Model Checking Multitask Applications for OSEK Compliant Real-Time Operating Systems

Mark L. McKelvin, Jr., Edward Gamble, and Gerard Holzmann
Laboratory for Reliable Software
Jet Propulsion Laboratory
California Institute of Technology

PRDC 2011
December 12-14

Acknowledgment: Copyright 2011 California Institute of Technology. The work described in this presentation was performed at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.
Motivation

- Complexity of software continues to grow
- As complexity increases, the ability to test software becomes extremely difficult
- **Challenge:** Producing reliable software

[Source: Holzmann, “Software Size and Complexity” presentation]
Objective

- Model OSEK (Open Systems and the Corresponding Interfaces for Automotive Electronics) services for verification of multitask applications that are intended to be OSEK compliant.

OSEK services:
- Task management and Scheduling
- Resource Management
- Alarms and events
- Interrupt handling
- Conformance class specification
Approach

Idea: Avoid exploring unnecessary paths and states

- **Given**: $R$, a set of RTOS properties (i.e. policies, rules)
- **Construct**: $M_R = A_0, A_1, ..., A_n$, a finite automata model of the RTOS behavior as a composition of $n$ RTOS component behaviors and an application behavior $M_A$

**Property satisfied, $M \models \varphi$?**
- **Yes**: property satisfied
- **No**: generate counter example

**Property specification**: $G(p \Rightarrow X q)$

**Model checking problem**: $M \models \varphi$

**Challenge**: state explosion

---

System model, $M = (M_A, M_R)$

**Model checking algorithm**: $F$
Model Checking with SPIN

- **Promela (Process Meta Language)**
  - Language for modeling finite-state, concurrent systems
  - Finite and bounded data structures
  - Synchronization and message passing constructs

- **SPIN (Simple Promela Interpreter)**
  - Explicit state model checker
  - Simulation and verification engine for Promela models
  - Provides translation of Linear Temporal Logic properties to Promela models
**Example: Avoiding Priority Inversion**

- **Priority inversion** is a problem that occurs when a higher priority task is blocked inadvertently by a lower priority task.

```c
#include "osek.h"
byte data = 0;
RESOURCE(mutex, 10);
TASK (high, 10) {
    ...
    OSEK_GetResource(mutex);
    data--; printf("task high consumes data, %d\n", data); /* critical section - consume data */
    OSEK_ReleaseResource(mutex);
    ...
}
TASK (medium, 5) {
    do
    :: ...
        printf("task medium is running\n");
        assert(data == 0); /* check that data is produced*/
    od
}
TASK (low, 2) {
    do
    :: ...
        OSEK_GetResource(mutex);
        run high(); run medium();
        data++; printf("task low produces data, %d\n", data); /* critical section - produce data */
        OSEK_ReleaseResource(mutex);
        assert(0)
    od
}
```
#include “osek.h”
byte data = 0;
RESOURCE(mutex, 10);
TASK (high, 10) {
...
OSEK_GetResource(mutex);
data--; printf("task high consumes data, %d\n", data); /* critical section - consume data */
OSEK_ReleaseResource(mutex);
...
}
TASK (medium, 5) {
do:: ...
printf("task medium is running\n");
assert(data == 0); /* check that data is produced*/
}
do
}
TASK (low, 2) {
do:: ...
OSEK_GetResource(mutex);
run high(); run medium();
data++; printf("task low produces data, %d\n", data); /* critical section - produce data */
OSEK_ReleaseResource(mutex);
assert(0)
do
}
## Verification of the Application Model

**SPIN verification:**

Full statespace search for:

- never claim - (none specified)
- assertion violations +
- acceptance cycles - (not selected)
- invalid end states +

State-vector 36 byte, depth reached 28, errors: 0

- 21 states, stored
- 1 states, matched
- 22 transitions (= stored+matched)
- 8 atomic steps

hash conflicts: 0 (resolved)

**SPIN simulation output:**

- task low produces data, 1
- task high consumes data, 0
- task medium is running
- task medium is running
- task medium is running ...

- Verifies that priority inversion does not occur in the application model
- Ensures that the application model applies OSEK library services correctly
- Validates the implementation of the OSEK Ceiling Protocol
Summary

• Extended SPIN to support verification and simulation of priority based scheduling policies

• Constructed a library of services that provides an OSEK interface that is useful for the verification of multitask applications

• Future Work
  • Complete the OSEK specification (i.e. address interrupt handling services, alarms, events, conformance classes)
  • Apply, validate, and measure the performance of more practical examples
Steps to LTL Model Checking Using Automata
Theoretic Approach

• **Step 1**: translate both system model $M$ to automaton, $A_M$ and negated property, $\neg \varphi$ to automata, $A_{\neg \varphi}$

• **Step 2**: compose product automata, $P$,

$$P = L(A_M \times A_{\neg \varphi}) = L(A_M \cap A_{\neg \varphi}) = L(A_M) \cap L(A_{\neg \varphi})$$

• **Step 3**: check for emptiness
  • Is $L(A_M \cap A_{\neg \varphi}) = \emptyset \Rightarrow$ graph search?

  **Yes**: system model, $M$ is correct!
  **No**: system model, $M$ violates property $\varphi$, produce error trace