

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







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







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*

SPIN CONTROL 5.2.3 25 November 2009								
File	Edit	View	Run	Help	SPIN	DESIGN	VERIFICATION	
<pre>bool mutex_0; /*swm*/ bool mutex_1; /*nav*/ bool mutex_2; /*param*/ bool mutex_3; /*wpt*/ bool mutex_4; /*adt*/</pre>								
b b b b b	/te share /te share /te share /te share /te share	dVar0; dVar1; dVar2; dVar3; dVar4;						
<pre>bool semaphore0; bool semaphore1; bool semaphore2; bool semaphore3; inline Mutex_lock(mutex) { atomic {</pre>								
+ <done preprocess=""> <stop simulation=""> - <done></done></stop></done>								





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

SPIN CONTROL 5.2.3 25 November 2009 💷 💷 🔀						
File Edit View Run Help SPIN DESIGN VERIFICATION						
<pre> assert((mutex_0 != 2) && (mutex_1 != 2) && (mutex_2 != 2) && (mutex_3 != 2) && (mutex_3 != 2) && (mutex_4 != 2)) init { chan sema0 = [0] of {bit}; chan sema1 = [0] of {bit}; chan sema2 = [0] of {bit}; chan sema2 = [0] of {bit}; chan sema3 = [0] of {bit}; chan sema3</pre>						
atomic{ run monitor(sema0, sema1, sema2, sema3); run reader0(sema0); run reader1(sema1); run reader2(sema2); run reader3(sema3); run controller()						
<pre></pre>						





Properties for Verification

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

 \rightarrow Reader process can read data from sensor eventually. \rightarrow In all stats if sensor send holds then eventually either real

→ 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

Linear Time Temporal Logic Formulae	• 🛛 🕺 •	LTL Property
Formula: [] (sensor_send -> <> reader_recv)	Load	
Operators: [] <> U -> in or no		
Property holds for: All Executions (desired behavior) No Executions (error behavior)	or)	[] (gamgan gand) () mood many)
Notes:		[] (sensor_send -> <> read_recv)
Use Load to open a file or a template.		
·		#define sensor_send sensor[0] == true
Symbol Definitions:		#deinfe reader_recv reader0.sema == true
<pre>#define sensor_send sensor[0] == 1 #define reader_recv semaphore0 == 1</pre>		
Never Claim:	Generate	Verification result
/* * Formula As Typed: [] (sensor_send -> <> reader_recv)		 No error
* The Never Claim Below Corresponds * To The Negated Formula !([] (sensor_send -> <> reader_r	ecv))	- All data from sensors is always
+ (formalizing violations of the original) +/		eventually read by reader
<pre>- never { /* !([] (sensor_send -> <> reader_recv)) */</pre>		eventually lead by leader
Verification Result: valid	Run Verification	process.
assertion violations + (if within scope of clawn) acceptance cycles + (fairness disabled)		 monitor process manages
invalid end states - (disabled by never claim)		semaphores contectly.
State-vector 64 byte, depth reached 130883, errors: 0 800471 states, stored (848676 visited)		
Help Clear	Close Save As	
LEDENDARIE SOFTWARE		









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.

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











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

