

Formal Verification by Model Checking

Natasha Sharygina
Carnegie Mellon University

*Guest Lectures at the Analysis of Software Artifacts
Class, Spring 2005*

Outline

- Lecture 1:* Overview of Model Checking
- Lecture 2:* Complexity Reduction Techniques
- Lecture 3:* Software Model Checking
- Lecture 4:* State/Event-based software model checking
- Lecture 5:* Deadlock Detection and Component Substitutability
- Lecture 6:* Model Checking Practicum (Student Reports on the Lab exercises)

2

Actual Goal

- Deadlock for **concurrent** blocking **message-passing C programs**
- Tackle **complexity** using **automated abstraction** and **compositional reasoning**
- Obtain **precise** answers using **automated iterative abstraction refinement**

3

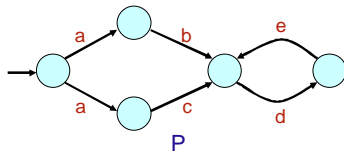
For this talk

- Focus on finite **state machines**
 - Labeled transition systems (LTSs)
- Parallel **composition** of state machines
 - **Synchronous** communication
 - **Asynchronous** execution
 - Natural for **modeling** blocking **message-passing C programs**

4

Finite LTS

- $P = (Q, I, \Sigma, T)$
 - $Q \equiv$ 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



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

5

Concurrency

- Components communicate by **handshaking** (synchronizing) over **shared actions**
- Else proceed independently (**asynchronously**)
- Essentially **CSP semantics**
- Composition of A_1 & $A_2 \equiv A_1 \parallel A_2$

6

Operational Semantics

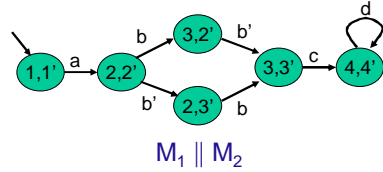
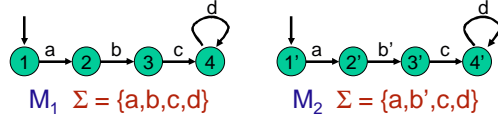
• State of $M_1 \parallel M_2$ is of the form (s_1, s_2) where s_i is a state of M_i

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

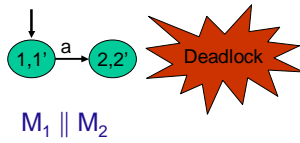
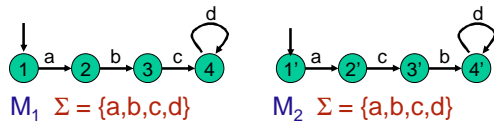
$$\frac{s_1 \xrightarrow{a} s'_1 \quad s_2 \xrightarrow{a} s'_2}{(s_1, s_2) \xrightarrow{a} (s'_1, s'_2)}$$

State-space exponential in # of components

Example

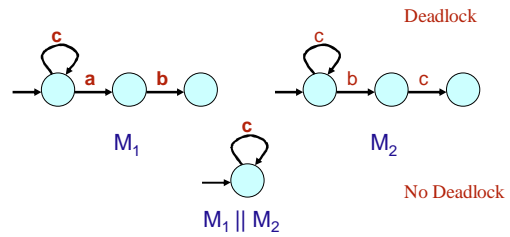


Deadlock

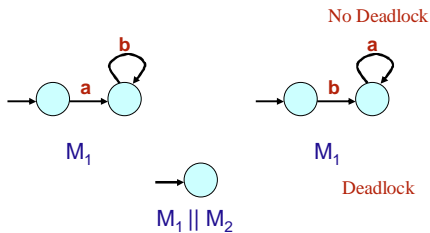


Deadlock \Leftrightarrow a reachable state cannot perform any actions at all

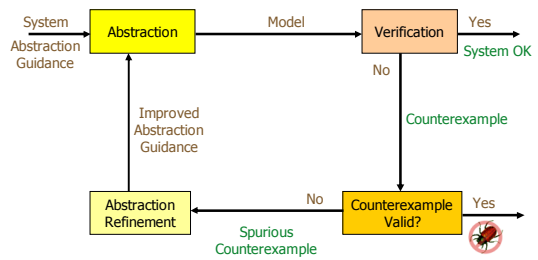
Deadlock and Composition



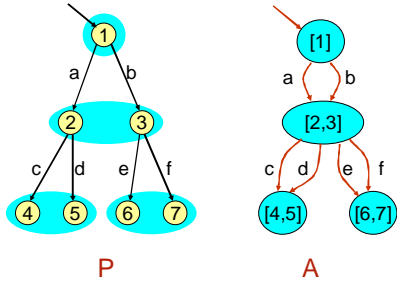
Deadlock and Composition



Iterative Refinement



Conservative Abstraction



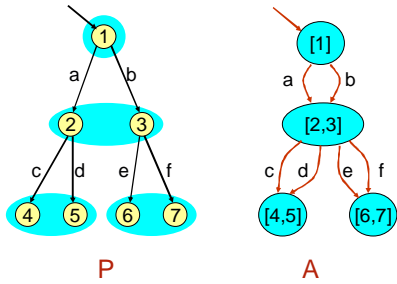
13

Conservative Abstraction

- Every trace of P is a trace of A
 - Preserves safety properties: $A \models \phi \Rightarrow P \models \phi$
 - A over-approximates what P can do
- Some traces of A may not be traces of P
 - May yield spurious counterexamples - (a, e)
- Eliminated via abstraction refinement
 - Splitting some clusters in smaller ones
 - Refinement can be automated

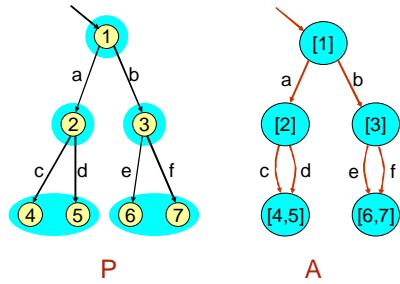
14

Original Abstraction



15

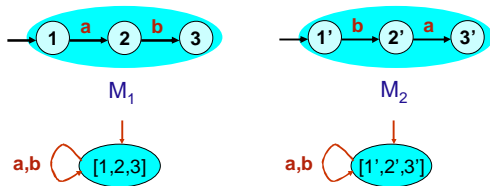
Refined Abstraction



16

Deadlock : Problem

- Deadlock is not preserved by abstraction



17

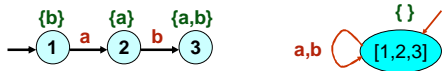
Deadlock Detection : Insight

- Deadlock \Leftrightarrow a reachable state cannot perform any actions at all
 - Deadlock depends on the set of actions that a reachable state cannot perform
- In order to preserve deadlock A must over-approximate not just what P can do but also what P refuses

18

Refusal & Deadlock

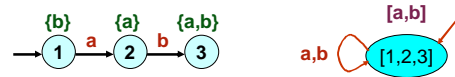
- $Ref(s)$ = set of actions s cannot perform



- M **deadlocks** iff there is a reachable state s such that $Ref(s) = \Sigma$
 - Denote by $DLock(M)$
- $Ref([s_1 .. s_n]) = Ref(s_1) \cap .. \cap Ref(s_n)$

Abstract Refusal

- $AR([s_1 .. s_n]) = Ref(s_1) \cup .. \cup Ref(s_n)$



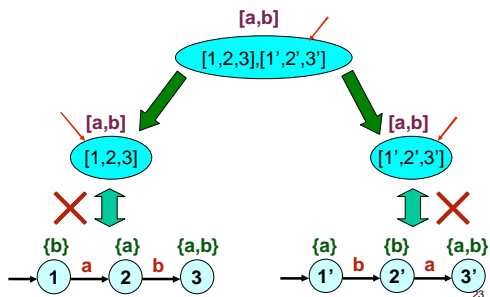
- $AR([M_1] .. [M_n]) = AR([M_1]) \cup .. \cup AR([M_n])$

Abstract Deadlock

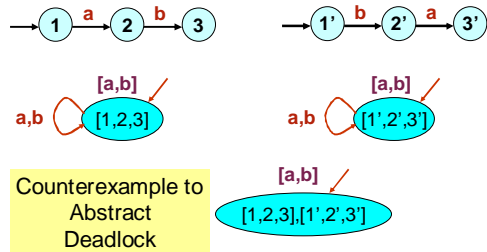
- M **abstractly deadlocks** iff there is a reachable state s such that $AR(s) = \Sigma$
 - Denote by $ADLock(M)$

$\neg ADLock([M_1] || .. || [M_n])$
 \Rightarrow
 $\neg DLock(M_1 || .. || M_n)$

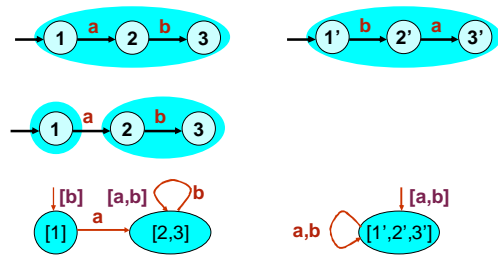
Counterexample Validation

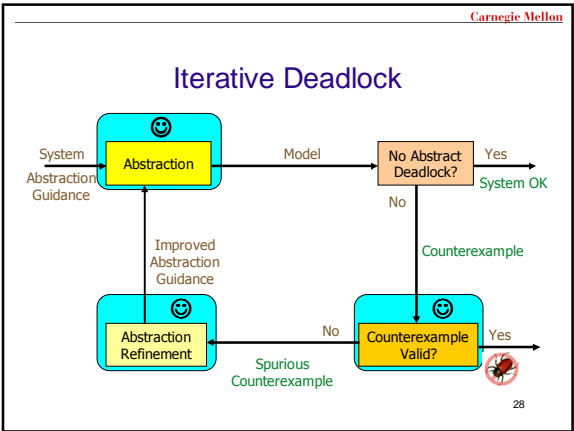
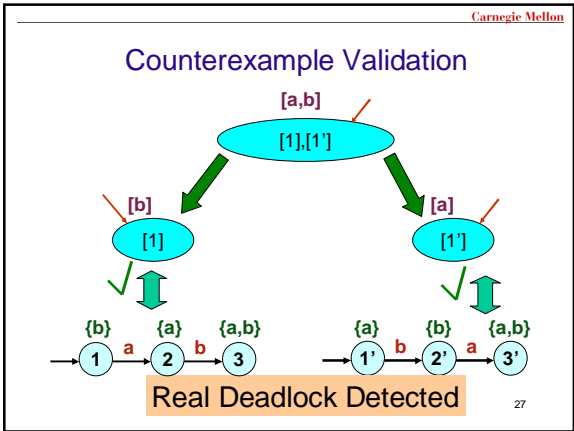
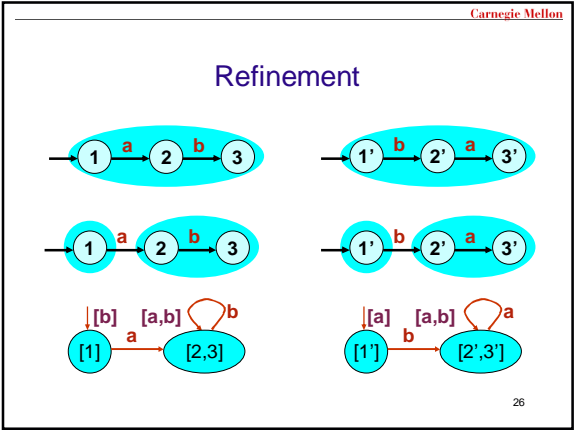
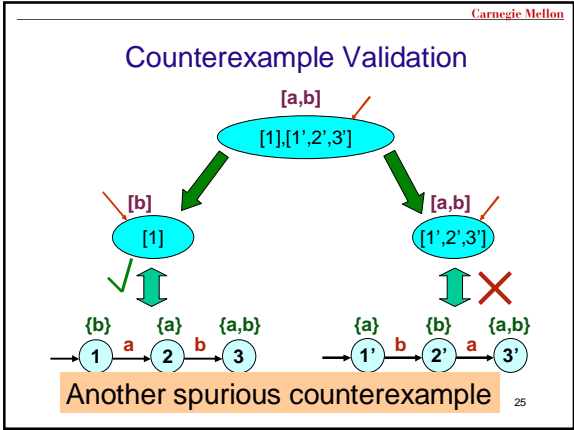


Iterative Deadlock Detection



Refinement





- Carnegie Mellon
- ### Case Studies
- **MicroC/OS-II**
 - Real-time OS for **embedded** applications
 - Widely used (cell phones, medical devices, routers, washing machines...)
 - 6000+ LOC
 - **ABB IPC Module**
 - Deployed by a world leader in robotics
 - 15000+ LOC
 - 4 components
 - Over 30 billion states after predicate abstraction
- 29

Carnegie Mellon

Results

Name	Plain			IterDeadlock			
	St	T	Mem	St	It	T	Mem
ABB	*	*	162	1973	861	1446	33.3
SSL	25731	44	43.5	16	16	31.9	40.8
μCD-3	*	*	58.6	4930	120	221.8	15
μCN-6	*	*	219.3	71875	44	813	30.8
DPN-6	*	*	203	62426	48	831	26.1
DPD-10	38268	87.6	17.3	44493	51	755	18.4

* indicates out of time limit (1500s)

30

Ongoing and Future Work

- Shared **memory**
- Assume-Guarantee reasoning
- Industrial size **examples**
- **Symbolic** implementation
- **Branching-time** state/event logic (completed)

35

Component Substitutability: Motivation

- Model checking is a highly time consuming, labor intensive effort
- For example, a system of 25 components (~20K LOC) and 100+ properties might take up to a month of verification effort
- It discourages practitioner use when system evolves
- Can model checking be used to **automatically** determine if previously established properties will hold for the *evolved* system **without repeating each of the individual checks** ³⁶

What's The Problem

- Software evolution is inevitable in any real system:
 - Changing requirements
 - Bug fixes
 - Product changes (underlying platform, third-party, etc.)
 - Incremental verification during the design process

37

Component Substitutability Check

- Component-based Software
 - Software modules shipped by separate developers
 - Undergo several updates/bug-fixes during their lifecycle
- Component assembly verification
 - Necessary on upgrade of any component
 - High costs of complete global verification
- Idea:
 - Instead check **locally** for **substitutability** of new components

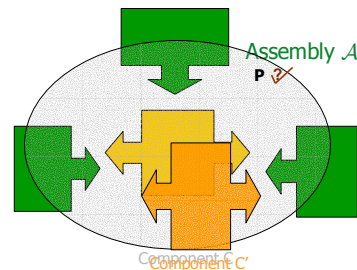
38

Potential Contribution

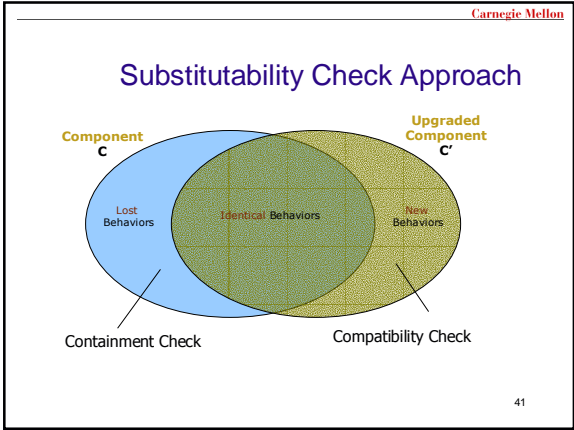
- Verify upgraded components locally
- Reuse previous verification results
- For example, for a system of 25 components (~20K LOC) and 100+ properties verification might take up to a month of verification effort
- If 3 components change, instead of repeating a month effort of re-verifying 100+ properties, our technique will ensure the substitutability of all properties in one iteration of the substitutability check (~ 1 day effort).

39

Component Substitutability Check

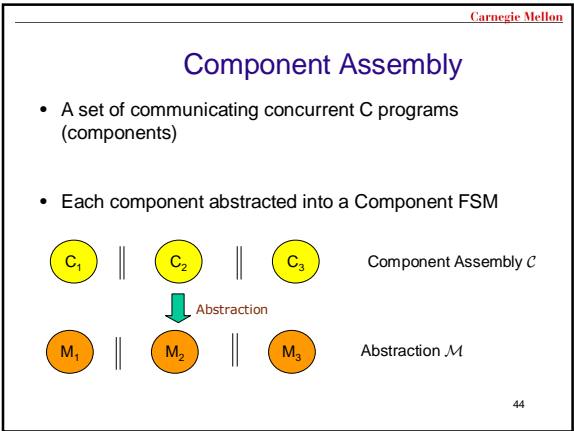


40

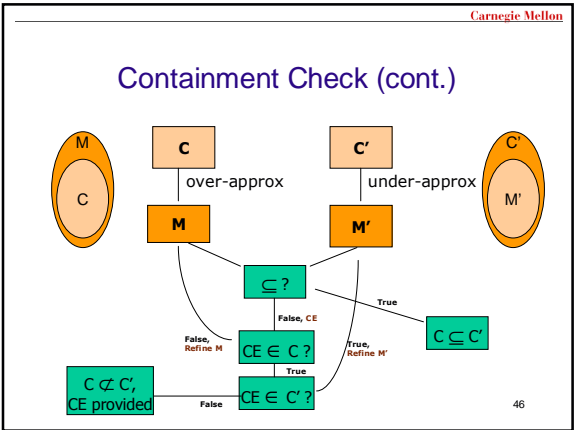


- Carnegie Mellon
- ## Substitutability Check Approach
- Two phases:
 - **Containment check (Local correctness)**
 - Are all *local* old services (properties) of the verified component contained in the upgraded component?
 - **Compatibility check (Global safety check)**
 - Are new services of the upgraded component safe with respect to other components in assembly: all *global* specifications still hold?
- 42

- Carnegie Mellon
- ## Substitutability Check
- Approach:
 - Obtain finite state models of all components by abstraction
 - **Containment Check:**
 - Use *under-* and *over-* approximations (*new*)
 - **Compatibility Check:**
 - Use *dynamic* assume-guarantee reasoning (*new*)
- 43



- Carnegie Mellon
- ## Containment Check
- Goal: Check $C \subseteq C'$ (Every behavior of C is an allowable behavior of C')
 - All behaviors *retained* after upgrade
 - Solution:
 - Create abstraction (over-approximation) M: $C \subseteq M$
 - Create abstraction (under-approximation) M': $M' \subseteq C'$
 - Check for $M \subseteq M'$
-
- 45

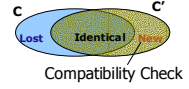


Containment Check (cont.)

- Computing over-approximation
 - Conventional predicate abstraction
- Computing under-approximation
 - Modified predicate abstraction
 - Compute **Must** transitions instead of **May**

Compatibility Check

- Assume-guarantee to verify assembly properties
 - Related: Cobleigh et. al. at NASA Ames



- Reuse previous verification results

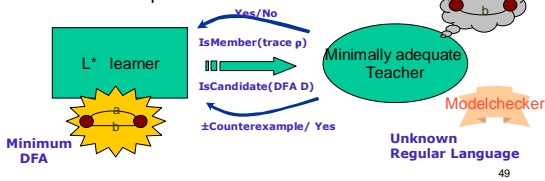
$$\frac{M_1 \parallel A \models P \quad M_2 \models A}{M_1 \parallel M_2 \models P}$$

AG-NonCircular

- Use **learning** algorithm for regular languages, L*
- **Automatically** generate assumption A

Learning Regular languages: L*

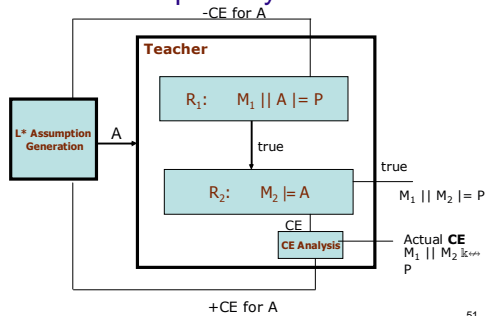
- Proposed by D. Angluin, improved by Rivest et al.
 - *Learning regular sets from queries and counterexamples*, Information and Computation, 75(2), 1987.
- Polynomial in the number of states and length of counterexample



Learning for Verification

- Model checker as a **Teacher**
 - Possesses information about concrete components
 - Model checks and returns true/counterexample
- **Learner** builds a model sufficient to verify properties
- Wide applications:
 - *Adaptive Model Checking*: Groce et al.
 - *Automated Assume-Guarantee Reasoning*: Cobleigh et al.
 - *Synthesize Interface Specifications for Java Programs*: Alur et al.

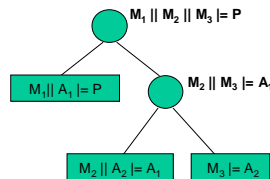
Compatibility Check



Handling Multiple Components

- AG-NC is recursive
 - (Cobleigh et al.)

$$\frac{R_1: M_1 \parallel A \models P \quad R_2: M_2 \models A}{M_1 \parallel M_2 \models P}$$



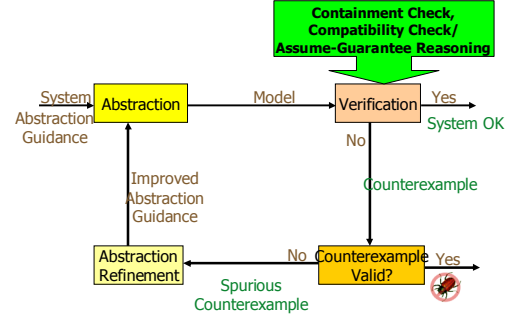
- Each A_i computed by a separate L* instantiation

Implementation

- ComFoRT Framework
- Validated on an Industrial benchmark
 - Inter-process Communication (IPC) ABB software
 - 4 main components – *CriticalSection*, *IPCQueue*, *ReadMQ*, *WriteMQ*
- Evaluated on single and simultaneous upgrades
 - *WriteMQ* and *IPCQueue* components
- Properties
 - P_1 : Write after obtaining CS lock
 - P_2 : Correct protocol to write to *IPCQueue*

53

ComFoRT Schema



54

Lab Assignment

- Spit into groups of 4-5 people
- Design, implementation and verification of the current surge protector
 - In PROMELA/SPIN
 - In ComFoRT
- Comparative validation
- Presentations on March 31, 2005

55

Lab Assignment (2)

- Questions about ComFoRT
 - Natasha Sharygina: nys@sei.cmu.edu - *theory*
 - Sagar Chaki: chaki@sei.cmu.edu – *tool support*

56

Collaboration Opportunities

- Research and development projects on verification of software (ComFoRT project)
- As part of the PACC (Predictable Assembly from Certifiable Components) project at the SEI
- Joint work with Prof. Ed Clarke

57

Collaboration Opportunities

- Independent studies
 - M.S. and Ph.D. Research (jointly with your current advisors)
 - Internships
- If interested contact me and we can discuss options

58