

# Outline

## *Lecture 1, part 1*

- Motivation
- Model Checking

## ***Lecture 1, part 2***

- State/Event-based software model checking

## *Lecture 2*

- Component Substitutability

# Objectives

- Model checking of realistic software
- Properties involve both **data (states)** and **communication (events)**
  - Specified as (State/Event) LTL formulas
  - **Safety** and **Liveness**
- Communication via **shared actions**
  - **Synchronous** communication
  - **Asynchronous** execution

# Bluetooth L2CAP Spec

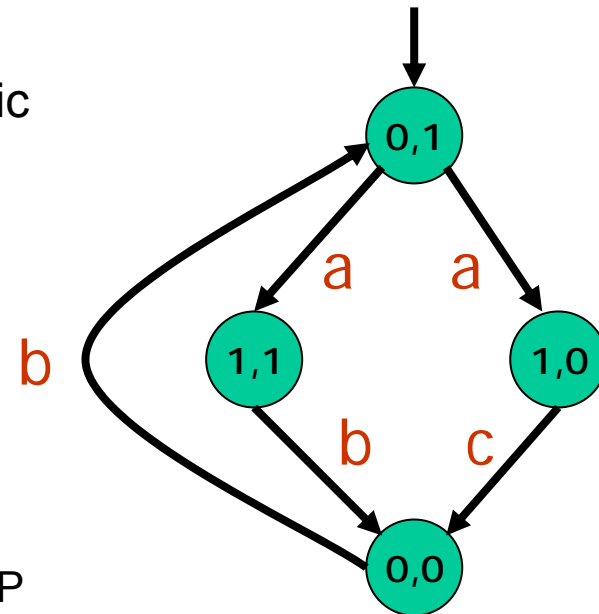
“When an `L2CAP_ConnectRsp` event is received in a `W4_L2CAP_CONNECT_RSP` state, an L2CAP process may send out an `L2CA_ConnectInd` event, disable the RTX timer, and move to state `CONFIG`.”

...

Spec involves both `states` and `events`

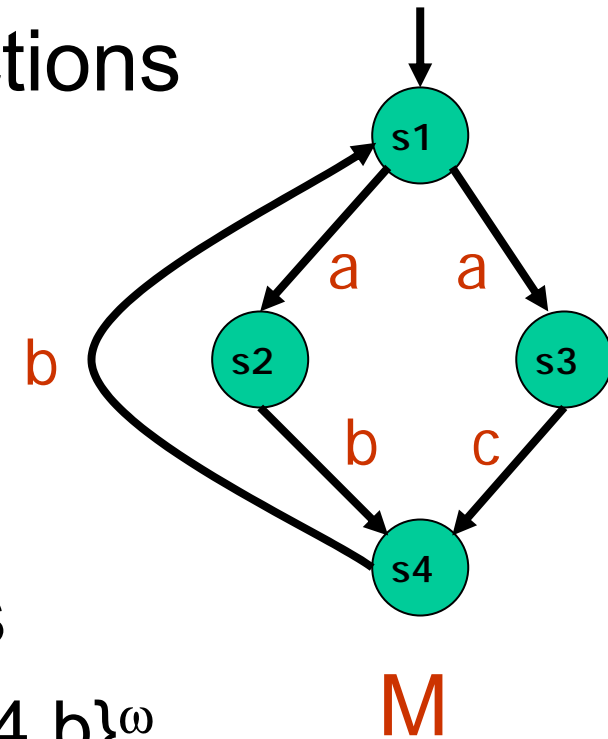
# Labelled Kripke Structures

- Directed graph with labels on **edges and states**,  
(S, Init, P, L, T,  $\Sigma$ , E)
  - Every state is labeled with a set of atomic propositions, P, true in the state
  - Every LKS comes with an alphabet of actions,  $\Sigma$
- State labeling function :  $L: S \rightarrow 2^P$
- Transition labeling function :  $E: T \rightarrow (2^\Sigma \setminus \{\})$ 
  - Assumption: LKSs are deadlock-free [cf. MEMOCODE 04]



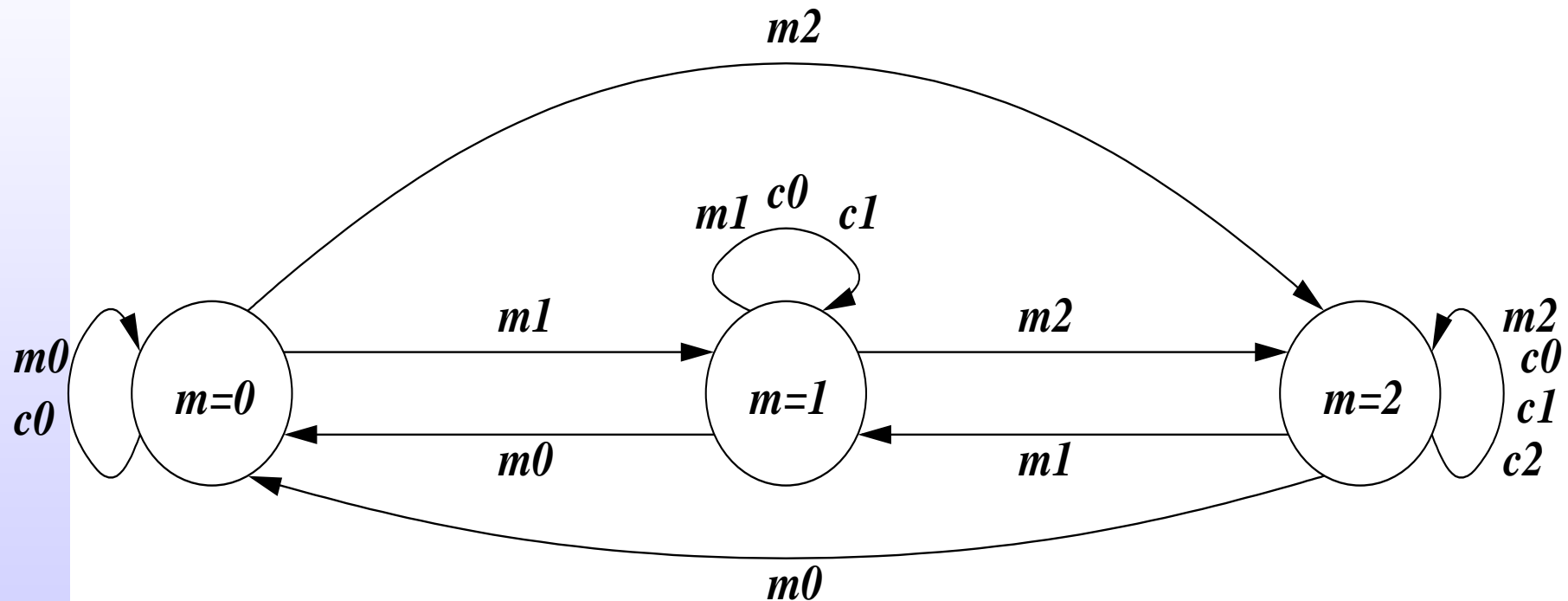
# Traces and Languages

- **Trace:** infinite alternating sequence of states and actions
  - (s1 a s2 b s4 b s1...)



- **Language:** set of all traces
  - $L(M) = \{s1\ a\ (s2\ b\ +\ s3\ c)\ s4\ b\}^\omega$

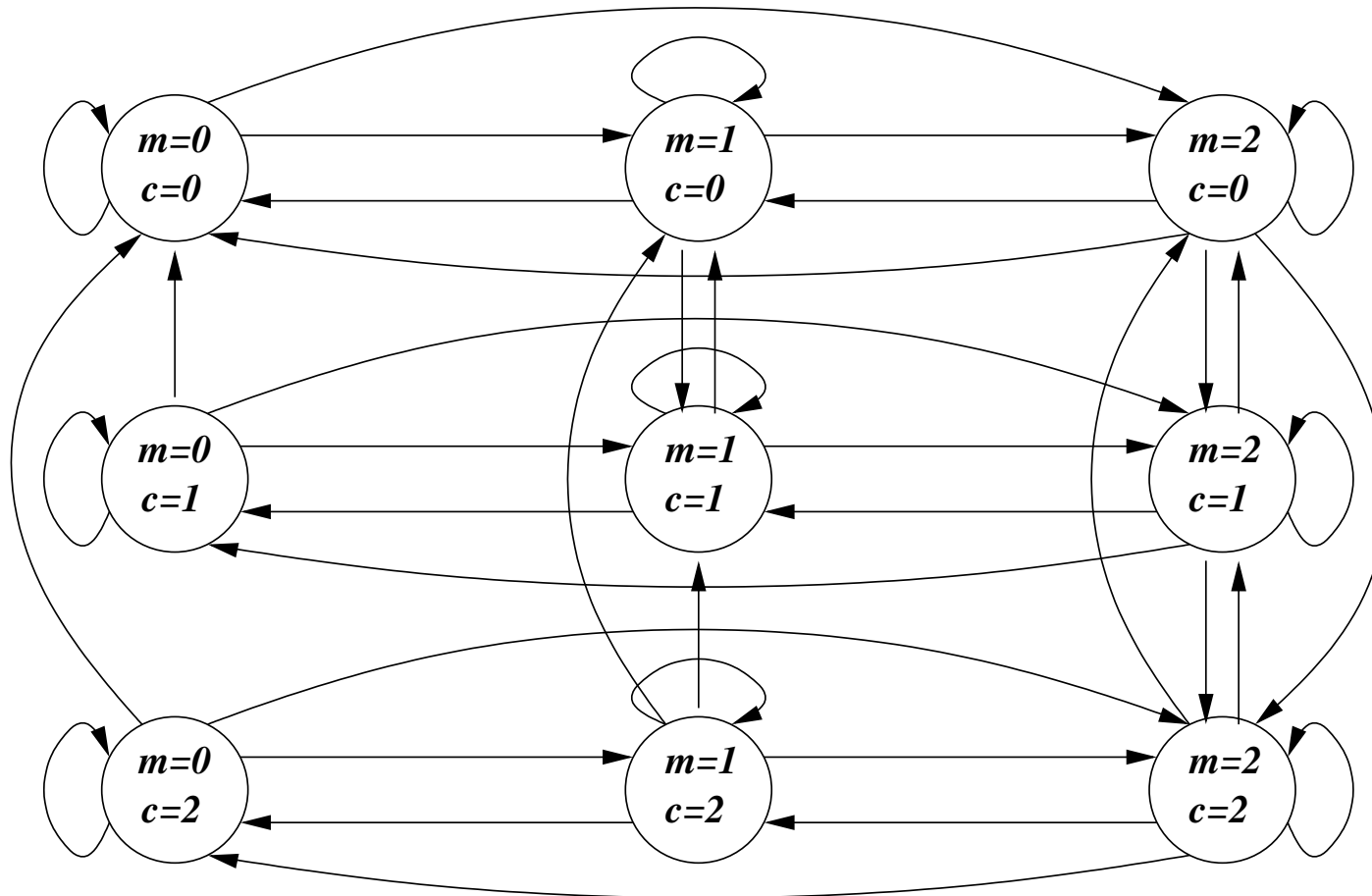
# Surge Protector : State/Event



State/Event model of the Surge Protector

(example is given for  $m: [0..2]$ ,  $c: [0..2]$ )

# Surge Protector : State Only



Kripke structure of the Surge Protector  
(example is given for  $m: [0..2]$ ,  $c: [0..2]$ )

# State/Event LTL

Given LKS  $M = (S, \text{Init}, P, L, T, \Sigma, E)$ , and  $p \in P$ ,  $a \in \Sigma$ ,

$$\varphi ::= p \mid a \mid \sim\varphi \mid \varphi \& \varphi \mid \mathbf{X}\varphi \mid \mathbf{G}\varphi \mid \mathbf{F}\varphi \mid \varphi \mathbf{U} \varphi$$

$\pi = (s_1 a_1 s_2 a_2 \dots)$  is a path and  $\pi^i$  is the suffix of  $\pi$  starting at state  $s_i$

$\pi \models p$  iff  $s_1$  is the first state of  $\pi$  and  $p \in L(s_1)$

$\pi \models a$  iff  $a$  is the first action of  $\pi$

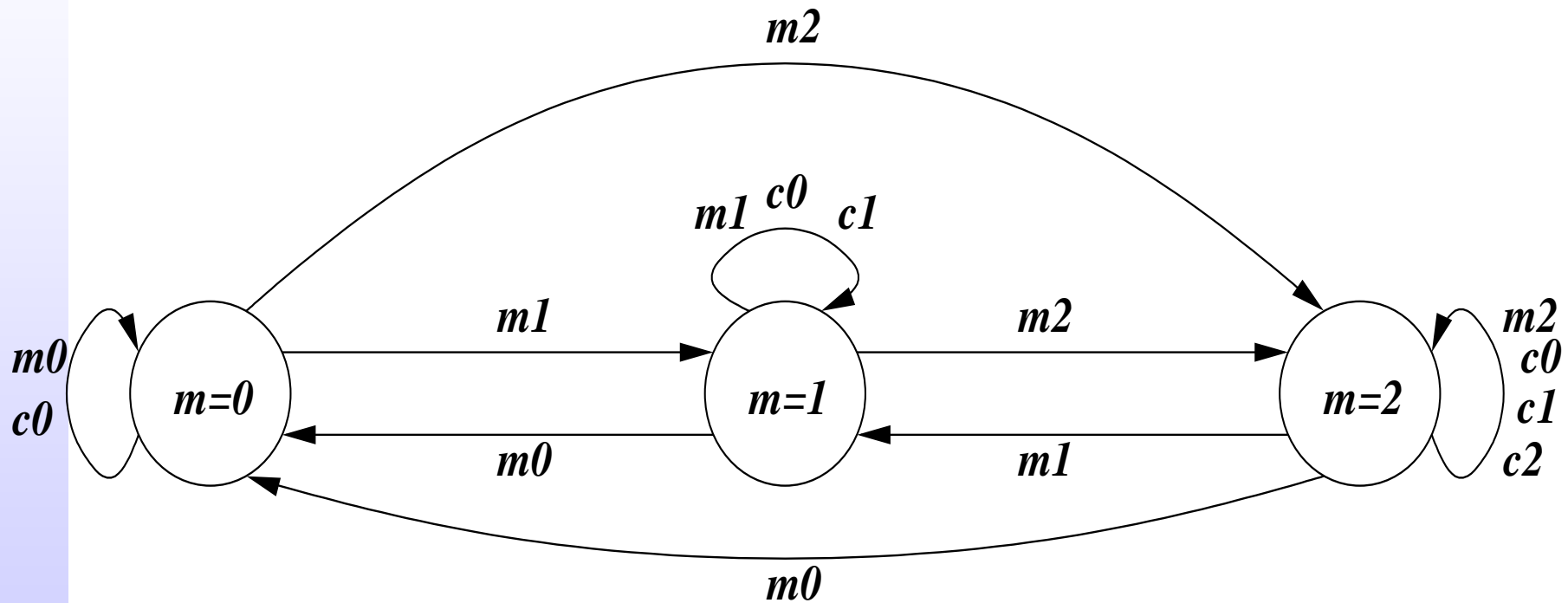
$\pi \models \sim\varphi$  iff  $\sim(\pi \models \varphi)$

$\pi \models \mathbf{X}\varphi$  iff  $\pi^2 \models \varphi$

$\pi \models \varphi_1 \mathbf{U} \varphi_2$  iff there is some  $i \geq 1$  such that  $\pi^i \models \varphi_2$   
and for all  $i \geq j \geq i-1$ ,  $\pi^j \models \varphi_1$

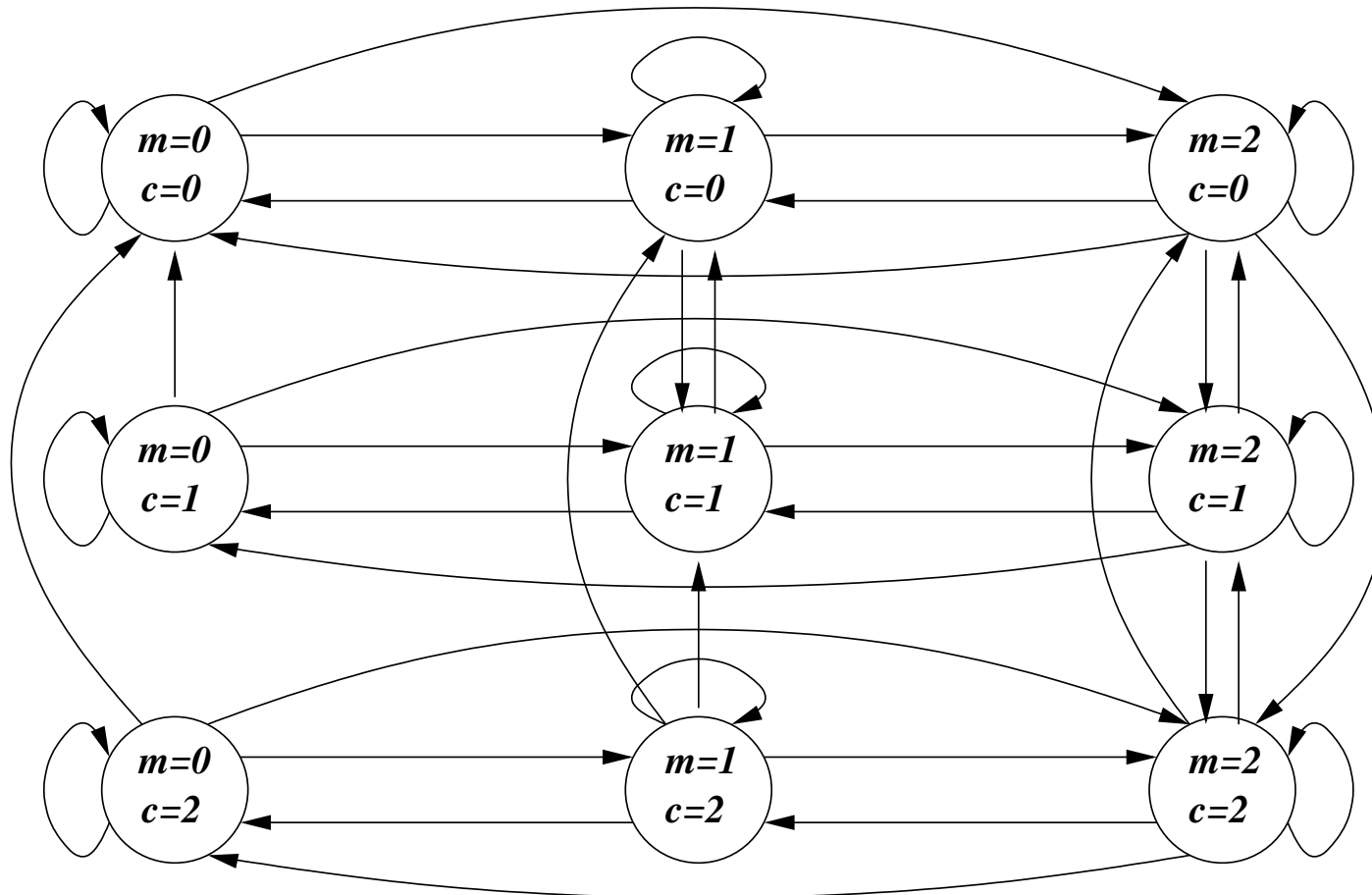
$M \models \varphi$  iff, for every path  $\pi \in L(M)$ ,  $\pi \models \varphi$

# Surge Spec : State/Event



**G** (( $c2 \rightarrow m=2$ ) & ( $c1 \rightarrow (m=1 \vee m=2)$ ))

# Surge Spec : State Only



**G** (((c=0 V c=2) & X (c=1)) → (m=1 V m=2)) &

**G** (((c=0 V c=1) & X (c=2)) → m=2)

# Surge Protector Verification

- Changes of current beyond threshold are disallowed
- State/Event Formula:  $\mathbf{G} ((c2 \rightarrow m=2) \ \& \ (c1 \rightarrow (m=1 \vee m=2)))$
- State Formula:  $\mathbf{G} (((c=0 \vee c=2) \ \& \ \mathbf{X} (c=1)) \rightarrow (m=1 \vee m=2))$   
 $\ \& \ \mathbf{G} (((c=0 \vee c=1) \ \& \ \mathbf{X} (c=2)) \rightarrow m=2)$
- Event Formula:  $\mathbf{G} (m0 \rightarrow ((\sim c1 \ \mathbf{W} (m1 \vee m2) ) ) ) \ \&$   
 $\ \mathbf{G} (m0 \rightarrow ((\sim c2 \ \mathbf{W} m2)) \ \&$   
 $\ \mathbf{G} (m1 \rightarrow ((\sim c2 \ \mathbf{W} m2))$

# Automata-based Verification

*Given:*  $M$  – LKS over  $\Sigma, P$   
 $\varphi$  – SE/LTL formula

*How to check:*  $M \models \varphi$

*Possible Approach:*

1. Convert  $M$  into a conventional state-only Kripke structure
2. Convert  $\varphi$  into a state-only LTL formula
3. Check whether  $M \models \varphi$

*Inefficient!*

*What we do:*

1. Interpret  $\varphi$  as an LTL formula over  $\Sigma \cup P$
2. Compute  $B_{\sim\varphi}$  (using Wring [Somenzi, Bloem'00])
3. Construct  $M \otimes B_{\sim\varphi}$  (result is a Buchi automaton)
4. **Theorem:** We have  $L(M \otimes B_{\sim\varphi}) = \{\}$  iff  $M \models \varphi$

*No extra cost in time or space!*

# Abstraction

- Program **Statements** , **States**
- **Control flow** , **Transitions**
- **Transitions** depict **observable behavior**
- **Automated**

# Predicate Abstraction into LKS

```
void OSSemPend(...) {
```

```
  L1: lock = 1;
```

```
  if (x < y) {
```

```
    L2: lock = 0;
```

```
    ...
```

```
  }
```

```
  if (x >= y) {
```

```
    ...
```

```
    L3: lock = 0;
```

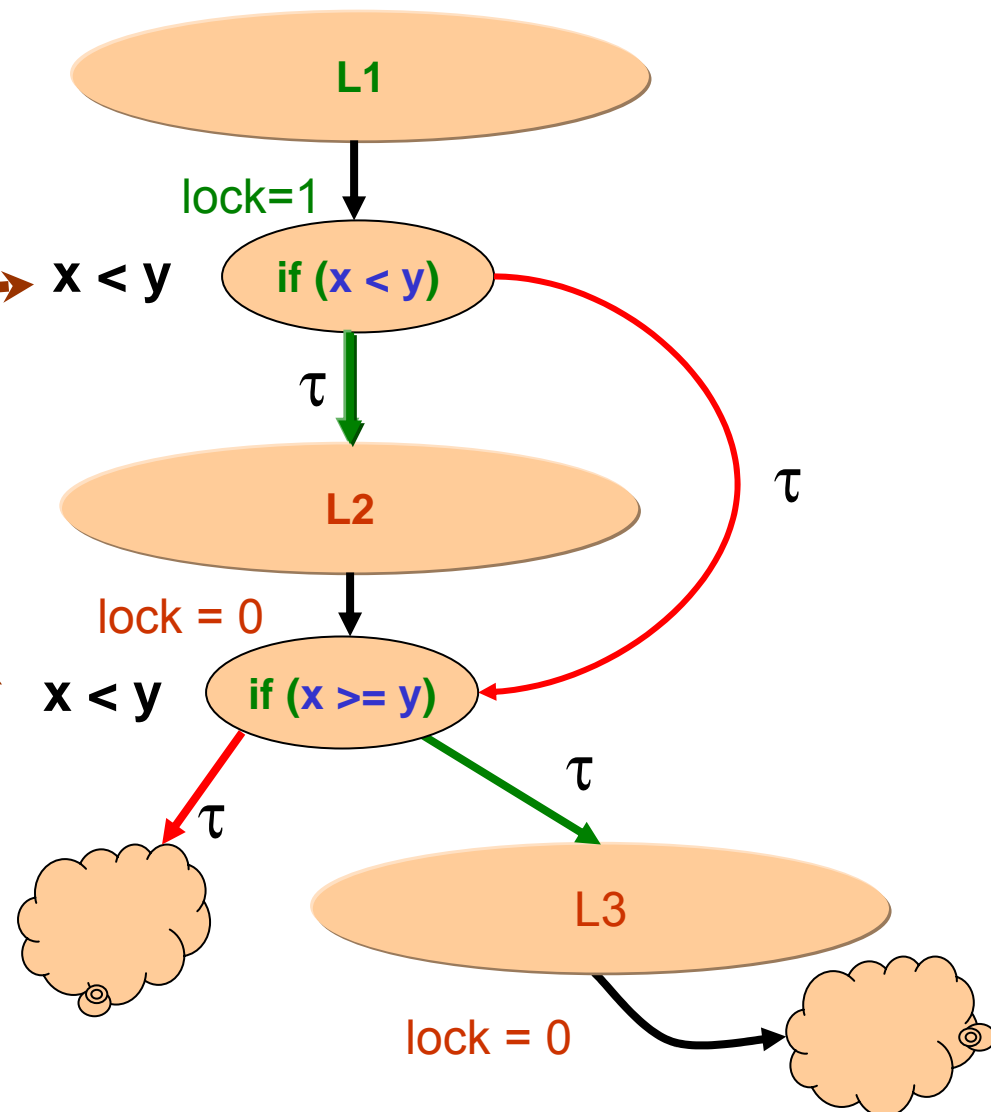
```
    ...
```

```
  } else {
```

```
    ...
```

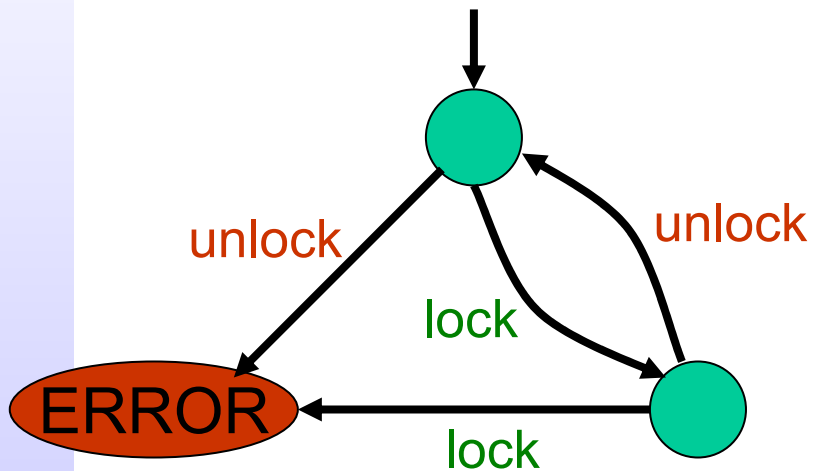
```
  }
```

```
}
```

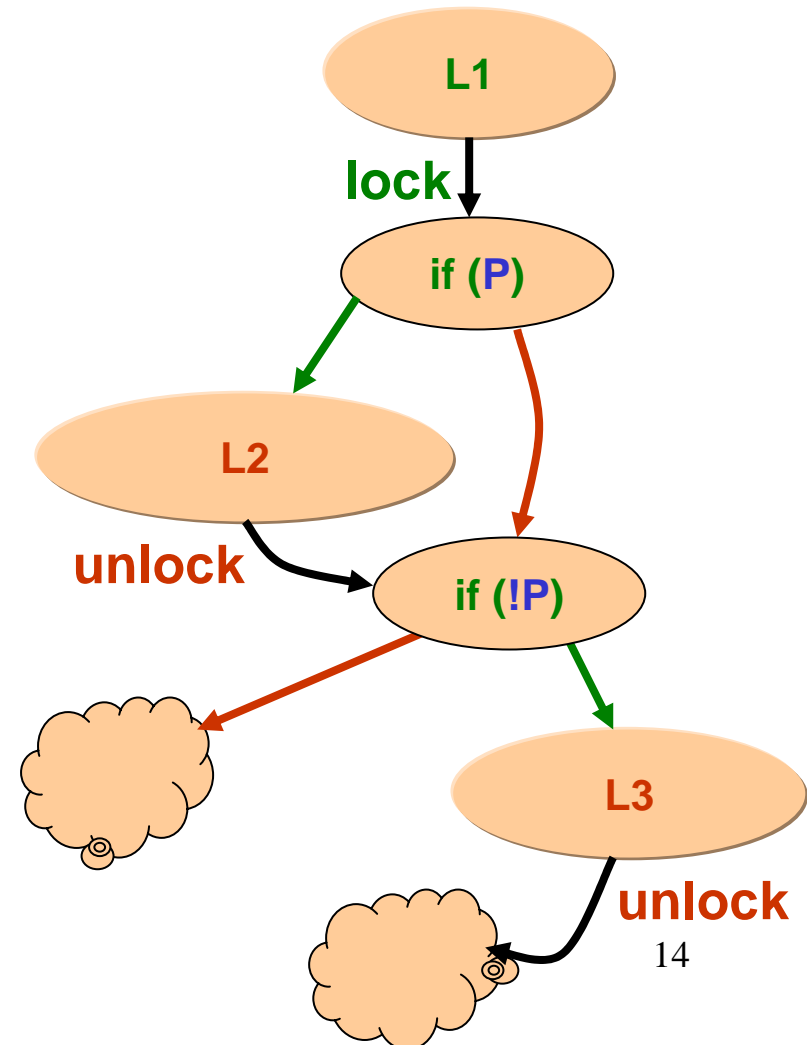


# Verification

## Specification

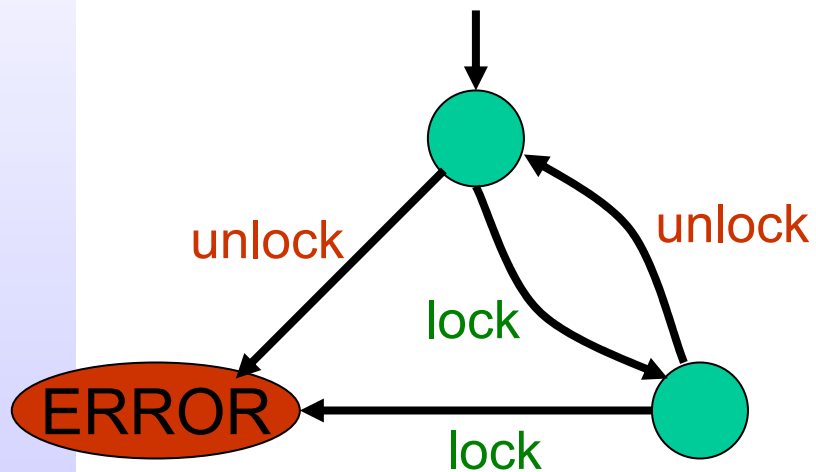


## Model

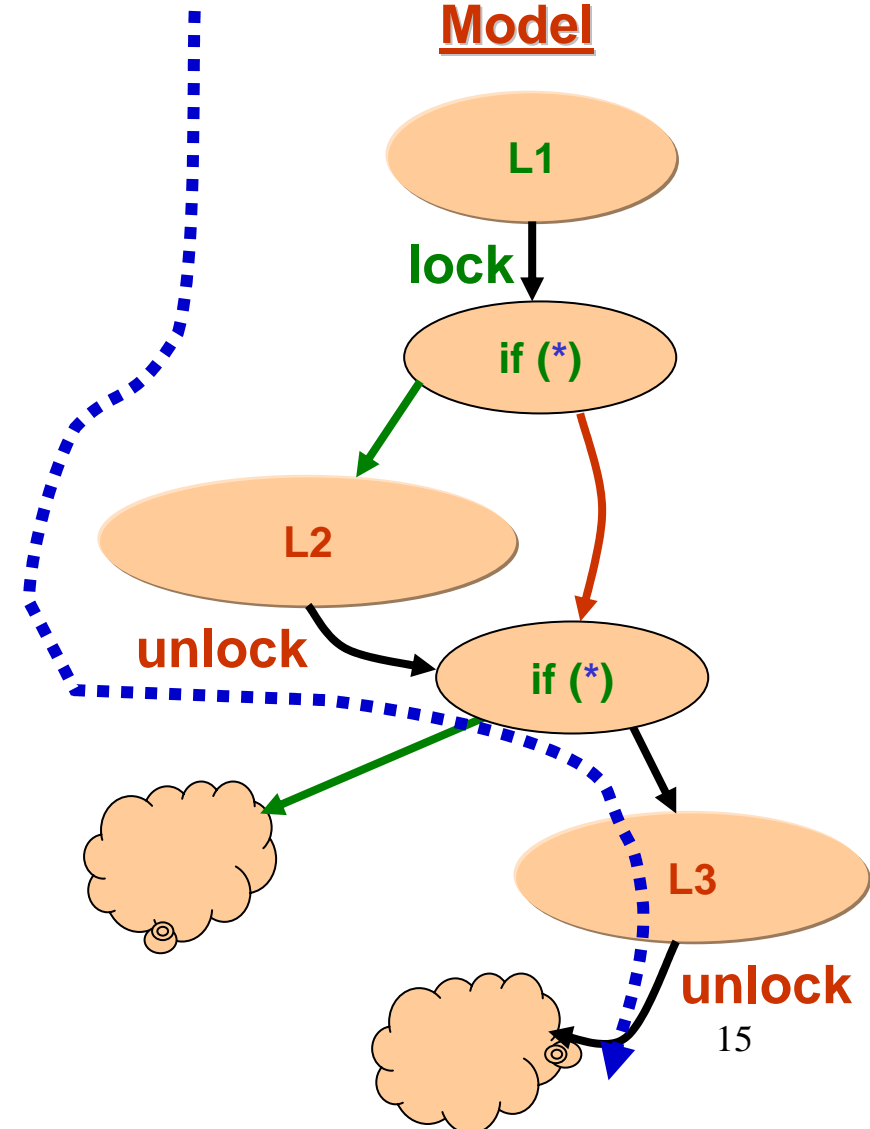


# Verification

## Specification

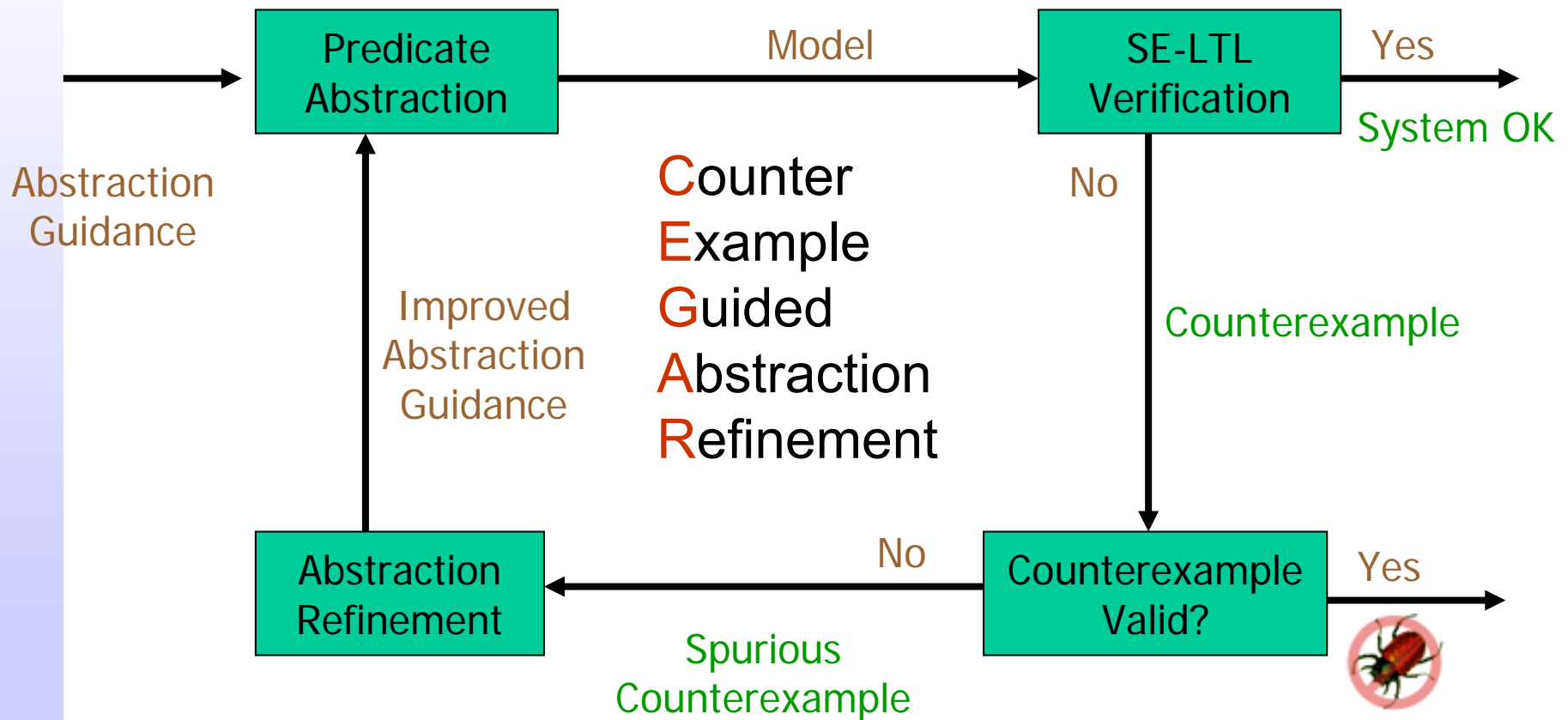


## Model



# ComFoRT: Component Formal Verification Framework

C programs/  
CCL Programs/  
Components Assembly



# Verification Results

Current range	State Formula				Event Formula				State/Event Formula			
	Aut. Size		Time		Aut. Size		Time		Aut. Size		Time	
	St	Tr	BC	MC	St	Tr	BC	MC	St	Tr	BC	MC
2	4	5	0.25	0.383	6	10	0.245	0.32	3	4	0.184	0.252
4	14	23	0.49	1.141	20	41	1.597	1.77	5	8	0.243	0.391
6	32	57	2.43	4.818	42	92	12.08	12.66	7	12	0.614	0.962
8	58	107	17.5	24.60	72	163	372.8	374.17	9	16	2.622	3.133
10	92	173	196	214.0	X	X	X	X	11	20	33.56	34.5
12	X	X	X	X	X	X	X	X	13	24	534.9	536.4
13	X	X	X	X	X	X	X	X	X	X	X	X

# Parallel Composition

- Components **synchronize** on shared actions
  - proceed **independently** on local actions
- **Propositions** of components are **disjoint** (no shared variables)

# Operational Semantics

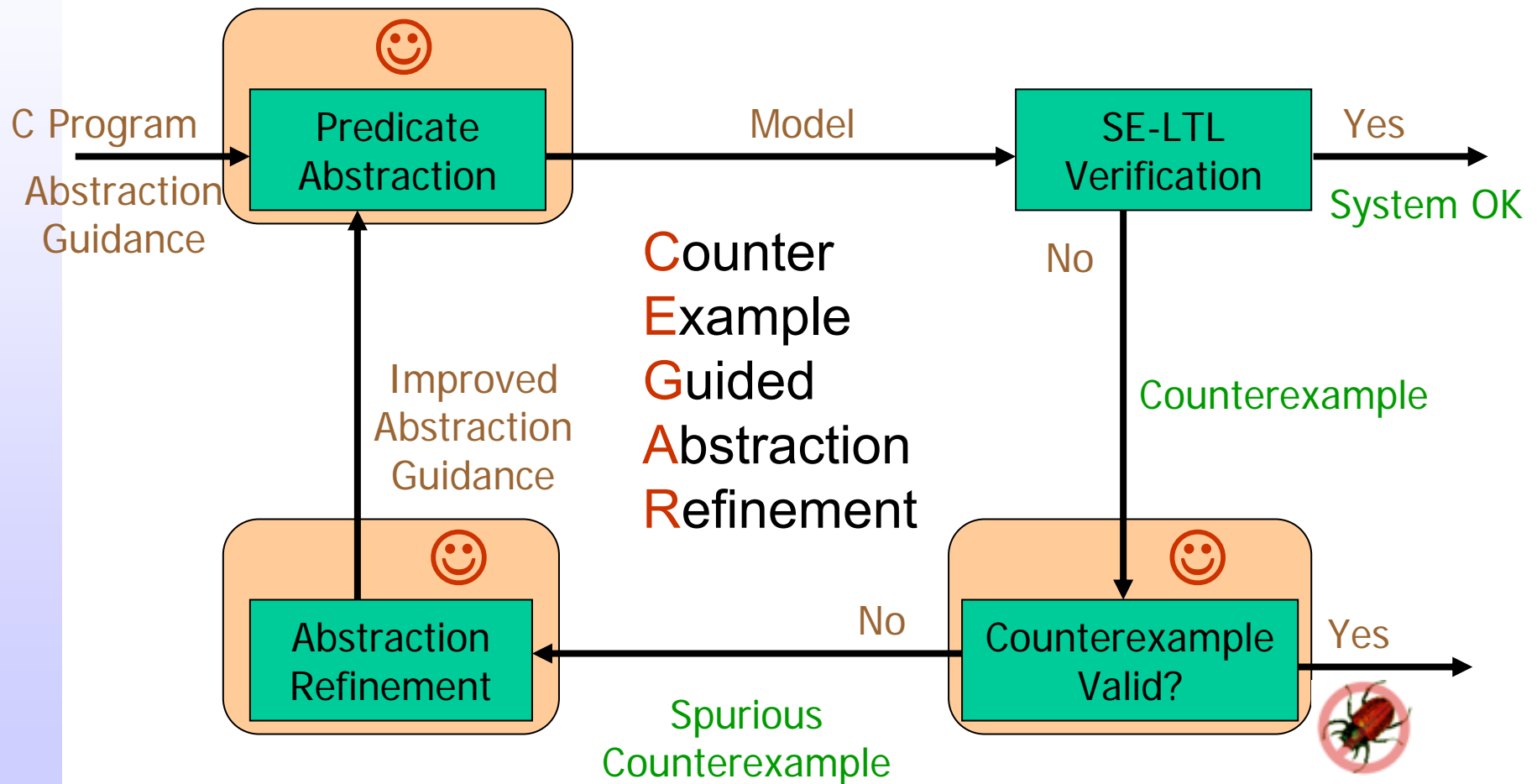
$$\frac{M_1 \xrightarrow{a} M_1' \quad M_2 \xrightarrow{a} M_2'}{(M_1 \parallel M_2) \xrightarrow{a} (M_1' \parallel M_2')}$$

Synchronization on  
Shared action

$$\frac{M_1 \xrightarrow{a} M_1' \quad a \notin \Sigma(M_2)}{(M_1 \parallel M_2) \xrightarrow{a} (M_1' \parallel M_2)}$$

Asynchronous  
Execution

# Compositional Verification



# Case Studies

- **MicroC/OS-II**
  - **Real-time** OS for **embedded** applications
  - Widely used (cell phones, medical devices, routers, washing machines...)
  - 6000+ LOC
  - Verified **locking discipline**
    - Locks and Unlocks **alternate** and locks are **eventually** released
  - Found **four bugs**
    - Missing **unlock** and **return** (three known – one unknown)

# Results

Name	St-B	Tr-B	St-Mdl	T-BA	T-Mdl	T-Ver	T-Tot	Mem
SS	25	47	7951	0.690	48.8	6.84	56.9	39.3
SE	20	45	4331	0.497	18.8	2.92	22.8	24.2
SS	25	47	7574	0.699	43.6	1.65	46.4	38.1
SE	18	40	3691	0.407	15.3	1.089	17.3	21.2
SS	25	47	24.8 M	0.874	65.6	X	X	851
SE	20	45	13.6M	0.655	33.1	2.17	2207	162
SS	25	47	32.6M	0.836	66.0	X	X	347
SE	18	40	15.9M	0.713	34.6	4149	4185	321
BUG	8	14	873	0.205	3.41	0.261	3.88	X

# Case Studies

- **ABB IPC Module**
  - Deployed by a world leader in robotics engineering systems
  - 1500+ LOC
  - 4 components
  - Over 30 billion states after predicate abstraction
- Discovered **synchronization bug** in a matter of hours
  - Process can incorrectly block while writing to a queue
  - Undetected despite seven years of testing/industrial use

# Case Studies

- **Controller for a metal casting plant**, used by Alcoa
  - ~30,000 LOC
  - Verified proper sequencing of various stages of casting:
    - Sequencing happens in prescribed order
    - The next stage eventually gets sequenced
- No **bugs** found... yet.

# Key Contributions

- **State/Event**-based modeling and specification
- **Efficient** (direct) model checking algorithm
- **CEGAR** loop for software systems
  - **Safety** and **liveness** properties
- **Compositional** software verification
  - Component-wise **abstractions**
  - Component-wise counterexample **validation**
  - Component-wise **refinements**

# References

- **State/Event-based Software Model Checking**, In Proceedings of *IFM (Integrated Formal Methods) 2004 International Conference*, LNCS 2999, p. 128 - 147 (with Sagar Chaki, Edmund Clarke, Joel Ouaknine, and Nishant Sinha), **Best Paper Award**.
- **Automated, compositional and iterative deadlock detection**, In Proceedings of *the Second ACM-IEEE International Conference on Formal Methods for Codesign (MEMOCODE) 2004*, (with Sagar Chaki, Edmund Clarke, and Joel Ouaknine).
- **Overview of ComFoRT, A Model Checking Reasoning Framework**, Technical Report CMU/SEI-2004-TN-0181, (with James Ivers).
- **An Expressive Verification Framework for State/Event Systems**, TR CMU-CS-04-45, (with Sagar Chaki, Edmund Clarke, Orna Grumberg, Joel Ouaknine, Tayssir Touili, and Helmut Veith).

# References

- **Lessons Learned from Model Checking a NASA Robot Controller**, *Formal Methods in System Design journal*, September 2004.
- **Development of Verifiable Programs - Application of an Approach based on Executable Design Specifications**, The University of Texas at Austin, Department of Computer Sciences, *Technical Report TR-02-16*, 2002 (with James C. Browne, and Delbert Tesar).
- **A Combined Testing and Verification Approach for Software Reliability**, In the *FME 2001: Formal Methods for Increasing Software Productivity: International Symposium of Formal Methods Europe, Berlin, Germany, March 12-16, 2001*, LNCS 2021, p. 611 - 628 (with Doron Peled).
- **A Formal Object-Oriented Analysis for Software Reliability**, In the *Fundamental Approaches to Software Engineering : 4th International Conference, FASE 2001 : Held as Part of ETAPS 2001, Genova, Italy, April 2-6, 2001*, LNCS 2029, p. 318 - 322 (with James C. Browne, and Robert Kurshan).

# Related Work

- Modal **mu-calculus** [Kozen 83]
- **Doubly labeled** transition systems [De Nicola and Vaandrager 95]
- **CTL-based** verification of doubly labeled transition systems [Gnesi et al. 96]
- **State/event** framework for **three-valued logic** verification [Huth et al. 01]
- **SLAM** [Ball et al. 00-...]
- **BLAST** [Henzinger et al. 02-...]

Questions?