Who says model checking doesn’t find real problems?

ALICE MILLER

DEPARTMENT OF COMPUTING
SCIENCE
UNIVERSITY OF GLASGOW
Outline

• An introduction to Model checking + SPIN

• Wireless sensor networks, the DIAS project

• WSN modelling language - Insense

• Model checking Insense

• Future work
Model checking

Model (using Promela) → Kripke structure (FSA) → Buchi automaton

Specify property (using LTL) → property holds

Model check (using SPIN)

Counterexample, modify system, model or property
Design Implementation and Adaptation of Sensor Networks

The DIAS project was a collaboration between:

- University of Manchester
- University of Glasgow
- University of Kent
- Strathclyde University
- Lancaster University
- St Andrews University

Aim: to extend the notion of hw/sw co-design into multiple dimensions

Systems generated by combining individual components with appropriate software/firmware “glue”
Wireless Sensor Networks

- WSNs used by experts in different fields to gather data, e.g.
  - Environmental measurements
  - Air/water pollution

- Users expected to produce software for concurrent, RT and resource-constrained computing environment

- E.g. Crowden Great Brook, Derbyshire (Lancs. University)

Crowden Great Brook
Crowden Great Brook

- To characterise the activity in a small side stream
- Current deployment 15 nodes
  - 2 base-stations with a GSM modem to communicate alarm events back.
  - 12 measure temperature, humidity, soil moisture
  - 2 of these have extra features (turbidity sensor/rain gauge)
  - 3 extra nodes (no radio) measure stream depth

- Nodes have enough battery/solar to last 1yr
- 174MHz and 433MHz radios
WSNs contd.

More what we had in mind..
Tmote Sky

- 8MHz Texas Instruments MSP430 microcontroller
- 10k RAM, 48k Flash
- 250kbps 2.4GHz IEEE 802.15.4 Chipcon Wireless Transceiver
- Integrated Humidity, Temperature, and Light sensors
- Running (e.g. Contiki) OS
Insense

Language to run on a WSN. Developed at St Andrews. Features include:

• **abstraction** over complex/low-level programming issues

• **reduced risk of run-time errors** due to strong typing + ability to statically determine space and time requirements of programs

• **ability to dynamically change** configuration of applications,

• **portability of application**: programs independent of any specific operating system or hardware platform
sensor = new TempSensor ()
averager = new TempAverager ()
logger = new TempLogger ()
connect sensor.output to averager.input
connect averager.output to logger.input
Channel implementation

- Each connection consists of two half-channels
- Supported operations:
  - Send / Receive
  - Connect / Disconnect
  - (non-deterministic) Select

- When connection made between outgoing and incoming channels, channels locked in turn using mutex
Half-channel

- Object that contains:
  - Buffer for storing 1 item of corresponding message type
  - Ready flag
  - List of half-channels it’s connected to
  - 2 binary semaphores: mutex and blocked
Why model check Insense?

- Verify correctness of implementation
  - Deadlocks / Livelocks
  - Violations of constraints

- SPIN as a debugger for Insense Programs
  - Guided and random simulations
  - Communication sequences
  - Inspection of variable values
  - Check basic properties

This talk!
Half channel representation:

```c
typedef halfchan{
    // Binary semaphores
    bit mutex; // locks access to channels
    bit blocked; // indicates channel is blocked
    // Boolean flags
    bit ready; // TRUE if ready to send/recv
    // Buffer
    byte buffer;
    // List of connections to other half-channels
    bit connections[NUMHALFCHANS];
}
Send and Receive in Promela contd.

Locks:

```promela
atomic{
    hctab[me].mutex!=LOCKED; // wait for mutex
    hctab[me].mutex=LOCKED // LOCK mutex
}
```

Data Transfer:

Sender *pushes* data
Receiver *pulls* data

(register via global flags, for verification purposes)
Bug in published version

Using Spin, easily uncovered error in published algorithms

What was it?

*Receiver could return without receiving correct data*

How discovered with Spin?

*Simple (terminating) Sender + Receiver processes*
*Littered with assert statements*
*Assertion violation + examination of error trace*
Send/Recv Algorithms

1. wait( cout.mutex )
2. set( cout.ready )
3. signal( cout.mutex )
foreach (halfchan hc in cout.connList)
{
    wait( hc.mutex )
    if ( hc.ready ) {
        hc.buf = data // push
        ...
        return
    }
    signal( hc.mutex )
}
cout.buf = data
wait( cout.blocked )
return

4. wait( cin.mutex )
5. set( cin.ready )
6. signal( cin.mutex )
7. foreach (halfchan hc in cin.connList)
{
    8. wait( hc.mutex )
    9. if ( hc.ready ) {
        10. cin.buf = hc.buf // pull
            ...
            return
    }
    signal( hc.mutex )
}
cout.buf = data
wait( cin.blocked )
return

London Hoppers

4/5/2010
The initial bug contd.

• Why not discovered during simulation (Contiki simulator tool: Cooja)?
  
  Contiki not truly concurrent

• Why do we care?
  
  Want it to be correct regardless of the underlying operating system

• Only a simple bug!
  
  But it convinced Insense developers of the usefulness of Spin

• Solution?
  
  Set Sender’s buffer before ready flag set
Set of properties

1. In a connected system, send and receive operations are free from deadlock.
2. Finite progress- in a connected system data always flows from senders to receivers.
3. For any connection between a sender and a receiver, either the sender can push or the receiver can pull, but not both.
4. The send operation does not return until data has been written to a receiver’s buffer (either by sender-push or receiver-pull).

(plus 3 more properties, see paper)
Properties in LTL

- S senders and R receivers (non-terminating), S+R ≤ 4, R,S ≥ 1

- Property 1 can be checked via “invalid endstate” check with Spin

Property 2: \([ \ ] < > \text{Push}_1 \parallel \text{Pull}_1 \parallel \text{Push}_2 \parallel \text{Pull}_2 \parallel \ldots \parallel \text{Push}_R \parallel \text{Pull}_R \]

Property 3: \([ \ ] ! (\text{Push}_1 \&\& \text{Pull}_1) \parallel (\text{Push}_2 \&\& \text{Pull}_2) \parallel \ldots \parallel (\text{Push}_R \&\& \text{Pull}_R) \]
Spin verification

Promela models generated from template via Perl script.

See Spin paper for table of results
All properties verified for $S + R \leq 4$, $R,S \geq 1$

Max depth ($S=3$, $R=1$) reached at most $1.5 \times 10^6$

Max no. states ($S=2$, $R=2$, property 2), $2.8 \times 10^7$

Max memory required ($S=2$, $R=2$, property 2) approx 1Gb
Connect/Disconnect

• Algorithms for dynamic connection/disconnection using Spin to test for deadlock
• Many iterations required!
• Verified previously unpublished algorithms
• Prevent deadlocks via
  • Is_Input field of half channel, to impose common order on mutex locking in send/receive
  • (global) conn_op_mutex to prevent deadlock when executing concurrent connect/disconnect operations
  NOT IDEAL …
  .. but alternative version not fully verified (state-space explosion)

• In model RxS Connect processes, R+S Disconnect ...interleaving
Removal of the global locks

Leads to massive state space (exceeds available 32 Gb memory) – due to increase in interleavings

Usual tricks employed to improve tractability (“low fat” code, Spin options (e.g. state compression, stack cycling))

But . . . lots of symmetry:

E.g. If s is state in which S1 and R1 are connected, hctab[S1] and hctab[R1] have values v and u, then there is an equivalent state α(s) in which S1 and R2 are connected, hctab[S1], hctab[R2] have values α(v) and α(u), action of α defined in appropriate way.
Symmetry reduction

• Current symmetry reduction packages for Spin not appropriate (SymmSpin/TopSpin)

• Need to exploit symmetry between (pid indexed) global variables
  • hctab elements here

• Current work at Glasgow: extending TopSpin to handle this sort of case
  • Will hopefully allow for full verification (work in progress)

• (Meanwhile – any suggestions?)
Model checking Insense programs

- Exploit the similarity between Insense and Promela (both based on pi-calculus)

- SPIN used to debug Insense programs

- Subject of current work (converter implemented)

- Alternative approach – to develop direct model checking approach (à la JPF2) to model check automatically created abstract form (future work)
Conclusions

• Initial steps toward verifying correctness of WSN applications

• Verification of inter-component synchronisation mechanism
  • Still evolving – verification at design time

• Revealed error in previously published version of channel implementation
  • Aided development of
    • revised algorithms
    • New Connect/Disconnect algorithms
Future work

• Verification of Insense language implementation completed by modelling
  Select operation

• Show Send and Receive operations safe for any S and any R (PMCP)
  • Can show some properties hold for S=1 (any R>0)

• Symmetry reduction to reduce state space size

• Verification of Insense programs

• Direct model checker for Insense
Credit to:

(Glasgow/St Andrews contingent of) DIAS project team, specifically:
Oliver Sharma, Joe Sventek (Glasgow)
John Lewis, Al Dearle, Dharini Balasubramaniam, Ron Morrison,
(St Andrews)

TopSpin: Alastair Donaldson