Title
Deriving State Machines from TinyOS programs using Symbolic Execution

Authors
Kothari, Nupur
Millstein, Todd
Govindan, Ramesh

Publication Date
2009-05-12

Center for Embedded Networked Sensing

Deriving State Machines from TinyOS Programs using Symbolic Execution

Nupur Kothari, Todd Millstein and Ramesh Govindan

Introduction

- Programs written for Sensor Networks are quite complex
  - Event-driven programming style of most sensor network languages
  - Programmers need to handle resource and energy constraints
- Program complexity makes detecting discrepancies between programmer intent and program functionality hard for sensor networks
- A high-level representation makes programs easier to understand
- *FSMGen* is a tool to generate high-level representations in the form of user-understandable Finite State Machines (FSMs) from TinyOS programs, where TinyOS is one of the most popular programming systems for sensor networks

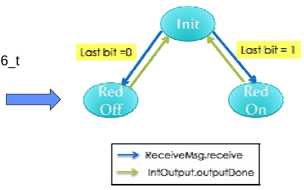
Example State Machine: RfmToLed

RfmToIntM.nc

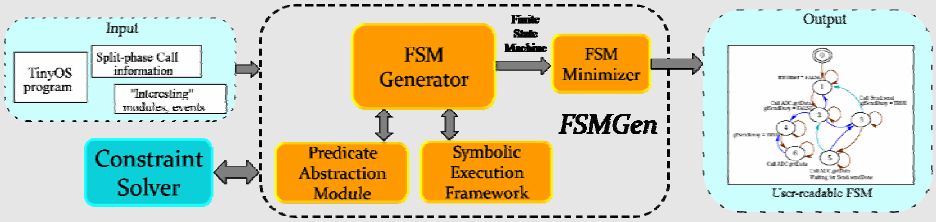
```
...
event TOS_MsgPtr
      ReceiveIntMsg.receive
      (...) {
  ...
  call IntOutput.output
     (message->val);
  ...
}
```

IntToLedsM.nc

```
...
command result_t
   IntOutput.output(uint16_t
   value){
 if (value & 1)
   call Leds.redOn();
 else
   call Leds.redOff();
 post outputDone();
 return SUCCESS;
}
```



FSMGen: System Overview


Input
TinyOS program
Split-phase Call information
"Interesting" modules, events
Constraint Solver
FSM Generator
Finite State Machine
FSM Minimizer
FSMGen
Predicate Abstraction Module
Symbolic Execution Framework
Output
User-readable FSM

FSMGen: Details

Symbolic Execution

- Program analysis technique that statically approximates program behaviour
- Involves simulating the execution of a program without actually running it, maintaining at each point information about the value of each variable (symbolic state)
- Adapted Symbolic Execution for event-driven model of TinyOS in *FSMGen*
  - Keep track of events enabled during execution and add this to symbolic state
  - Push task posted during execution into queue. At the end of the symbolic execution, execute all the tasks in the queue to obtain the final symbolic state
- *FSMGen* Symbolic Execution Framework built as an inter-procedural analysis in the CIL front-end for C
- Symbolic Execution Framework uses constraint solver CVC3 to solve for predicates during symbolic execution

Predicate Abstraction

- A technique to map a symbolic state to a corresponding valuation of a set of predicates (corresponds to a state in the FSM)
- The set of predicates that make up the state space of the FSM are obtained from the application code, and from the set of enabled events
- FSMGen Predicate Abstraction module
  - Takes as input a symbolic state
  - Generates as output a state in the FSM being derived

Generating an FSM

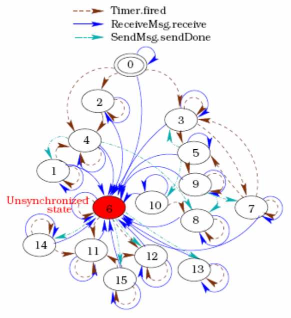
- FSM Generator first generates initial FSM state and makes note of possible events that can occur at this point
- For each possible event that is enabled initially, the FSM Generator calls upon the symbolic execution module to generate resulting symbolic states and converts them to FSM states using the Predicate Abstraction module
- The FSM Generator adds the new FSM the state machine using the events that were analyzed as edges
- The above process is repeated for all the new states that were created, and the events that are enabled for them

Limitations

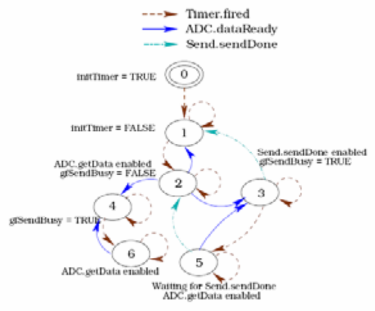
- Coarse approximation of execution model of TinyOS
  - Not applicable to low-level interrupt driven code
  - May miss potential execution paths
- Symbolic Execution Framework
  - Does not handle recursive functions, conservatively deals with pointers
  - Analysis of unbounded loops/iterative functionality is incomplete

Evaluation

- Tested *FSMGen* for a number of TinyOS applications and system components
  - TinyOS-1.x: Surge, RfmToLeds, CntToRfm, FTSP, MultiHopEngine
  - TinyOS-2.x: TestNetwork
- At most 15 minutes to infer FSMs in all but one cases – FTSP took ~24 hrs
  - Analysis is worst-case exponential in the number of predicates in the state space
- At most 16 states in all generated FSMs
- Inconsistencies discovered in 2 components – Surge, MultiHopEngine



Generated FSM for FTSP



Generated FSM for Surge

Future Work

- Optimizations to improve efficiency/running time of *FSMGen*
- Improve analysis of loops and iterative procedures
- Better approximation of TinyOS model
- Uses of the generated FSMs – race conditions, memory errors
- Release for public use

UCLA – UCR – Caltech – USC – UC Merced