Model-Checking Behavioral Programs

Assaf Marron

Weizmann Institute of Science
Introduction to behavioral programming
The model-checker
Team / Acknowledgements

- David Harel
- Gera Weiss
- Guy Wiener
- Smadar Szekely
- Robby Lampert
- Michal Gordon
- Nir Eitan
- Amir Nissim
- David Harel’s research group - past and present
The Behavioral Programming Vision

Can complex software be developed from simple threads of behavior by automatic interweaving?
Humans interweave behavior threads all the time...

Driving Directions

9. Merge onto I-78 W
   Partial toll road
   Entering Pennsylvania
   About 2 hours 1 min

10. Merge onto I-81 S
    About 39 mins

Daily Schedule

... Drive for 4 hrs.
Stop for Lunch
Drive for 5 hrs.
...

A 6-day trip from NY to LA

... can software be developed this way?
**LSC**: A visual language for scenario specification

- Damm and Harel 2001, Harel and Marely 2003
- Natural yet executable scenario-based specification
- Initially for requirement specification, evolved into a programming language
- PlayGo – an IDE for programming with LSC

**BPJ**: A package for programming scenarios in Java

(and equivalents for other languages)

- Harel, Marron, and Weiss 2010
- Bringing advantages of scenario-based specification to programming
- Integrate with & complement other paradigms (OOP, aspects, rule-based, agile, ...).
class AddHotFiveTimes extends BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addHot, none, none);
        }
    }
}

Req. 3.1

class AddColdFiveTimes BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addCold, none, none);
        }
    }
}

Req. 5.2.9

class Interleave extends BThread {
    public void runBThread() {
        while (true) {
            bSync(none, addHot, addCold);
            bSync(none, addCold, addHot);
        }
    }
}

Patch 7.1
Why do we need this?

A key benefit: incremental development

Need to accommodate a cross-cutting requirement? Add a module

Need to refine an inter-object scenario? Add a module

Need to remove a behavior? Add a module

... ? Add a module

No need to modify existing code
1. All behavior threads (b-threads) post declarations:
   - **Request** events: propose events to be considered for triggering;
   - **Wait** for events: ask to be notified when events are triggered;
   - **Block** events: temporarily forbid the triggering of events.

2. When all declarations are collected:
   - An event that is **requested** and not **blocked** is selected.
   - All b-threads **waiting** for this event can update their declaration.
Behavior execution cycle

Behavior Threads

Wait

Request

Block
Behavior execution cycle

Behavior Threads

Request

Wait

Block
Behavior execution cycle

Behavior Threads

Wait

Request

Block
The BPJ Library and API

- B-threads are Java threads
- Events and event sets are Java objects and collections
- Development and execution do not require special environments
- Direct integration with other Java code:

```java
class MyBThread extends BThread {
    void runBthread() {
        ...
        bSync(requestedEvents, watchedEvents, blockedEvents);
        ...
    }
}
```

- The transition system is implicit

Online: The Group’s SVN
Example: Coding b-threads in Java

class AddHotFiveTimes extends BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addHot, none, none);
        }
    }
}

class AddColdFiveTimes BThread {
    public void runBThread() {
        for (int i=1; i<=5; i++) {
            bSync(addCold, none, none);
        }
    }
}

class Interleave extends BThread {
    public void runBThread() {
        while (true) {
            bSync(none, addHot, addCold);
            bSync(none, addCold, addHot);
        }
    }
}
Main application: reactive systems

Complexity stems from the need to interleave many simultaneous behaviors
When I put two Xs in a line, you need to put an O in the third square.
Each new game rule or strategy is added in a separate b-thread without changing existing code.
Example: Flying a quadrotor helicopter

To correct the angle:
- request SlowDownR4
- block SpeedUpR4
- request SpeedUpR2
- block SlowDownR2

To increase altitude:
- request SpeedUpR4
- block SlowDownR4
- request SpeedUpR3
- block SlowDownR3
- request SpeedUpR1
- block SlowDownR1
- request SpeedUpR2
- block SlowDownR2

Selected event: SpeedUpR2
Balancing a quadrotor – behaviorally
» How do we know when we are done?

» When each module is programmed separately, how do we avoid conflicts?

» An answer: **Model Checking + Incremental Development**
Model Checking

» Given a model of a system, test automatically whether this model meets a given specification.

» Program model = state graph

» Specification:
  > Safety conditions (including deadlocks)
  > Liveness

» We are focused on *explicit* MC as opposed to *symbolic*. 
A **b-thread** is a tuple $\langle S, E, \rightarrow, init, R, B \rangle$

> Where $\langle S, E, \rightarrow, init \rangle$ is a transition system, and

> for each state $s$:

+ the set $R(s)$ models the **requested** events

+ the set $B(s)$ models the **blocked** events

\[
\begin{align*}
R(s_1) &= \{e_1, e_2\} \\
B(s_1) &= \{e_3, e_4\} \\
\end{align*}
\]

\[
\begin{align*}
R(s_2) &= \{e_1, e_7\} \\
B(s_2) &= \{e_8\} \\
\end{align*}
\]
Composition of the b-threads \( \{ \langle S_i, E_i, \rightarrow_i, \text{init}_i, R_i, B_i \rangle : i=1,\ldots,n \} \) is defined as a product transition system.

The composition contains the transition \( \langle s_1, \ldots, s_n \rangle \xrightarrow{e} \langle s'_1, \ldots, s'_n \rangle \) if:

\[
e \in \bigcup_{i=1}^{n} R_i(s_i) \quad \land \quad e \notin \bigcup_{i=1}^{n} B_i(s_i)
\]

- \( e \) is requested
- \( e \) is not blocked

\[
\bigwedge_{i=1}^{n} \left( (e \in E_i \Rightarrow s_i \xrightarrow{e} s'_i) \land (e \notin E_i \Rightarrow s_i = s'_i) \right)
\]

- affected b-threads move
- unaffected b-threads don’t move
Behavior Thread States
b-thread states at bSync

```c
labelNextVerificationState( "A" );
bSync( ... );
if( lastEvent == event1 ) {
    labelNextVerificationState( "B" );
bSync( ... );
}
if( lastEvent == event2 ) {
    labelNextVerificationState( "C" );
bSync( ... );
}
```
Program states are the Cartesian product of b-thread states.
Model-checking behavioral programs “in-vivo” (c.f. Java Path Finder)

Backtrack using Apache javaflow continuations

Transition using standard execution (by the native JVM)

State matching and search pruning by b-threads

Notations for nondeterministic transitions

Deadlocks detected automatically

State tagging for safety and liveness properties by b-threads
Counterexample: A path to a bad state
Demo
Development Summary

» Initial Development:
  > DetectXWin, DetectOWin, DetectDraw
  > EnforceTurns
  > DefaultMoves
  > XAllMoves

» Modify b-threads to prune search / mark bad states

» Model Check $\rightarrow$ Counterexample $\rightarrow$ Add b-thread / change priority:
  > PreventThirdX
  > PreventXFork
  > PreventAnotherXFork
  > AddThirdO
  > PreventYetAnotherXFork
Let $c = e_1, \ldots, e_m, \ldots, e_n$ be a counterexample.

Can generalize and code new b-threads or,

Can use the counterexample in a patch behavior. E.g.,

> Let $e_m$ be the last event requested by the system

  + Wait for $e_1, \ldots, e_{m-1}$

  + Block $e_m$

> Other b-threads will take care of the right action, “the detour”.

> Model-check again
Fairness Constraints

- Unconditional: “Every process gets its turn infinitely often”.

- Strong: “Every process that is enabled infinitely often gets its turn infinitely often”

- Weak: “Every process that is continuously enabled from a certain time instant on gets its turn infinitely often”
» Input: fairness constraints as event sets

» MC: Look for cold states only in FAIR cycles
Other examples and experiences

» Bridge-crossing problem

» Dining Philosophers

» Scheduling in a signal-processing board
# Initial Performance

<table>
<thead>
<tr>
<th></th>
<th>Time (seconds)</th>
<th>States</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Spin/BEEM</td>
<td>BPmc counterexample</td>
</tr>
<tr>
<td></td>
<td>database</td>
<td></td>
</tr>
<tr>
<td>4 dining philosophers</td>
<td>0</td>
<td>0.031</td>
</tr>
<tr>
<td>6 dining philosophers</td>
<td>0</td>
<td>0.063</td>
</tr>
<tr>
<td>12 dining philosophers</td>
<td>4.26</td>
<td>3.812</td>
</tr>
<tr>
<td>4 persons crossing bridge</td>
<td>0</td>
<td>0.547</td>
</tr>
</tbody>
</table>

|                      | Spin/BEEM      | BPmc counterexample | BPmc no deadlock    |
|                      | database       |                     |                     |
| 4 dining philosophers| 80             | 50                  | 80                  |
| 6 dining philosophers| 729            | 528                 | 728                 |
| 12 dining philosophers| 531440         | 46632               | 531440              |
| 4 persons crossing bridge| 96194         | 24                  | N/A                 |
Limitations / opportunities

» Abstracts program only per behavioral states
» Dependent on application for state labeling
» Single threading during model-checking
» Dependency on Javaflow
» No support for dynamic B-Threads
» Application-dependent performance
» Explicit MC only
Some Interesting research experiences

» The Java Pathfinder (JPF) attempt

» The iterative execution version – “poor man’s verification”

» The backtracking challenge and finding Javaflow
Visualizing and Comprehending

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>AddHotThreeTimes</th>
<th>AddColdThreeTimes</th>
<th>Interleave</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Leader</td>
<td>Leader</td>
<td>Active</td>
</tr>
<tr>
<td>1</td>
<td>AddHot(ID=0)</td>
<td>![AddHot(ID=0) Diagram]</td>
<td>![AddCold(ID=0) Diagram]</td>
<td>![AddHot(ID=0) Diagram]</td>
</tr>
<tr>
<td>2</td>
<td>AddCold(ID=1)</td>
<td>![AddCold(ID=1) Diagram]</td>
<td>![AddCold(ID=1) Diagram]</td>
<td>![AddCold(ID=1) Diagram]</td>
</tr>
<tr>
<td>3</td>
<td>AddHot(ID=0)</td>
<td>![AddHot(ID=0) Diagram]</td>
<td>![AddCold(ID=0) Diagram]</td>
<td>![AddHot(ID=0) Diagram]</td>
</tr>
<tr>
<td>4</td>
<td>AddCold(ID=1)</td>
<td>![AddCold(ID=1) Diagram]</td>
<td>![AddCold(ID=1) Diagram]</td>
<td>![AddHot(ID=0) Diagram]</td>
</tr>
</tbody>
</table>
Visualization and Comprehension

Blocking to prevent playing out of turn

Event trace (rows) with b-thread states and R/W/B event sets

Prioritizing program over user and defense over default moves

A lower priority event (on right), is selected because a higher-priority event (on left) is blocked.
But still ...

» Can it scale to large applications?

» ... and what about external events?
Remote Events – Local Behavior

Real-life behavioral applications require distributed execution

- Asynchronous communication between nodes
- Synchronous collaboration inside nodes
- Each node has scenarios for handling remote events
Research Directions around MC for BP

Theory, tools, methodologies for:

> **Compositional model-checking**
  check each b-thread separately

> **Run-time model-checking**
  for event selection

> **Program synthesis**
  for automatic b-thread generation (e.g., for patching)
Summary

The behavioral programming paradigm

Direct model checking of behavioral Java programs

Synergies between BP and MC
Thank You!
Wargames: 1983, Dir. John Badham  
http://www.youtube.com/watch?v=NHWjlCaIrQo
BACKUP SLIDES
Example: A game strategy

**Move events:** $X_{<0,0>}, \ldots, X_{<2,2>}, O_{<0,0>}, \ldots, O_{<2,2>}$

**Game events:** OWin, XWin, Tie

**EnforceTurns:** One player marks a square in a 3 by 3 grid with $X$, then the other marks a square with $O$, then it is $X$’s turn again, and so on;

**SquareTaken:** Once a square is marked, it cannot be marked again;

**DetectWin:** When a player marks three squares in a horizontal, vertical, or diagonal line, she wins;

**AddThirdO:** After marking two $O$s in a line, the $O$ player should try to mark the third square (to win);

**PreventThirdX:** After the $X$ player marks two squares in a line, the $O$ player should try to mark the third square (to foil the attack);

**DefaultOMoves:** When other tactics are not applicable, player $O$ should prefer the center square, then the corners, and mark an edge square only when there is no other choice;
Javaflow

- [http://commons.apache.org/sandbox/javaflow/](http://commons.apache.org/sandbox/javaflow/)

- Save a thread’s stack in an object called a *continuation*.

- Can restore the continuation in any thread – and continue execution from there.

- BPmc optionally serializes the continuation with all pointed objects.

- See BP user guide.
Some answers to common questions and challenges

What about conflicting requirements?
- Model Checking
- Incremental development
- ...

Scalability in terms number of behaviors and interleaving complexity?
- Agent oriented architectures
- Machine learning for event selection
- ...

Comprehension of systems constructed by behavior composition?
- Trace visualization tool
- ...

[Diagram showing complex interactions and structures]