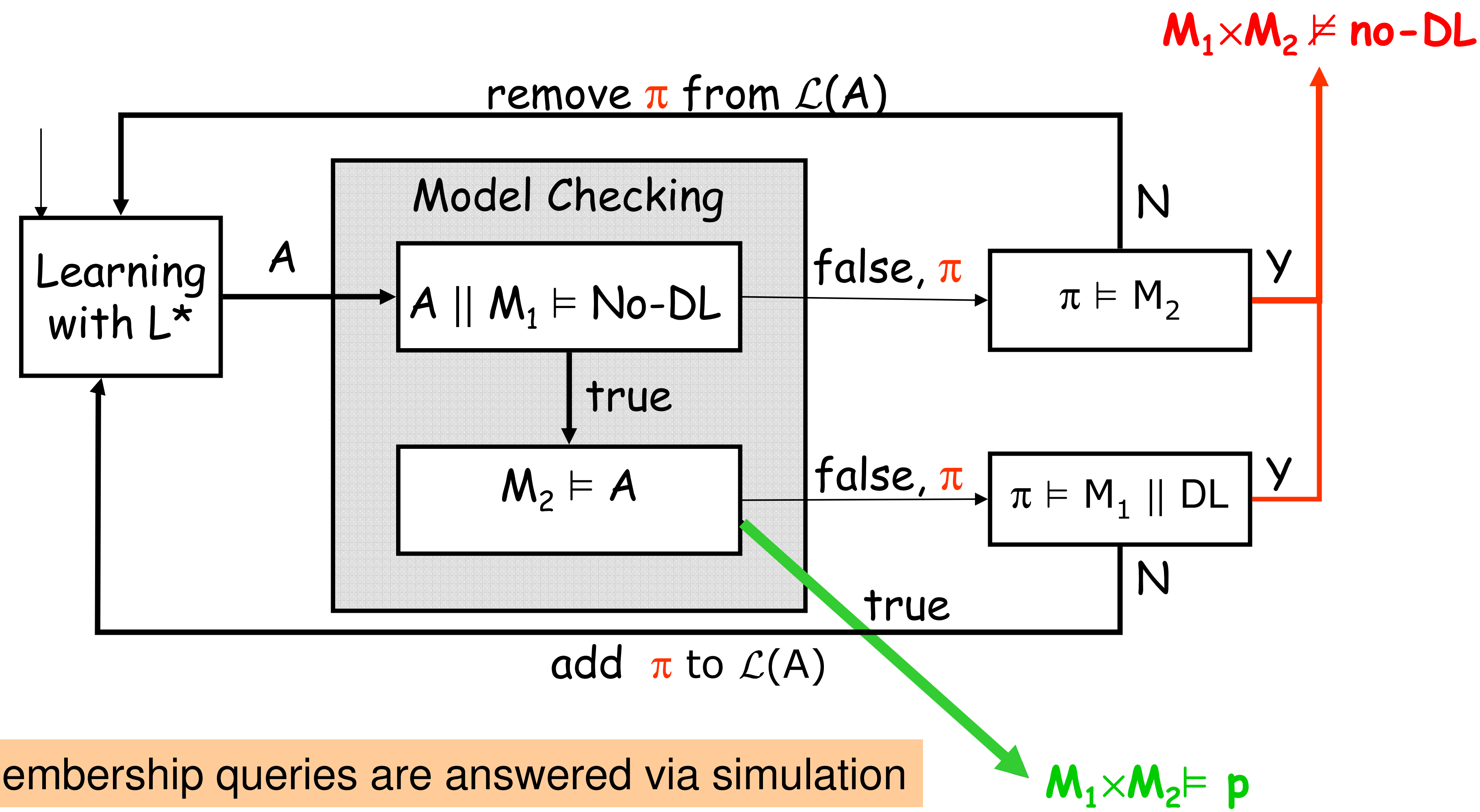
2) Constructs a candidate DFLA C from the table and makes a candidate query with C

3) If candidate query succeeds, returns C as the final answer

4) If candidate query fails, uses the counterexample to construct a new failure f and adds f to the columns of the table. Repeats from Step 1.

The new f added ensures that the number of rows will increase strictly in the next iteration. Number of rows cannot exceed the number of states of the minimal DFLA accepting U. Hence L^F always terminates and moreover, uses polynomial amount of resources.

Overall Deadlock Detection Procedure



Membership queries are answered via simulation

Experimental Setup

Implemented AG-NC as well as the following circular rule:

$$\frac{L(M_1 \parallel A_1) \subseteq \text{No-DL} \qquad L(M_2 \parallel A_2) \subseteq \text{No-DL} \qquad W(A_1) \parallel W(A_2) \subseteq \text{No-DL}}{L(M_1 \parallel M_2) \subseteq \text{No-DL}}$$

Experimented with benchmarks derived from Linux device drivers and Inter-Process Communication library (synchronizing via locks) and Dining Philosophers

2.4 GHz machine with 2 GB RAM limit and 1 hour timeout

Experimental Results: No Deadlock

Exp	Loc	Comp	Non-Circ			Circular		
			T	M	A	T	M	A
MC	7272	2	308	903	5	**307**	903	6
MC	7272	4	*	1453	-	**716**	1453	24
Ide	18905	3	338	50	11	**62**	47	12
Ide	18905	5	*	84	-	**639**	85	48
Synclink	17262	4	1547	19	21	**58**	21	24
Synclink	17262	6	*	27	-	**1815**	189	96
Mxser	15717	3	2079	140	11	**639**	123	12
Mxser	15717	5	-	179	-	**2131**	185	48
Tg3	36774	3	1568	118	11	**406**	111	12
Tg3	36774	6	-	157	-	**3406**	313	96
IPC	818	3	703	338	49	**478**	355	49
DP	82	6	**100**	330	11	286	414	9
DP	109	8	**1551**	565	11	*	1474	-

1 hour timeout; 2 GB memory limit; * = out of resource; - = no data

Experimental Results: Deadlock

Exp	Loc	Comp	Non-Circ			Circular		
			T	M	A	T	M	A
MC	7272	2	386	980	13	313	979	16
MC	7272	4	-	-	-	-	-	-
Ide	18905	3	*	80	-	557	551	125
Ide	18905	5	*	89	-	*	498	-
Synclink	17262	4	127	181	2	133	181	6
Synclink	17262	6	1188	*	-	-	*	-
Mxser	15717	3	657	364	2	630	364	5
Mxser	15717	5	3368	*	-	2276	*	-
Tg3	36774	3	486	393	2	499	393	5
Tg3	36774	6	*	-	-	1954	*	-
IPC	818	3	-	-	-	-	-	-
DP	82	6	-	-	-	-	-	-
DP	109	8	-	-	-	-	-	-

1 hour timeout; 2 GB memory limit; * = out of resource; - = no data

Related Work

Use of learning for automated AG reasoning proposed originally by Cobleigh et al. [TACAS'03] for safety properties

Since been extended to simulation [CAV'05] and the use of symbolic techniques [CAV'05]

Brookes and Roscoe investigate failure semantics and its use for deadlock detection.

Assume-Guarantee reasoning is a rich area, but limited automation

Iterative abstraction-refinement has also been used in the context of compositional deadlock detection [MEMOCODE'03]

Questions?

Software Engineering Institute | Carnegie Mellon

chaki@sei.cmu.edu

nishants@cs.cmu.edu