NASA’s State-Space Exploration: Verifying Safety-Critical Systems

Dimitra Giannakopoulou
CMU / NASA Ames Research Center
model checking

program / model

```java
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:
(provides a violating execution)
model checking vs. testing

Testing

Model checking
is it a good idea?

Turing Award 2007

E. Clarke, A. Emerson, J. Sifakis

- Targets subtle (concurrency) errors
- Safety critical applications
- Successful in hardware industry

- Input language
- Properties
- Computational complexity
state-explosion problem
compositional verification
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)
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 $P$?
  
  - components typically satisfy requirements in specific contexts / environments

- assume-guarantee reasoning
  
  - 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):

1. \( \langle A \rangle M_1 \langle P \rangle \)
2. \( \langle \text{true} \rangle M_2 \langle A \rangle \)

\( \langle \text{true} \rangle M_1 \parallel M_2 \langle P \rangle \)

“discharge” the assumption
examples of assumptions

- will not invoke “close” on a file if “open” has not previously been invoked
- 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
how do we come up with assumptions?

(best paper ASE 2002, ACM distinguished paper)

compute weakest assumption $\text{WA}$ s.t., for all environments $E$

$\langle \text{true} \rangle M_1 \parallel E \langle P \rangle \iff \langle \text{true} \rangle E \langle \text{WA} \rangle$
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
  - $P_{\text{err}}$ — complement of P
    - determinize & complete P with an “error” state;
    - bad behaviors lead to error
  - component M satisfies P iff error state unreachable in $(M \parallel P_{\text{err}})$

- **assume-guarantee reasoning**
  - assumptions and guarantees are FSMs
  - $\langle A \rangle M \langle P \rangle$ holds iff error state unreachable in $(A \parallel M \parallel P_{\text{err}})$
L* learns unknown regular language $U$ (over alphabet $\Sigma$) and produces minimal DFA $A$ such that $L(A) = U$ 
($L^*$ originally proposed by Angluin)
(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)
learning assumptions

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

$L^*$

query: string $s$

query $c^\alpha A$

conjecture: $A_i$

$c^\alpha A$

model checking

false + crex $c$

false + crex $c$

true / false

$c^\alpha A$

false

true

true

false

$\langle A_i \rangle M_1 \langle P \rangle$

$\langle true \rangle M_2 \langle A_i \rangle$

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

$P$ holds in $M_1 || M_2$

$P$ violated in $M_1 || M_2$

- $\alpha A = (\alpha M_1 \cup \alpha P) \cap \alpha M_2$
- terminates with weakest assumption or earlier
- $|A_1| < |A_2| < \ldots < |A|$, with $|A| \leq |WA|$
more than 2 components…

- extension of basic rule ASYM [TACAS 2003, FMSD 2009]
- symmetric / circular rules [SAVCBS 2003, FMSD 2009]
**extension of ASYM to $n$ components**

- to check if $M_1 \parallel M_2 \parallel \ldots \parallel M_n$ satisfies $P$
  - decompose into $M_i$ and $M'_i = M_2 \parallel \ldots \parallel M_n$
  - apply learning framework recursively for 2nd premise of rule
  - $A_1$ plays the role of the property for $M'_i$

1. $\langle A_1 \rangle M_1 \langle P \rangle$
2. $\langle true \rangle M_2 \parallel \ldots \parallel M_n \langle A_1 \rangle$

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

- at each recursive invocation for $M_j$ and $M'_j = M_{j+1} \parallel \ldots \parallel M_n$
  - use learning to compute $A_j$ such that
    - $\langle A_j \rangle M_j \langle A_{j-1} \rangle$ is true
    - $\langle true \rangle M_{j+1} \parallel \ldots \parallel M_n \langle A_j \rangle$ is true
- 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]
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]
interface generation
component 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

we use learning to generate interfaces [FASE 2009]
  - conjectured interfaces must be safe and permissive
  - queries and safety checked as in compositional scheme
  - permissiveness checked with heuristics
assume-guarantee reasoning

interface generation / discharge

Extensions/cv

http://javapathfinder.sourceforge.net
example 1: crew exploration vehicle

- **tool**: JavaPathfinder
- **UML statechart model of the Ascent and EarthOrbit flight phases of a spacecraft**
- **properties**: 
  - “An event IsamRendezvous, which represents a docking maneuver with another spacecraft, fails if the LAS (launch abort system) is still attached to the spacecraft”
  - “Event tliBurn (trans-lunar interface burn takes spacecraft out of the earth orbit and gets it into transition to the moon) can only be invoked if EDS (Earth Departure Stage) rocket is available”
generated interface assumptions encode Flight Rules in terms of events
example 2: procedure verification

- procedures guide work performed by astronauts
  - ISS power system troubleshooting / fixing

- procedure authoring environment

- verification with JavaPathfinder
  - domain models as UML statecharts
  - procedures in PRL language
  - automatically translated to statecharts for JavaPathfinder

- discovered error missed by simulation

- interface uncovered subtle timing issue
summary

- automating compositional verification was a breakthrough
- our techniques are generic
- not a panacea...
  - perform well when alphabets & assumptions are small
- what makes a system amenable to compositional techniques?
model checking
thank you!

http://javapathfinder.sourceforge.net