ICIUS 2010, Bali Indonesia
2010.11.03 ~ 11.05

Formal Verification of Process Communications in Operational Flight Program for a Small-Scale Unmanned Helicopter

JUNBEOM YOO
Dependable Software Laboratory
KONKUK University, Korea

2010.11.04

Contents

- Introduction
- Background
  - OFP (Operational Flight Program)
  - Model Checking using SPIN
- Formalization of the OFP in PROMELA
- Verification Results
- Conclusion and Future Work

INTRODUCTION

Introduction

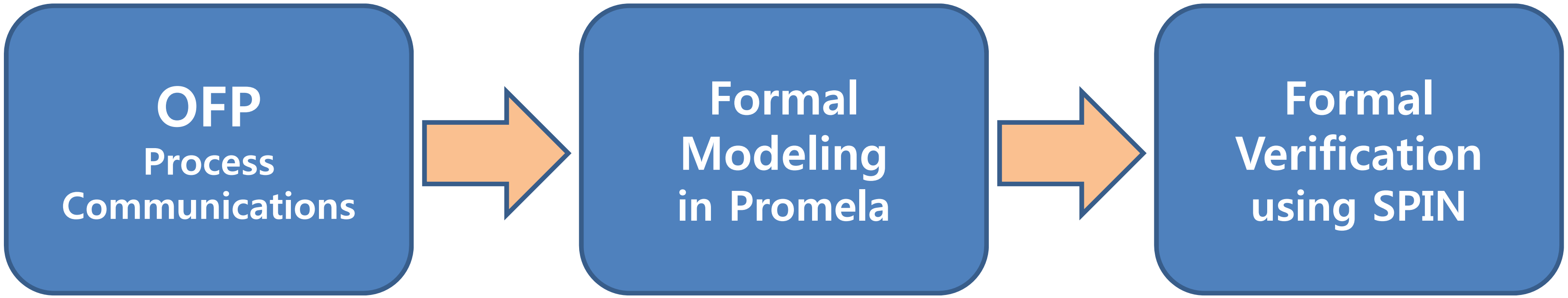
- HELISCOPE project
  - On-flight computing system
  - Embedded S/W
  - includes services for unmanned helicopter
  - for disaster response and recovery

- OFP (Operational Flight Program)
  - Subpart of HELISCOPE project
  - Software controller

Test flight of a small-scale unmanned helicopter

Introduction

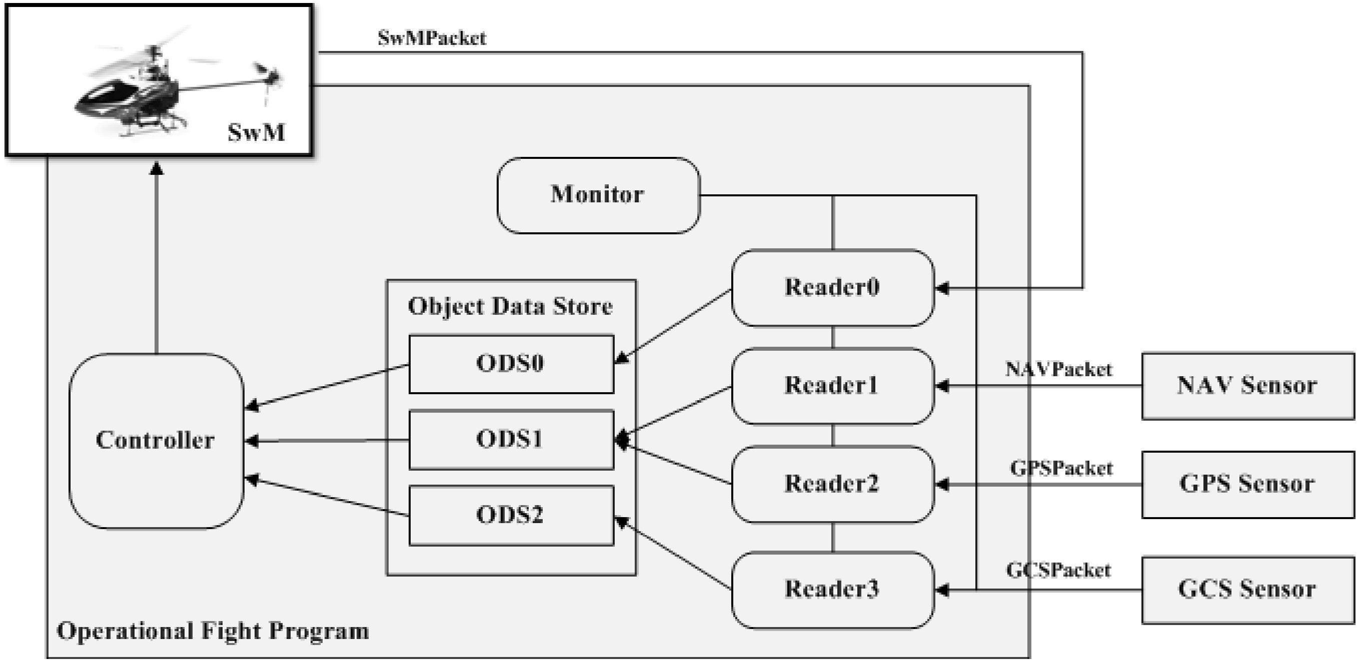
- The OFP should be safe, correct and stable.
- Formal Verification can help the OFP eliminate defects efficiently
  - Model checking using SPIN model checker
  - Target: process communications of the OFP



Dependable Software Laboratory

BACKGROUND

Operational Flight Program



- 3 ODS
  - 5 Shared Data Variables
- 6 Processes
  - Controller, Monitor and Readers

Operational Flight Program

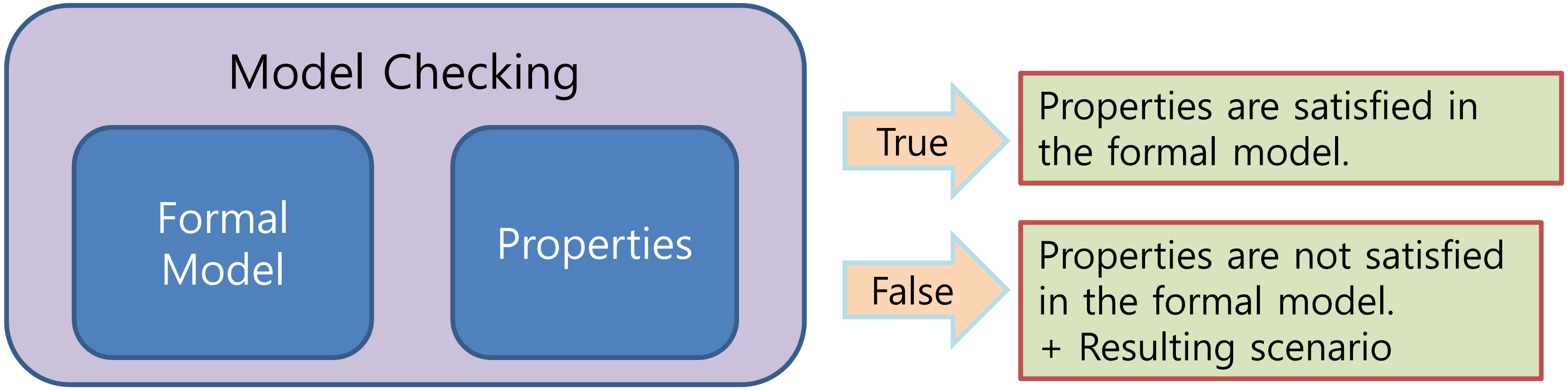
- Processes
  - 1 Monitor
    - Monitor four serial ports that connect with sensors
    - Manage semaphore to awake reader processes
  - 4 Readers
    - Reads packets from serial port and write data in object data store
    - Waits until semaphore is posted by monitor
  - 1 Controller
    - Reads data from object data store
    - Computes the data and operates servomotor

Operational Flight Program

- Object Data Store (ODS)
  - ODS0
    - Current flight information
    - reader0 and controller processes access
  - ODS1
    - GPS and Navigation information
    - reader1, reader2, reader3 and controller processes access
  - ODS2
    - Flight Mode and destination information
    - reader3 and controller processes access

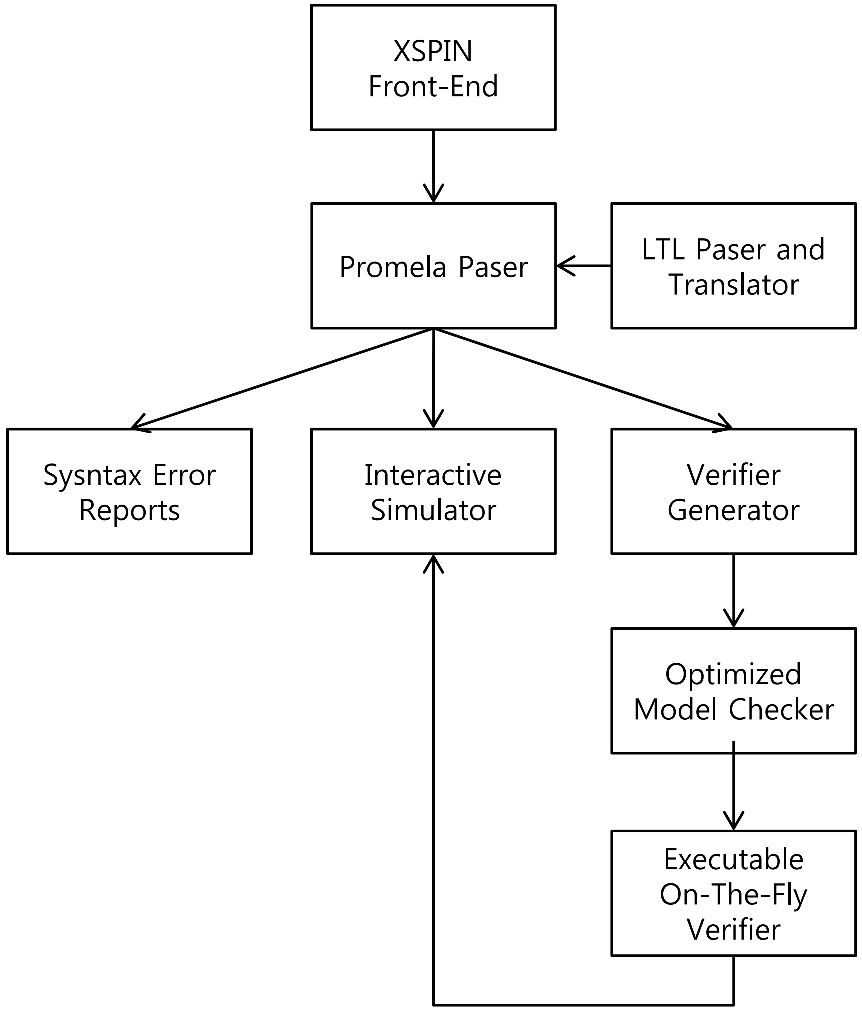
Model Checking

- Model Checking
  - An automatic technique for verifying finite state systems against properties
    - Formal model of a system
    - Temporal logic for specifying properties of the system



Model Checker SPIN

- Model Checker SPIN
  - Formal verification system
  - Supports design and verification of distributed/current software systems
  - XSPIN: graphical front-end
  - Verification & simulation
- Model definition
  - PROMELA (PROcess MEta LAnguage)
- Properties definition
  - LTL (Linear Temporal Logic)
  - Assertion statement

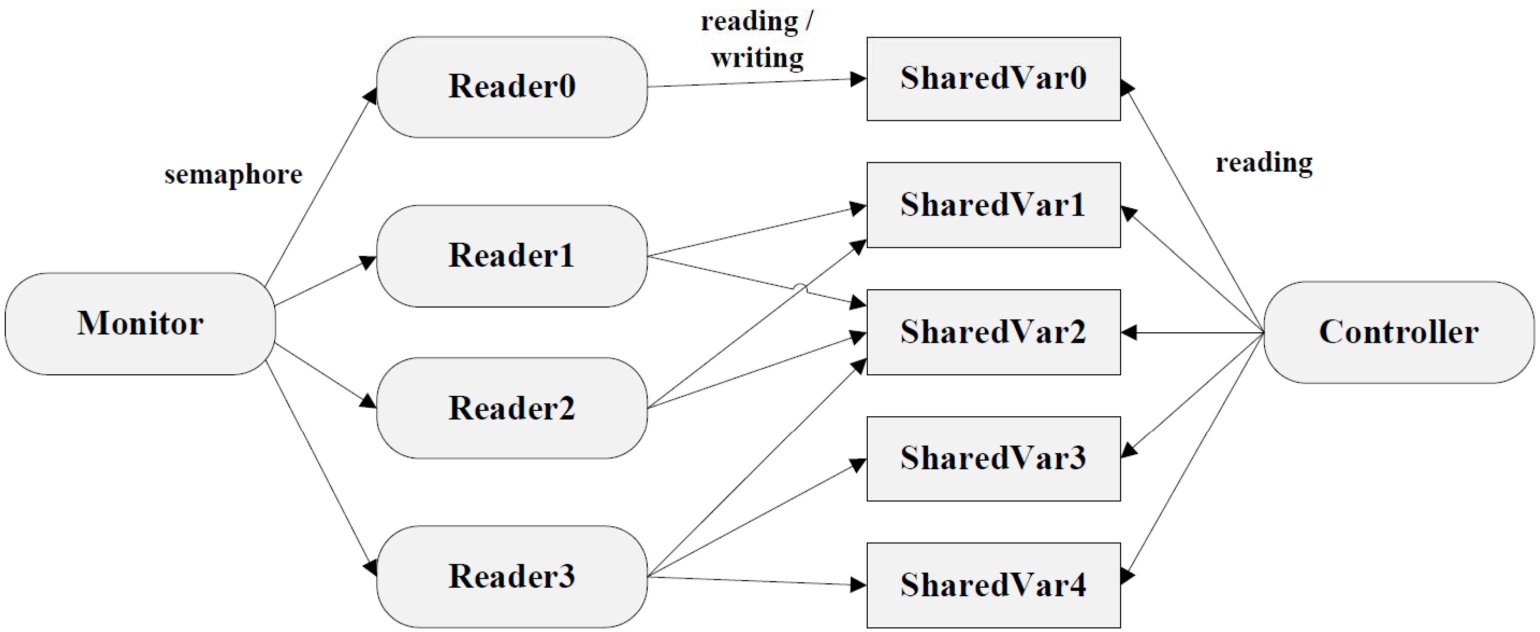


FORMALIZATION OF THE OFP IN PROMELA

1st Formalization of the OFP

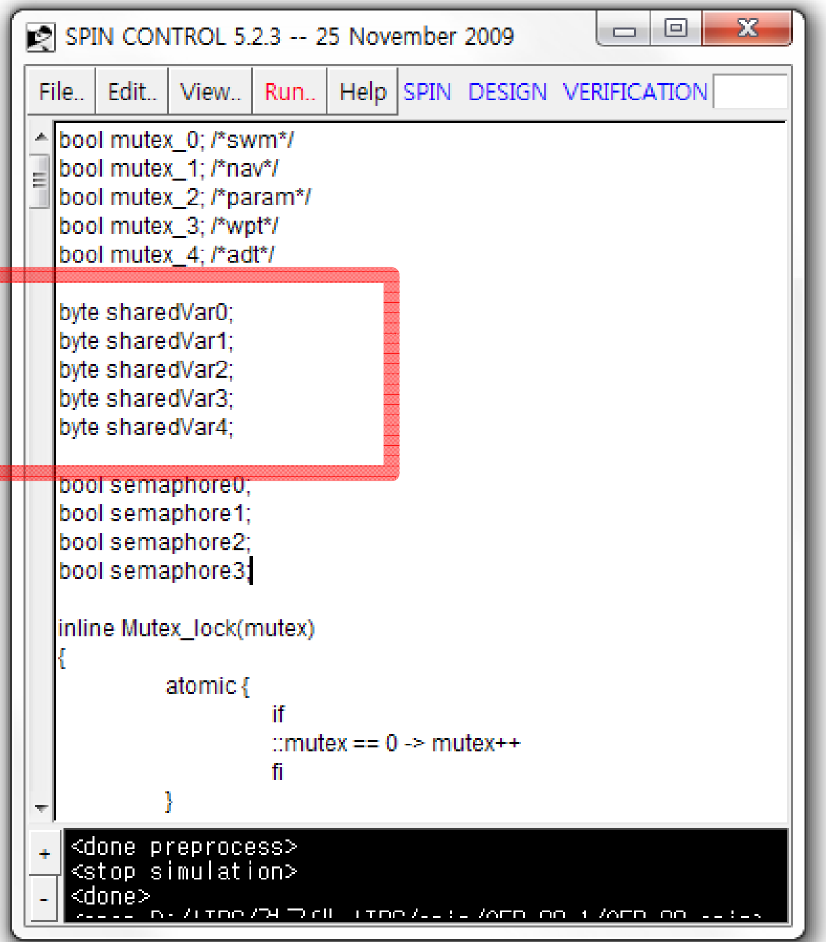
5 shared data variables

6 processes



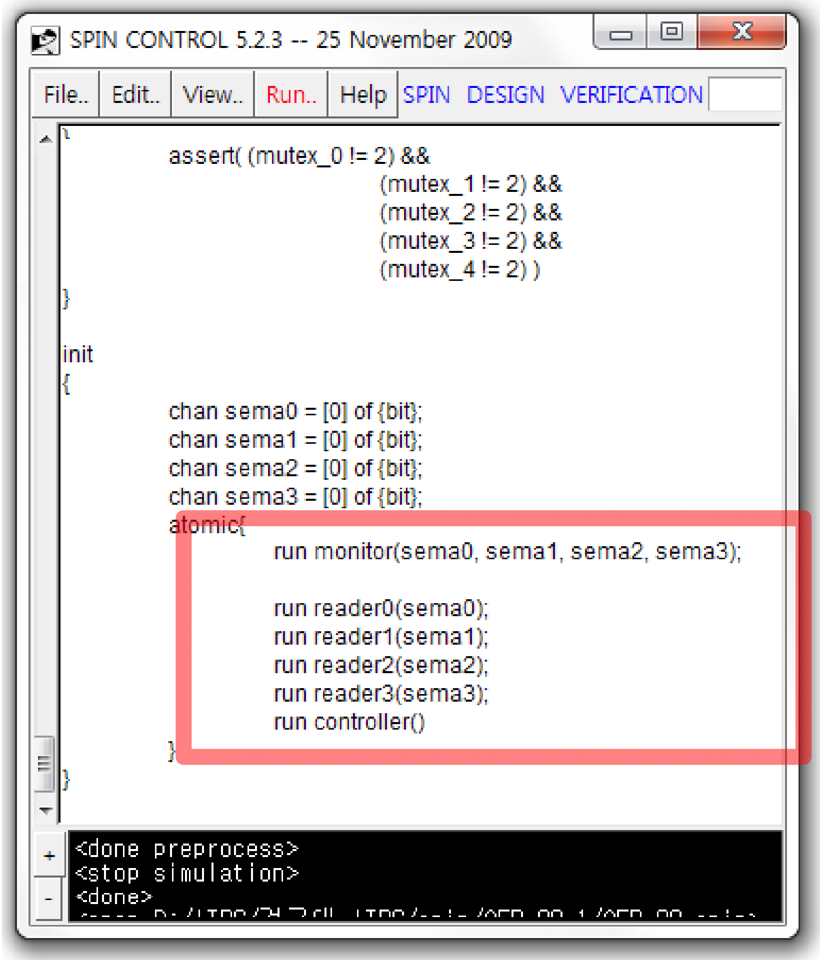
Formalization of the OFP in PROMELA

- 5 shared data variables
- Accessed by 5 processes
  - Reader0~3
  - Controller
- Monitored by 1 process
  - Monitor
- Processes can access variables using *mutex*



Formalization of the OFP in PROMELA

- 6 Processes
- Monitor
  - 4 channel to connect with readers
- Reader 0~3
  - 1 channel for each to connect with monitor
  - access shared variables
- Controller
  - access all shared variables



Properties for Verification

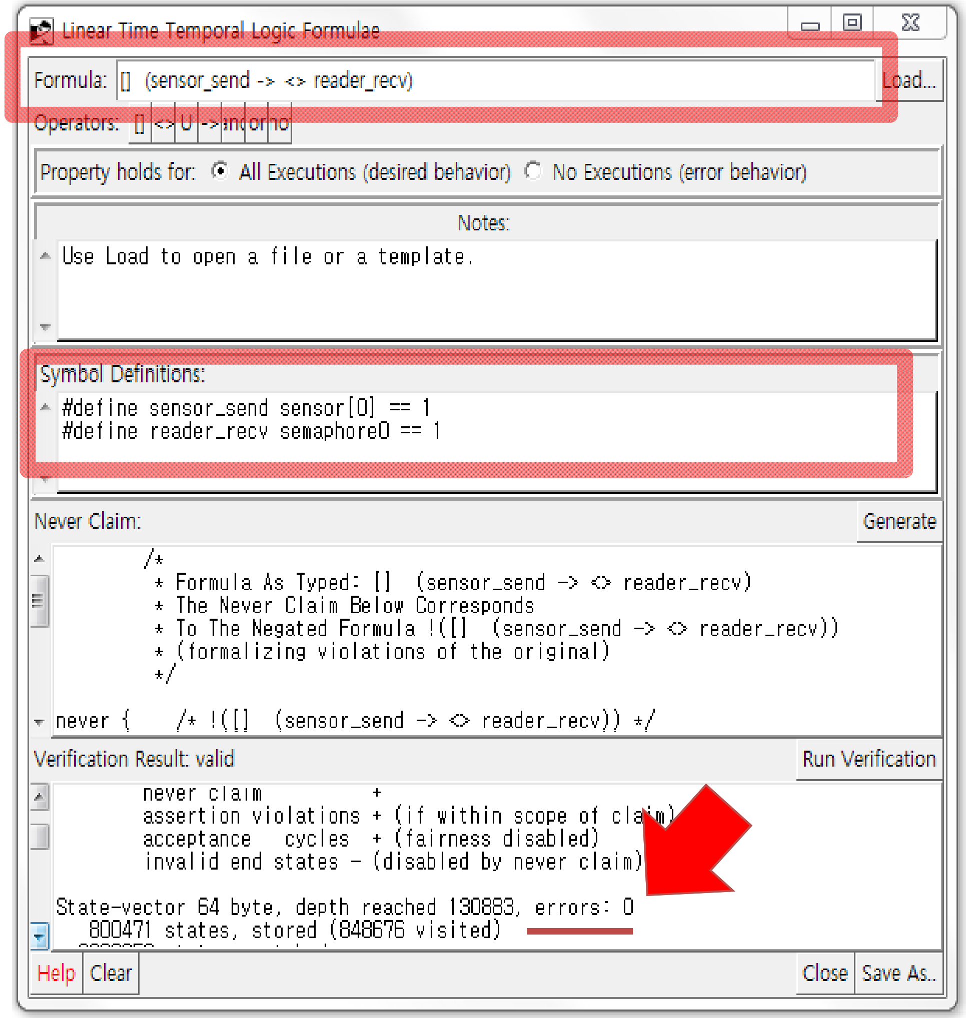
The process monitor's Semaphores on four reading processes should function correctly.

→ Reader process can read data from sensor eventually.
→ In all stats, if *sensor_send* holds, then eventually either *read_recv* will hold.

LTL Property: [ ] (sensor_send -> <> read_recv)

VERIFICATION RESULTS

Verification Results – LTL Property

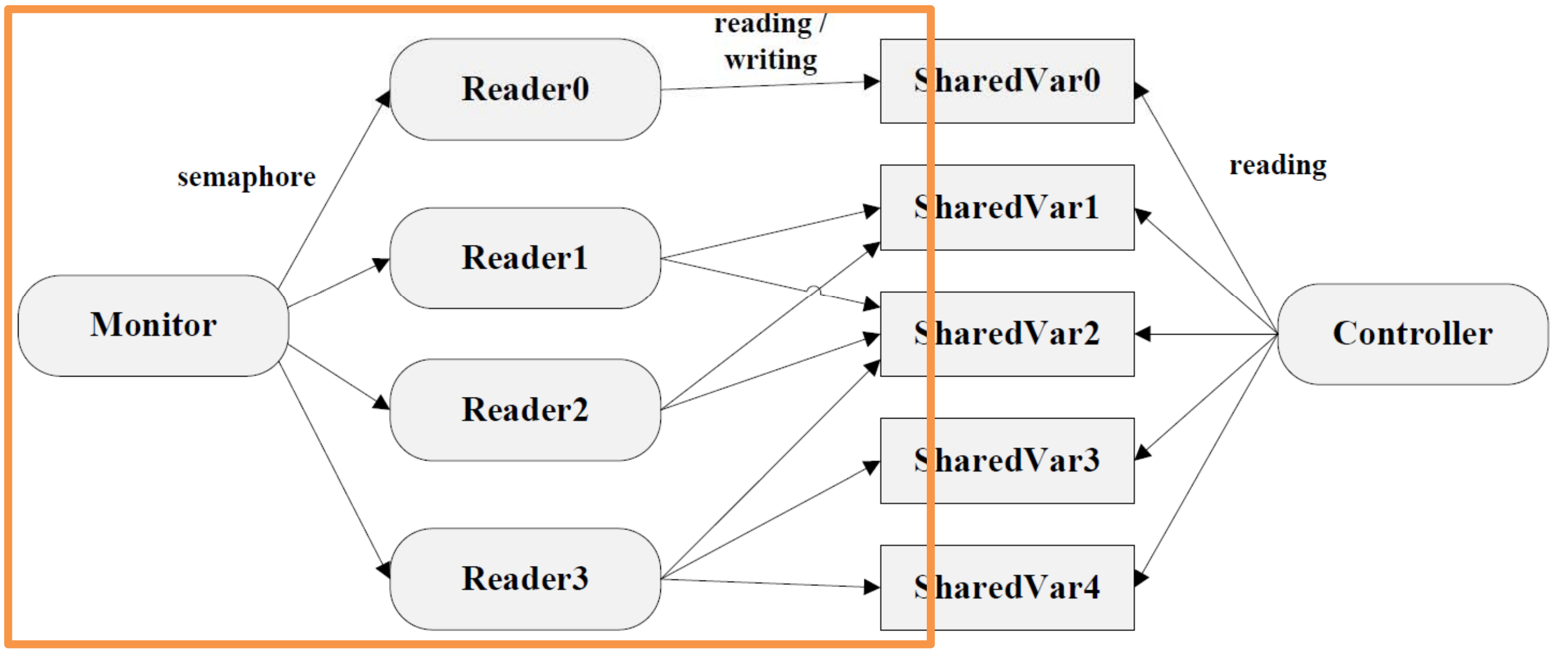


- LTL Property

[] (sensor_send -> <> read_recv)

#define sensor_send sensor[0] == true
#deinfe reader_recv reader0.sema == true

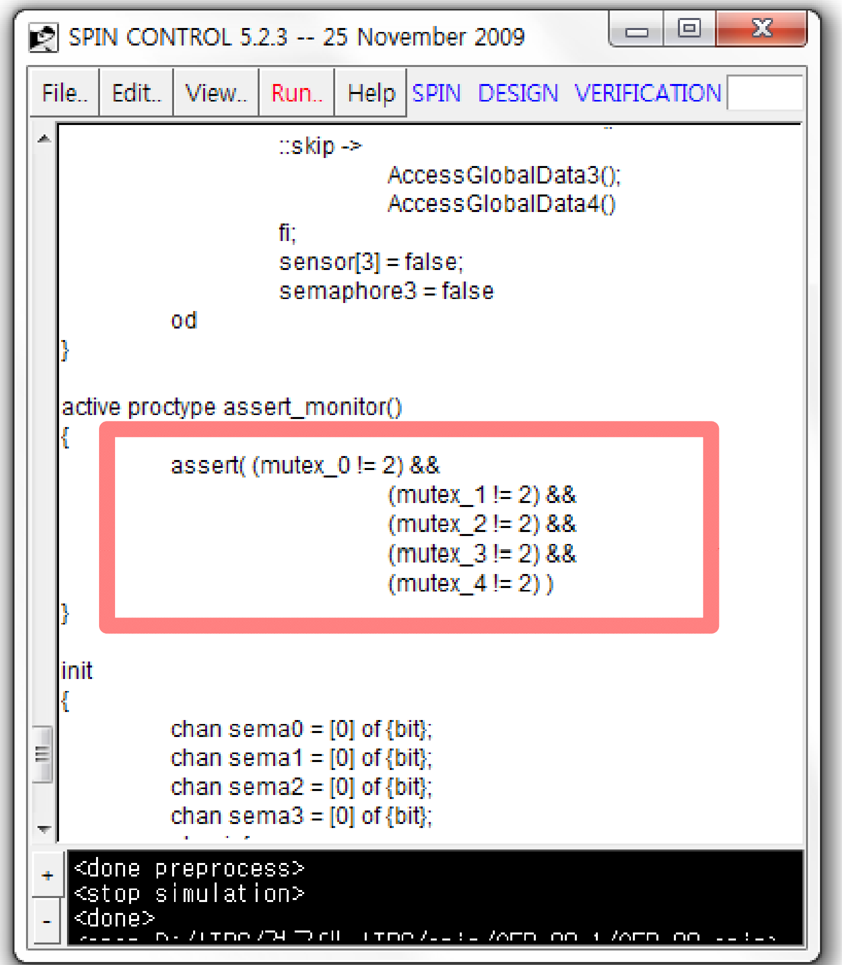
- Verification result
  - No error
  - All data from sensors is always eventually read by reader process.
  - *monitor* process manages *semaphores* correctly.



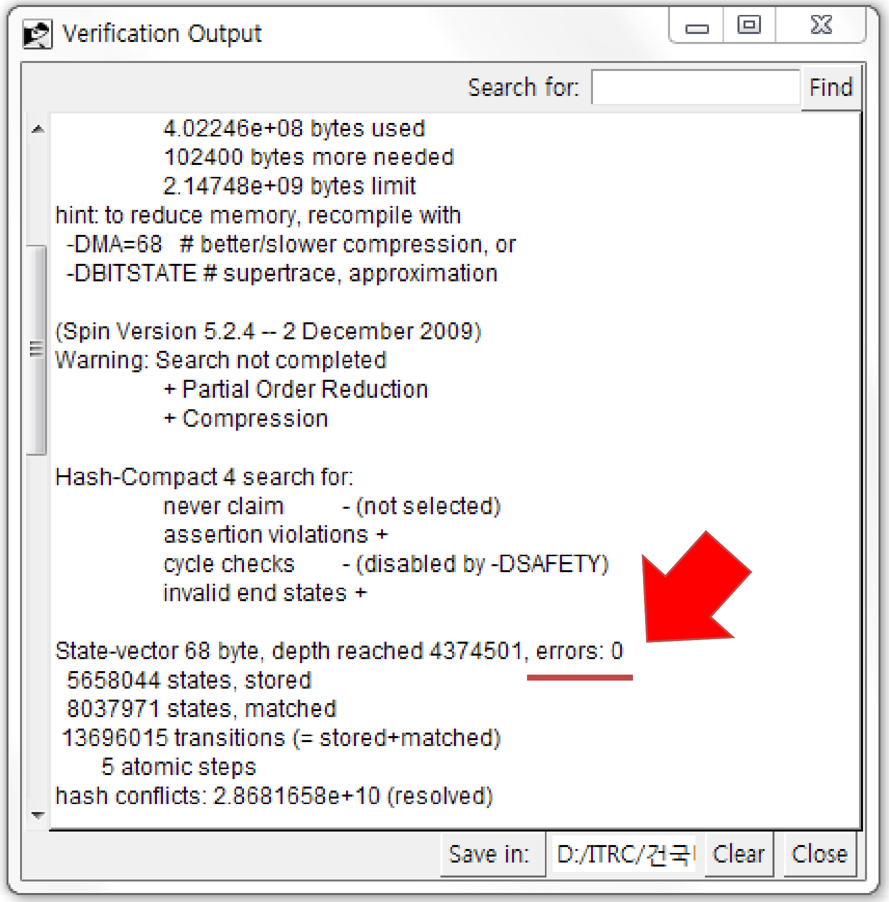
reading /
writing
Reader0
SharedVar0
semaphore
reading
Reader1
SharedVar1
Monitor
SharedVar2
Controller
Reader2
SharedVar3
Reader3
SharedVar4

Verification Results – Assertion Statement

- Assertion statement
  - 5 shared data variables should be accessed mutually exclusively by *reader 0~3* processes and *controller* process.
  - Assert whether two or more processes access a variable at once.
  - Each variable adds 1 to *mutex* each time it's accessed by the processes.
  - Therefore, they all should be 0 or 1.



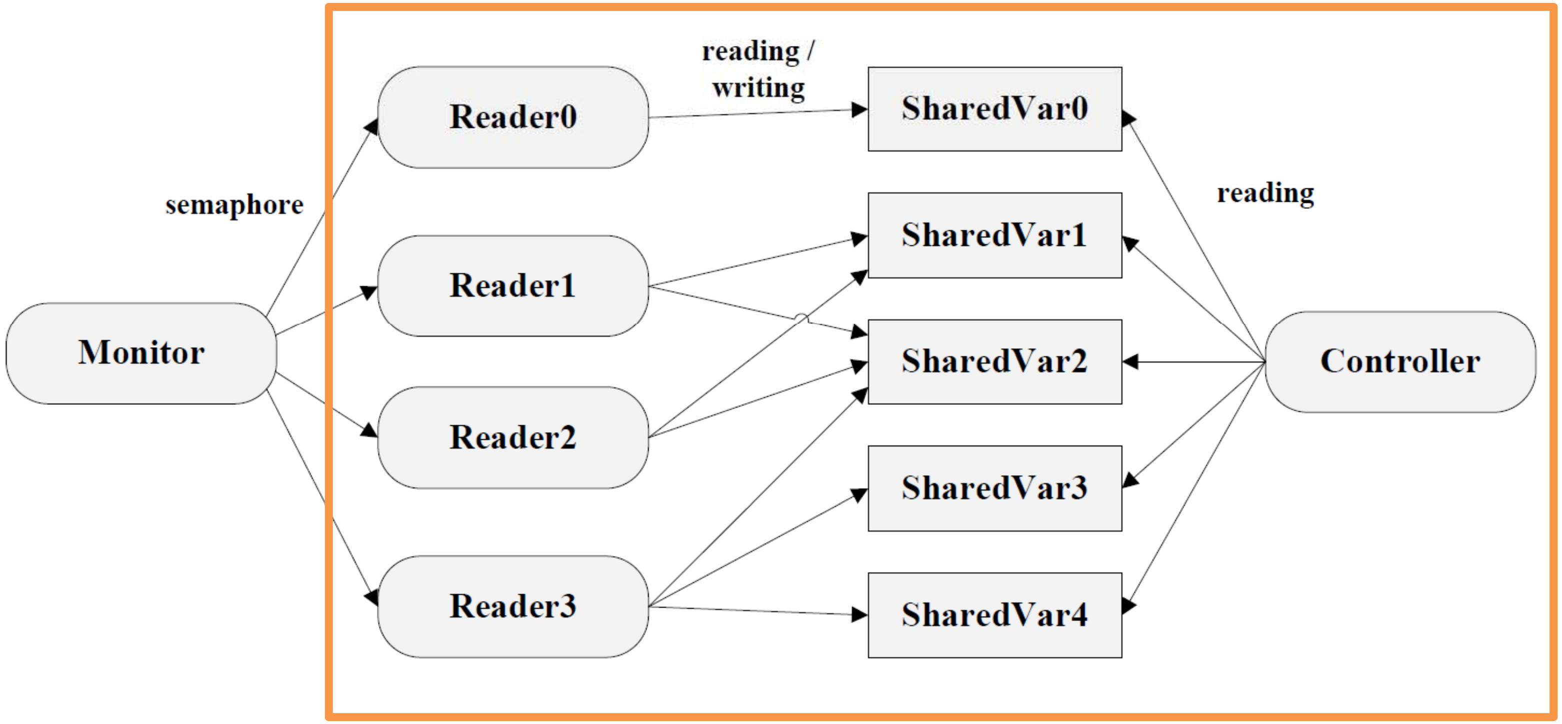
Verification Results – Assertion Statement



- *assert_monitor* process monitors 5 *mutexes.*

```
active proctype assert_monitor()
{
        assert( (mutex_0 != 2) &&
        (mutex_1 != 2)&&
        (mutex_2 != 2)&&
        (mutex_3 != 2)&&
        (mutex_4 != 2) )
}
```

- Verification result
  - No error
  - *controller* and 4 *reader* processes access shared variables mutually exclusively.


Monitor
semaphore
Reader0
Reader1
Reader2
Reader3
reading / writing
SharedVar0
SharedVar1
SharedVar2
SharedVar3
SharedVar4
reading
Controller

Conclusion and Future Work

- Formal Verification for OFP
  - Formal model of process communications
    - 5 shared data area
    - 6 processes
  - Result of verification
    - *monitor* process manages *semaphores* correctly.
    - *controller* and *reader* processes access shared variables mutually exclusively.
    - No possible error on semaphore operations and shared data
- Future work
  - Formal verification focused on timing constraint
  - UPPAAL
    - Timed automata