Improving Reactive System Analysis Through Staging Analysis

PhD Prospectus

Roy Shea
NESL, UCLA CS
Motivating Better Analysis

• Development of I2C driver in SOS

```c
if (i2c.msgBuf != NULL) {
    i2c.msgBuf->data = NULL;
    i2c.msgBuf = NULL;
}
i2c.msg_state = SOS_MSG_NO_STATE;
```

```c
if (i2c.msgBuf != NULL) {
    ker_free(i2c.msgBuf->data);
    ker_free(i2c.msgBuf);
}
i2c.msg_state = SOS_MSG_NO_STATE;
```

• Problems common to sensor networks
  - Misuse of pointers
  - Incorrect interfacing between driver and OS
  - Difficult to test exceptional paths
  - Violation of hardware specification
A Solved Problem?

• Memory safety

$ ./test_protection
Segmentation fault (core dumped)

• Interprocedural analysis and domain languages

static void Main() {
    int [] in ExHeap vec = GetVector();
    for ( int i =0; i<vec.Length; i++) { vec[i] = i ; } 
    Consume(vec);
}
static int [] in ExHeap GetVector() {
    int [] in ExHeap vec = new[ExHeap] int[512];
    return vec;
}
static void Consume([Claims] int [] in ExHeap vec) {
    delete vec;
}
Staged Analysis in ESN Systems

- Reactive nature of embedded sensor networks (ESNs) degrades the quality of general analysis
- Specification languages not used in the domain
- Alternate approach is required:
  - A staged analysis combining static verification and higher order specifications significantly increases analysis precision for reactive systems.
Overview

- Introduction
- **Background Work**
- memCheck and Lighthouse
- Staged Analysis
- Application of Staged Analysis
- Summary
Academic Background

- **Before NESL**
  - Began in artificial intelligence and security
  - Graduate studies in general operating systems
  - Transitioned to embedded systems for research

- **NESL**
  - Software agents with Sensorware
  - Embedded operating system design with SOS
  - Verification with memCheck and Lighthouse
Overview

- Introduction
- Background Work
- memCheck and Lighthouse
- Staged Analysis
- Application of Staged Analysis
- Summary
memCheck: Introduction

• Problem statement:
  – *Common simple mistakes hinder ESN programmers.*

• Proposed solution:
  – *Simple checkers locate many of these common problems. Include basic verification tools in the SOS build system.*
memCheck: Evaluation

Warning: No top-level SWITCH with both INIT and FINAL handlers

Warning: Modules can not use globals:
  definition of init_threshold (at proc_mod.c:14)

• Check for problems common to ESN systems

<table>
<thead>
<tr>
<th>Failure to Check</th>
<th>Potential Memory Leaks</th>
<th>Programs Without Entry Points</th>
<th>Acessing Dead Data</th>
<th>Invalid Use of Global Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>494</td>
<td>427</td>
<td>14</td>
<td>22</td>
<td>27</td>
</tr>
</tbody>
</table>

Reported warnings across 1716 builds in EE202A between Oct. 25 – Nov. 5, 2005

• Verifying pointer usage is very inaccurate introducing many false positives
memCheck: Related Work


Recalling ENS Memory Properties

- Memory often measured in tens of kilobytes

<table>
<thead>
<tr>
<th>Platform</th>
<th>Microcontroller</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mica2 / MicaZ</td>
<td>Atmega128L 8-bit RISC</td>
<td>4 KB</td>
</tr>
<tr>
<td>TelosB</td>
<td>MSP 430 16-bit RISC</td>
<td>10 KB</td>
</tr>
<tr>
<td>Imote2</td>
<td>PXA271 32-bit ARM</td>
<td>256 KB</td>
</tr>
</tbody>
</table>

- ENS systems rarely have memory protection

- Domain independent techniques for analysis
  - Dataflow to track pointer state
  - Pointer analysis to handle aliasing
  - Function summaries to track pointer state
Lighthouse: Introduction

- **Problem statement:**
  - *Misuse of dynamic resource in ESNs causes difficult to diagnose errors.*

- **Proposed solution:**
  - *User provided function annotations drive a dataflow analysis looking for dynamic memory misuse.*
Lighthouse: Introduction

- “But system X doesn't use dynamic memory...”
- Comments from UCLA TinyOS developers:
  - Pointer misuse
    “A lot of bugs come from simple C errors.”
  - Queue overflow
    “Only after stepping through GDB one line at a time did we realize that the last task fell off the end of the queue.”
  - Stack overflow
    “If load time RAM requirement is less than 3.6 KB we assume the stack will be fine.”
Lighthouse: Implementation

• Assume single exclusive owner of memory
  – Memory should only be under the control of one module at any given time
  – The module controlling the memory must either keep track of the memory, return the memory, or give the memory to another module

• Approach:
  – Apply standard dataflow and pointer analysis techniques
mod_op = (sos_module_op_t*) ker_msg_take_data(msg);
if(mod_op == NULL) return -ENOMEM;
if(mod_op->op == MODULE_OP_INSMOD) {
    existing_module = ker_get_module(mod_op->mod_id);
    if(existing_module != NULL) {
        uint8_t ver = sos_read_header_byte(...);
        if (ver < mod_op->version) {
            ker_unload_module(...);
        } else {
            return SOS_OK;
        }
    }
    ret = fetcher_request(KER_DFT_LOADER_PID, mod_op->mod_id, mod_op->version, entohs(mod_op->size), msg->saddr);
    s->pend = mod_op;
    ker_led(LED_RED_TOGGLE);
    return SOS_OK;
} else {
    return SOS_OK;
}
Lighthouse: Evaluation

<table>
<thead>
<tr>
<th>Verified Memory Leaks</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Memory Leaks</td>
<td>8</td>
</tr>
<tr>
<td>Verified Dangling Pointers</td>
<td>0</td>
</tr>
<tr>
<td>False Dangling Pointers</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Verified Memory Leaks</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>False Memory Leaks</td>
<td>22</td>
</tr>
<tr>
<td>Verified Dangling Pointers</td>
<td>0</td>
</tr>
<tr>
<td>False Dangling Pointers</td>
<td>11</td>
</tr>
</tbody>
</table>

Warnings generated from the analysis of 213 unique versions of SOS modules totaling 28042 SLOC

Warnings generated from the analysis of 40 source files making up the SOS kernel totaling 9223 SLOC

- Memory leak false positives come from intraprocedural must alias analysis and linked list data structures
- Dangling pointer false positives come from field insensitive and flow insensitive
Lighthouse: Conclusions

- Effectively finds memory misuse in ESNs
- Requires significant tuning for best performance
- Novel tool
  - Pushes the limits of static data tracking
  - Demonstrates that simple frameworks for programmers make difficult problems tractable
  - Reveals the problem of analyzing reactive systems
- What about the false negatives?
Lighthouse: Related Work

Overview

- Introduction
- Background Work
- memCheck and Lighthouse
- *Staged Analysis*
- Application of Staged Analysis
- Summary
Staged Analysis: Introduction

- Lighthouse is not a conservative analysis
- Assumes that stores (persistent data) are always in the correct state at function entry
- Example problem case with verifiable functions
  - Function A allocates memory into data pointer
  - Function B fills data with ADC readings
  - Function C frees data after transmission
Staged Analysis: Introduction

- Valid sequences are of form \((AB^*C)^*\)
- Invalid function sequences include
  - **AA** – Leaks memory by overwrites the data pointer
  - **ACB** – Attempts to dereference invalid pointer
Staged Analysis: Introduction

- Problem statement:
  - *Inadequate knowledge at analysis time results in imprecise analysis.*

- Proposed solution:
  - *Use knowledge from other stages of analysis as foundation to improve precision of another analysis.*
Staged Analysis: Introduction

- Staging considers analysis at
  - Compile time
  - Load time
  - Run time

- Example focuses on memory verification through the Lighthouse tool
Staged Analysis: Implementation

- Static analysis assumes pre-conditions and verifies functions A, B, and C
- Runtime monitoring guarantees call graph and transitions to ERROR state if violated
Staged Analysis: Implementation

- Update Lighthouse dataflows to use / verify the state machine pre- / post- conditions
- Prototype digraph specification of state machine
  - Generates runtime code to monitor state transitions
  - Embeds checks within embedded system
- Combine these into a functional analysis
Staged Analysis: Evaluation

- Precision of analysis?
  - Count additional errors found over old verification framework
  - Examine ratio of false positives to true errors

- Run time overhead on sensor node hardware?

- Burden for end user?
  - Lines of code in specification
  - Complexity of specification
Staged Analysis: Related Work


Overview

- Introduction
- Background Work
- memCheck and Lighthouse
- Staged Analysis
- Application of Staged Analysis
- Summary
Staged Analysis: Thesis Research

- Analysis depending on call graph
  - Run time heap and stack usage
  - Worst case timing and delay
  - System power consumption
- Alternative forms of staged analysis
- Refining specifications
Problem statement:

- Heap (static queue) overuse in reactive systems is a common source of difficult track down errors.

Proposed solution:

- Combine traditional stack analysis with analysis of runtime heap (static queue) usage. State machine specification provides call graph constraints required for heap (static queue) analysis.
Heap and Stack Usage: Discussion

• Tension in ENS systems
  – Heap (static queue) utilization analysis requires precision call graph knowledge
  – Ambiguous call graph in reactive systems

• Approach:
  – Port stack analysis for use with ENS systems
  – Focus on queue analysis, heap analysis later
  – Begin with simplifying point event model
Problem statement:
- ENS operations often desire bounded delays.

Proposed solution:
- Join function timing analysis and call graph information to provide program flow timing constraints for ENS systems.
Delay and Timing: Discussion

- Users care about bottlenecks in systems
  - Moving beyond “How long does in take?”
  - Answering “What is the active time in one period?”

- **Approach:**
  - Dataflow for time bounds on functions
  - Look at end to end timing of event bursts
  - Combine analysis with bonds specification for runtime admission policy based on delays
Power Consumption: Introduction

• Problem statement:
  – ENS developers have little insight into where applications consume use power.

• Proposed solution:
  – Use a call graph specification to reason about power usage from all peripherals while statically analyzing a specific function.
Power Consumption: Discussion

• Power is critical concern in ENS systems
  – Understand power usage on rarely executed paths
  – Isolate power bugs before deployment

• Approach:
  – Add power into timing analysis
  – External hardware can be turned on and left on
  – Call graph determines hardware state changes
Alternate Staged Analysis Directions

• Prior analysis generate call graphs from state machine specification for use in static analysis

• Benefits flow in both directions
  – Stronger static analysis improves SFI performance
  – Dynamic addition / removal of modules requires run time analysis using results of static analysis to evaluate aggregate runtime heap bounds
Refining Specifications: Introduction

- **Problem statement:**
  - Specifications can require substantial additional effort from developers and provide opportunities for new bugs in the system.

- **Proposed solution:**
  - Use and extend standard specification tools. For call graph specific analysis, explore call graph inference.
Refining Specifications: Introduction

• Proposed research depends on specifications
  – Rich field already developed
  – Evolve event specification languages

• Approach:
  – Explore inference of call graphs
  – Continue to use established specification languages
Overview

- Introduction
- Background Work
- memCheck and Lighthouse
- Staged Analysis
- Application of Staged Analysis
- Summary
Approximate Schedule

- Spring 2007
  - Lighthouse and staged analysis
  - Begin TinyOS port
- Summer 2007
  - Internship?
- Fall 2007
  - Finish TinyOS port
  - Heap / queue analysis
- Winter 2008
  - Heap / queue analysis with program flows
- Spring 2008
  - Begin thesis
- Summer 2008
  - Runtime heap / queue monitoring
- Fall 2008
  - Finish thesis and defend
Summary

- Reactive nature of ESN systems hinders static analysis while resource constrained environment hinders runtime analysis
- Dissertation will use staged analysis to make higher precision analysis for ESN systems
The End
# Time Line

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Staged Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TinyOS Port</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load / Run Time Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thesis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task</th>
<th>Mar. '08</th>
<th>Apr. '08</th>
<th>May '08</th>
<th>Jun. '08</th>
<th>Jul. '08</th>
<th>Aug. '08</th>
<th>Sep. '08</th>
<th>Oct. '08</th>
<th>Nov. '08</th>
<th>Dec. '08</th>
<th>Jan. '09</th>
<th>Feb. '09</th>
</tr>
</thead>
<tbody>
<tr>
<td>Staged Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TinyOS Port</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Runtime Heap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Load / Run Time Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thesis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>