Automated Testing of Real-Time Systems

Brian Nielsen, Ph. D.

bnielsen@cs.auc.dk

Department of Computer Science
Aalborg University, Denmark
Outline

1. Specification Based Testing
2. Automated Testing
3. Lego Experiment
4. Future work / Collaboration
Why Improve Testing?

- **Costly** (30% of development time)
- Significant errors not found
- *The* validation activity used by *industry* today
- Must be improved both wrt.
  - Time and resource usage
  - Obtained coverage
What is Testing?

- **Testing**: execution of an implementation with the purpose of finding errors

  **Conformance testing**: does the (blackbox) behavior of a system comply to the specification
Specification Based Testing

A Specification

Test cases

Click!  
Wait 1.5  
click!  
DBLClick?  
(pass)

Click!  
Wait 5  
click!  
DBLClick?  
(fail)

Click!  
Wait 0.1  
click!  
DBLClick?  
(pass)
How do we cope with real-life specs?

Philips Sender with collision detection

```plaintext
config
system sender;
int sent;
observable in1, in0, up, dn, empty, isUp, coll;
```
Automated Testing

- Automatic *generation* of test cases from (formal) specifications
- Automatic *execution* of test cases
- Tool supported *management* of test cases
- Commercial tools becoming available
- Cases:
  - International Space Station [Peleska et al. ’99]
  - SSCOP protocol [Jard et al. ’00]
  - Satellite power and thermal control system [Schlingloff ’97]
- Little support for real-time constraints
Automated Testing Components

System Specification
- FSM, TA, SDL, Lotos, ...
  - Behavior
  - Environment assumptions
  - Selection Criteria
    - Test purpose

Abstract Test Cases
- FSM, TTCN

Implementation
- Test execution tool
- Event mapping
- Driver

Test
- pass
- fail

Probes
On-The-Fly Testing

• Test cases are constructed as they are being executed, one event at a time
  • Literally 100,000 events per test case
  • Completely automatic execution during several weeks
  • Handles very large specifications

• Software-in-the-loop: test driver software executes at target platform
• Hardware-in-the-loop: external test computer connected to target system via hardware interfaces
My Research

• Until now
  - Theory, Algorithms, and Tools for
  - automatic generation of Real-Time Test Cases
  - from timed specifications (Timed Automata).
    • Symbolic Analysis technique for timed automata
    • Guarantee structural coverage of specification!
An Example: Lego Bricks Sorter

- *Indicate link to the real world*
- Simple Demo
- Demonstration, not scientific proof of methodology
- Test cases generated automatically from TA-specification by prototype tool (PhD thesis work)
- Execution tool is a quick and dirty implementation, i.e., no general tool available
Sorting of Lego Boxes

Ken Tindell

Boxes

Conveyor Belt

Piston

eject
remove

Controller

MAIN  PUSH

Black

Red
RCX

- Hitachi H8/3292 micro controller
  - 8-bit processor, 16 bit address space
  - DAC, I/O Ports, Timers, etc
  - 32k RAM,
  - 8 k ROM
  - 16 Mhz

3 output ports

IR ports

3 input ports
**LegOS**

- Alternative kernel to Lego’s firmware
  - Pre-emptive multitasking, 20 ms time slice
  - 1 ms timer resolution
  - 16 MHz native mode speed
  - Access to 32k RAM
  - Semaphores
  - Dynamic memory management
  - Drivers for all RCX subsystems
  - Dynamic loading of programs and modules
  - Full IR packet networking

- GNU C cross compiler
Implementation? (LegOS+"Quite C")

Detector and pusher are concurrent threads

```c
int detector(int argc, char**argv) {
    ds_active(&LIGHT_SENSOR);
    active=0;
    motor_a_speed(MAX_SPEED); //conveyor belt
    motor_a_dir(fwd);
    cputs("chk");

    while(1){
        wait_event(&red_brick_wakeup,0);
        hitTime=sys_time+DELAY;
        active=1;
        wait_event(&no_red_brick_wakeup,0);
    }
    return 0;
}

int pusher(int argc, char** argv){
    while(1){
        wait_event(&hit_brick_wakeup,0);
        motor_c_speed(MAX_SPEED);
        motor_c_dir(rev);
        msleep(200);
        motor_c_dir(fwd);
        msleep(100);
        motor_c_dir(brake);
    }
    return 0;
}

wakeup_t red_brick_wakeup(wakeup_t data) {
    return (LIGHT_VAL>=RED_LIGHT_LEVEL);
}

wakeup_t no_red_brick_wakeup(wakeup_t data) {
    return (LIGHT_VAL<=BACK_LIGHT_LEVEL);
}

wakeup_t hit_brick_wakeup(wakeup_t data) {
    return (sys_time>hitTime && active==1);
}
```
Bricks-Sorter for Lego Mindstorms

Specification

Example Abstract Tests
(Generated by RTCAT Prototype Tool)

Test 1:
blackBrick!
Wait(0)
redBrick!
Wait(3000)
Eject?
(pass)

Test 2:
blackBrick!
Wait(0)
redBrick!
Wait(2999)
Eject?
(fail)

Test 3:
blackBrick!
Wait(0)
redBrick!
Wait(3001)
Eject?
(fail)
Test Execution Architecture

1. Interpretes tests, and
2. Translates abstract events to observable events
3. Observe events
4. Assign verdicts
Abstract vs. Concrete events

Red brick:
1. write light intensity value typical for a red brick to register shadow variable
2. Wait 500 ms: duration of brick
3. write typical background light intensity value to register shadow variable

Eject: 3 sub events:
1. Check that motor is forward & speed set to Max by reading motor control registers
2. After 200 ms
3. Check that motor has reversed & speed set to max
4. After 100 ms
5. Check motor state is stopped and has zero speed

Events are accepted ‘around’ the expected time, i.e. 3000 ±δ, 2999 - δ
Test Outcome!

Test 1: pass
Test 2: fail - ejects too early: *cause ?????*
Test 3: fail - makes several successive ejects. *Cause: forgot to reset flag*

```c
int detector(int argc, char**argv){
    ds_active(&LIGHT_SENSOR);
    active=0;
    motor_a_speed(MAX_SPEED); //conveyor belt
    motor_a_dir(fwd);
    //cputs("chk");

    while(1){
        wait_event(&red_brick_wakeup,0);
        hitTime=sys_time+DELAY;

        active=1;
        wait_event(&no_red_brick_wakeup,0);
    }
    return 0;
}
```

```c
int pusher(int argc, char**argv){
    while(1){
        wait_event(&hit_brick_wakeup,0);

        active=0;
        motor_c_speed(MAX_SPEED);
        motor_c_dir(rev);
        msleep(200);
        motor_c_dir(fwd);
        msleep(100);
        motor_c_dir(brake);
    }
    return 0;
}
```

This line was missing
The Philips Audio Protocol

- A bus based protocol for exchanging control messages between audio components
  - Collisions
  - Tolerance on timing events

Bit stream: 1 0 0 0 1 1 0
Manchester encoding: 

<table>
<thead>
<tr>
<th>TX</th>
<th>RX</th>
</tr>
</thead>
<tbody>
<tr>
<td>up</td>
<td>in0</td>
</tr>
<tr>
<td>dn</td>
<td>in1</td>
</tr>
<tr>
<td>in0</td>
<td>empty</td>
</tr>
<tr>
<td>coll</td>
<td></td>
</tr>
<tr>
<td>isUP</td>
<td></td>
</tr>
<tr>
<td>out0</td>
<td>out1</td>
</tr>
<tr>
<td>end</td>
<td></td>
</tr>
</tbody>
</table>

Diagram of TX and RX components
Philips Sender with collision detection

Complete Coverage:
- 99 test cases
- Total 548 steps
- Generated in 2 secs
Philips Receiver

- Complete Coverage:
  - 97 test cases
  - Total 527 steps
  - Generated in 2 seconds.
Future Work

• Current state-of-the art
  – Reasonably mature techniques for untimed SDL, FSM like specifications

• Do real-life industrial Case Studies
  – Untimed test generation
    • UML-FSM or Sdl-specifications
    • Combined generation+execution (on-the-fly)

• Further develop Real-Time Techniques
  – timed testing from Timed Automata specifications.