Testing concurrent software using Java PathFinder (JPF): An implementation of the Stable Matching Algorithm using the Actor model
Ihar Laziuk1, Devin Etcitty1; Steven Lauterburg2, Ph.D. (Faculty Mentor)
1Columbia University, 2Salisbury University

Introduction
The problem pertains to Actor-based software testing and determining which different message scheduling produces optimal results.

Purpose
Concurrent software allows for efficient handling of large data, including remote machines. However, it is difficult to design and predict accurate results with multiple threads running. Tools for concurrent software testing help to explore and analyze all possible output scenarios within a given application before its deployment.

Methods & Materials
The “Stable Matching” algorithm was implemented using ActorFoundry, an academic Actor-based framework for developing concurrent applications. The algorithm was tested in NASA’s Java PathFinder actor module (jpf-actor) in order to explore different orderings of message processing.

“Stable Matching”
A matching M is stable if there is no pair (m, w) of man m and a woman w satisfying the following conditions:
• m and w are not married in M
• m prefers w to his current partner in M
• w prefers m to her current partner in M

Simple Actor Model Diagram

An Actor is an atomic unit of computation for which the following axioms hold:
• Can send and receive messages
• Create new actors
• How to process next message

Results for Different Message Scheduling
Self-destroying actor classes leads to more messages delivered and more new actors created.

Another combination of the same number of self-destroying actor classes yields different results.
(e.g., for 2 actor-classes killed order A: 1 & 3 vs order B: 1 & 2, we obtain different numbers)

Applying JPF’s different heuristics for message scheduling (LIFO, FIFO, EAC, LAC) demonstrated no differences.

Example – Message Schedule

Conclusions
We implemented an Actor-model based Stable Matching algorithm using the ActorFoundry framework and tested the implementation using NASA’s Java PathFinder testing tool.

Exhaustively exploring all possible message delivery schedules can be very costly. To manage this cost, the tool’s jpf-actor module provides multiple dynamic partial order reduction algorithms and message scheduling heuristics to more efficiently handle messages for the application to be tested.

Our experiments applying dynamic partial order reduction algorithms and message ordering schedules showed that use of DPOR reduces the time needed to exhaustively test the program.

References
5. Based on the Java implementation at http://rossetzodiac.org/wiki/Stable_marriage_problem

Acknowledgements
This research is funded by the National Science Foundation CCF-1460960 under the Research Experience for Undergraduates Program.