Automatic Repair of Concurrency Bugs

Jeremy S. Bradbury, **Kevin Jalbert**

Software Quality Research Group
University of Ontario Institute of Technology
Oshawa, Ontario, Canada

{jeremy.bradbury, kevin.jalbert}@uoit.ca

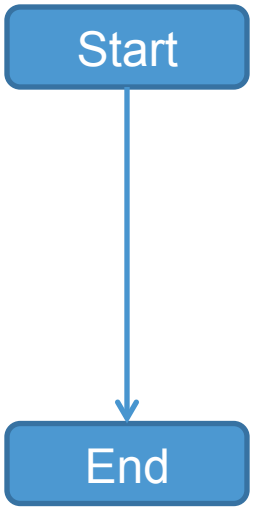
http://faculty.uoit.ca/bradbury/sqrg/

UOIT CHALLENGE INNOVATE CONNECT SQR GROUP

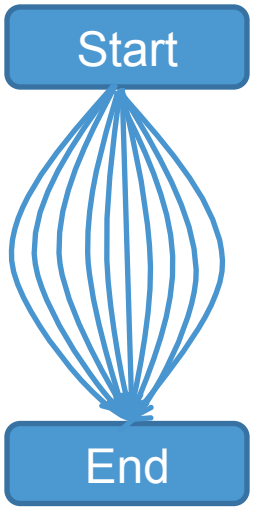
Funding provided by: NSERC CRSNG

Introduction to Concurrency

- Sequential software has only one thread
- Concurrent software has multiple threads
- Threads are the execution of source code



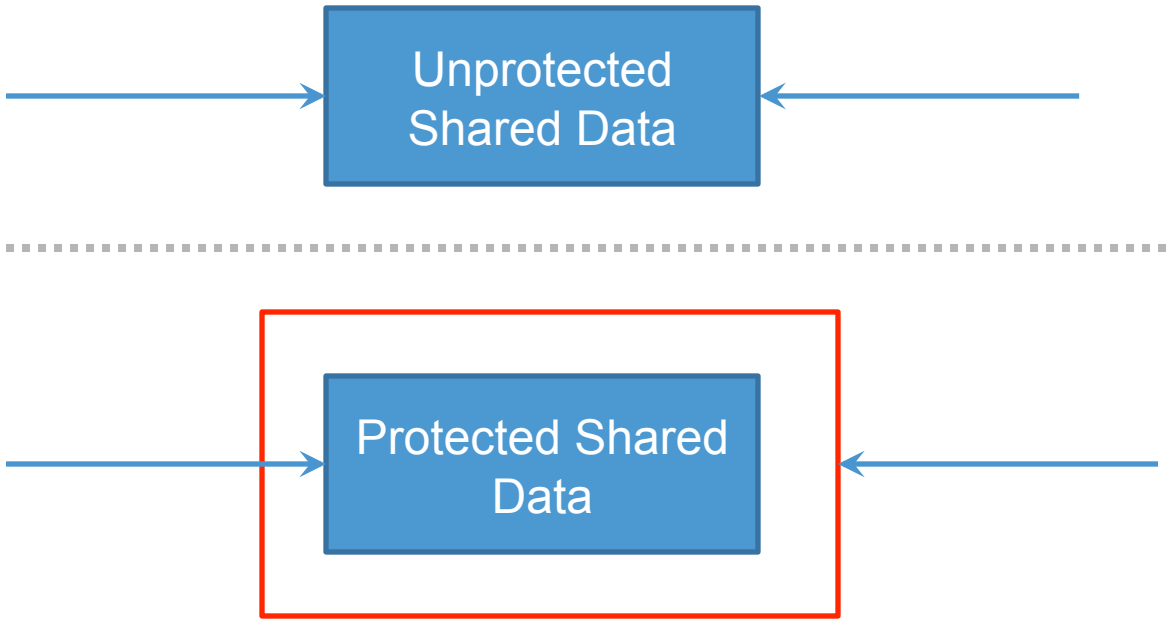
Sequential Software
with one thread



Concurrent Software
with multiple threads

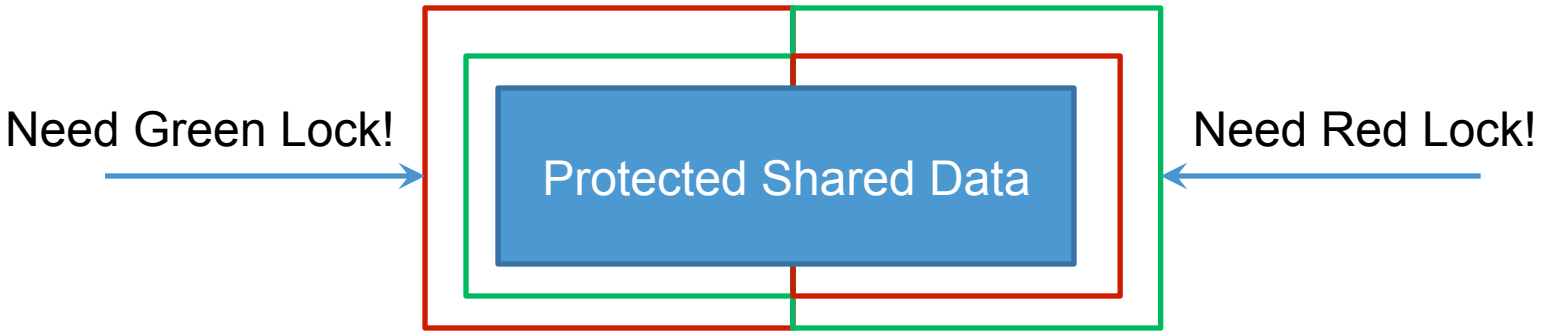
Concurrency Bugs – Data Races

- Two or more threads access unprotected shared data, resulting in inconsistent access to the shared data



Concurrency Bugs – Deadlock

- Two or more threads try to acquire the lock objects for a protected region
- The order of the lock acquisition prevents the other thread from acquiring the needed lock
- Both threads are waiting for the other's lock



Motivation

- Difficult to detect/test/fix concurrency bugs
  - Concurrency bugs appear intermittently
  - Unclear on repair approach
- Automated repair is attractive

Previous Related Work

- Related work done by Andrea Arcuri et al. [AY08] and Westley Wiemer et al. [WNLF09]
  - Both present ideas of automatic bug repair
  - Genetic Programming and Co-Evolution
- Demonstrated to work on sequential software

[AY08] A. Arcuri and X. Yao, "A novel co-evolutionary approach to automatic software bug fixing," in *Proc. of CEC*, 2008, pp. 162–168.
[WNLF09] W. Weimer, T. Nguyen, C. Le Goues, and S. Forrest, "Automatically finding patches using genetic programming," in *Proc. of ICSE*, 2009, pp. 364–374.

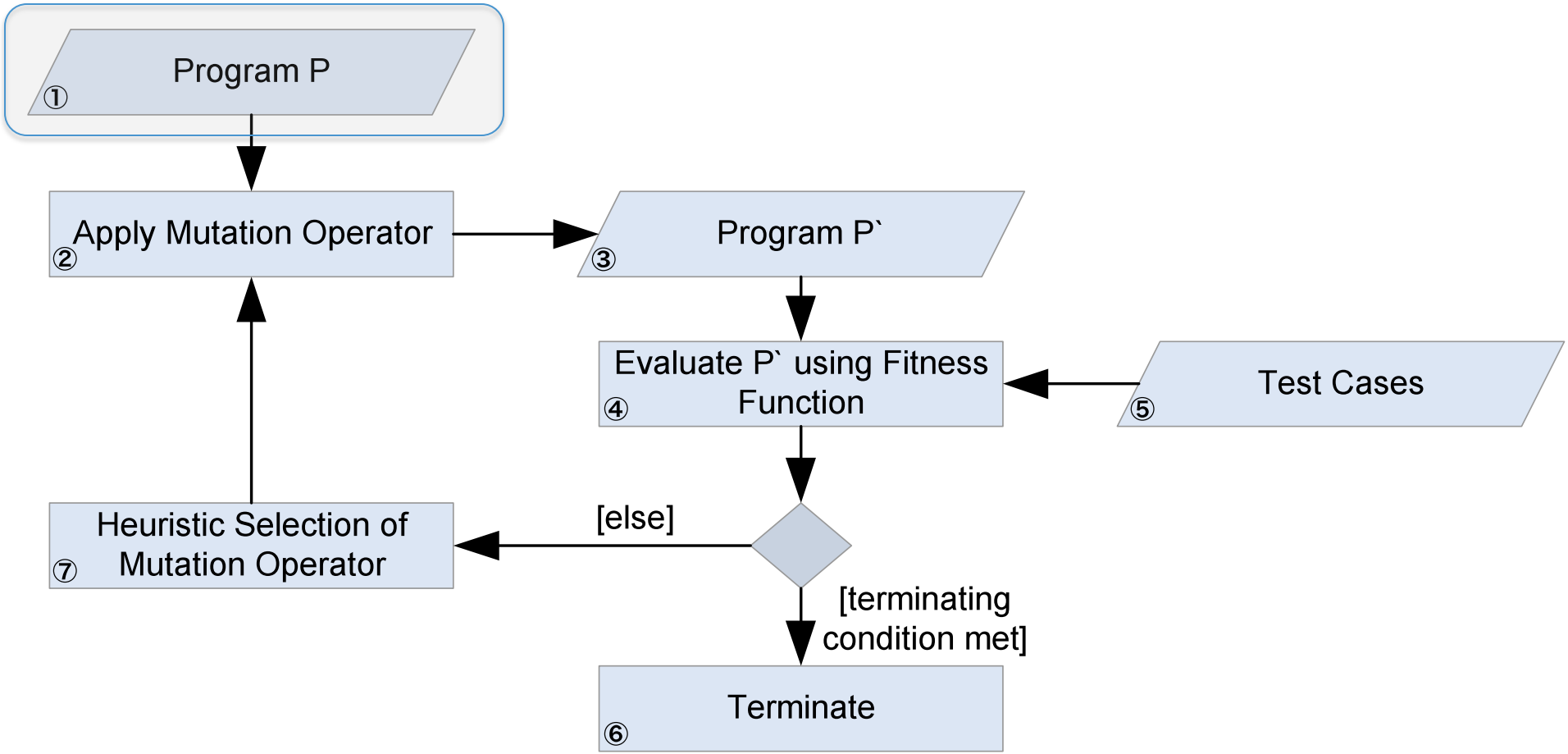
Research Goal

- Extend previous work and automatically repair concurrency bugs (data race and deadlock bugs) using genetic programming

Algorithm's Requirements

- Concurrent Program with Test Cases
- Mutation Operators
- Fitness Function
- Termination Condition
- Operator Selection Process

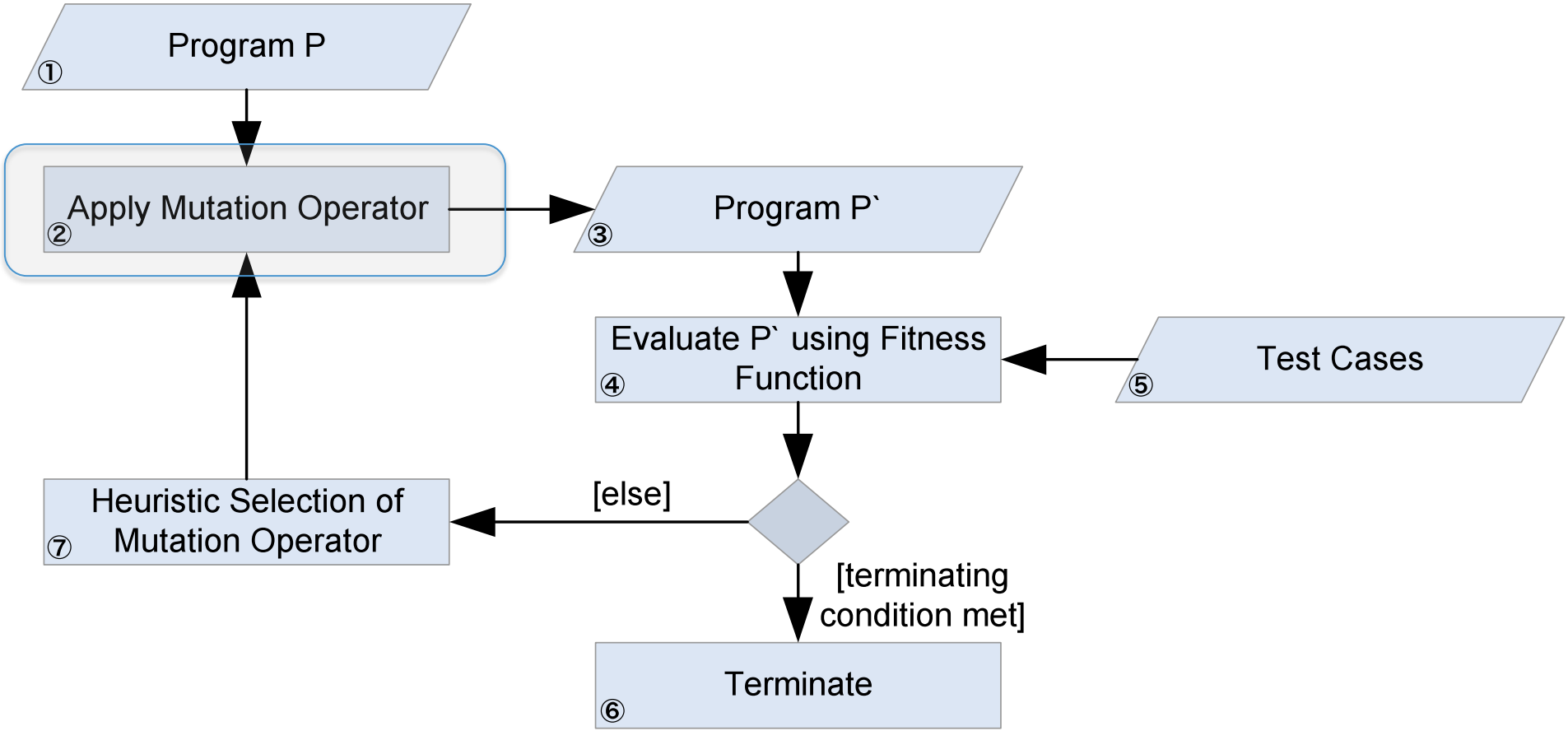
Putting it Together



Concurrent Program with Test Cases

- A concurrent program that is executable
- A set of test cases which capture the behaviour of the bug

Putting it Together – cont.



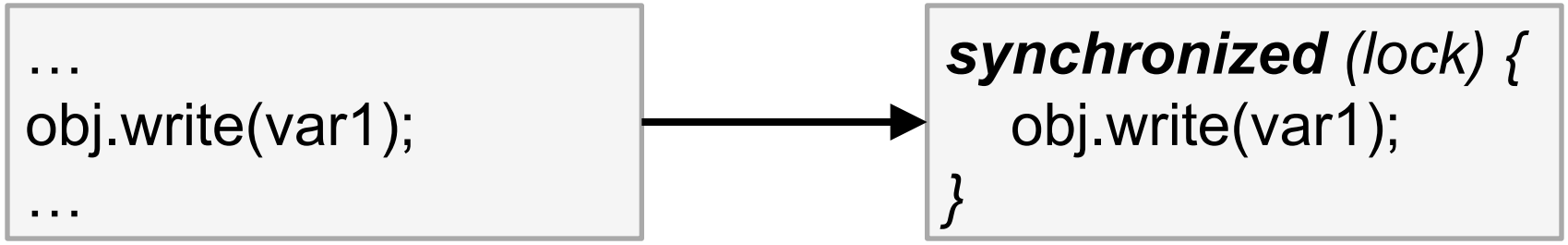
Mutation Operators

- Influenced by concurrency mutation operators [BCD06]
- Identified three operators
  1. Synchronize an unprotected shared resource
  2. Expand synchronization regions to include unprotected source code
  3. Interchange nested lock objects

[BCD06] J. S. Bradbury, J. R. Cordy, and J. Dingel, "Mutation operators for concurrent Java (J2SE 5.0)," *in Proc. of Mutation*, 2006, pp. 83–92.

Mutation Operator #1

- *Synchronize an unprotected shared resource*
  - Targets and fixes Data Race bugs



Mutation Operator #2

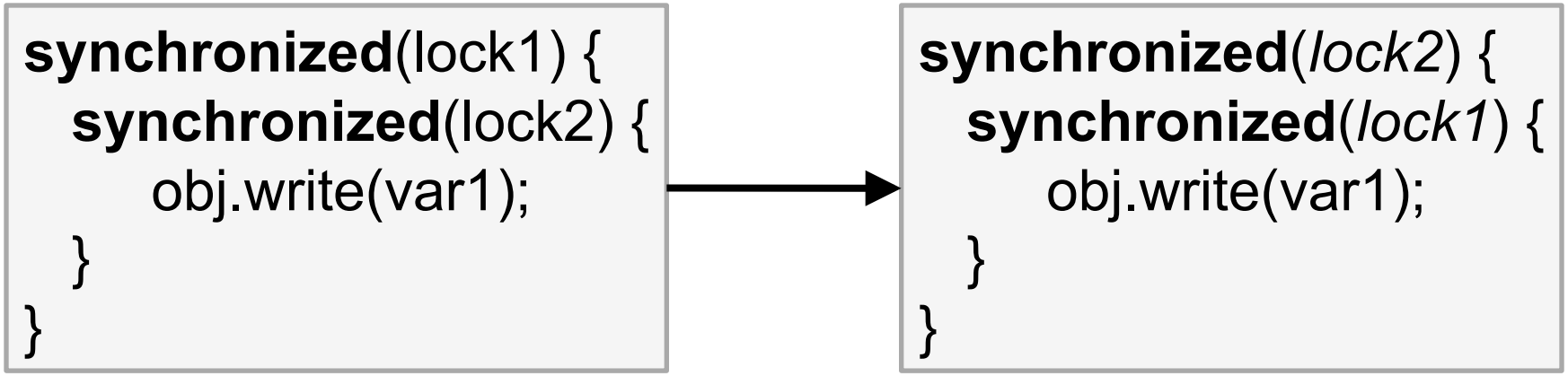
- *Expand synchronization regions to include unprotected source code*
  - Targets and fixes Data Race bugs

```
synchronized(lock) {
    obj.write(var1);
}
obj.write(var2);
```

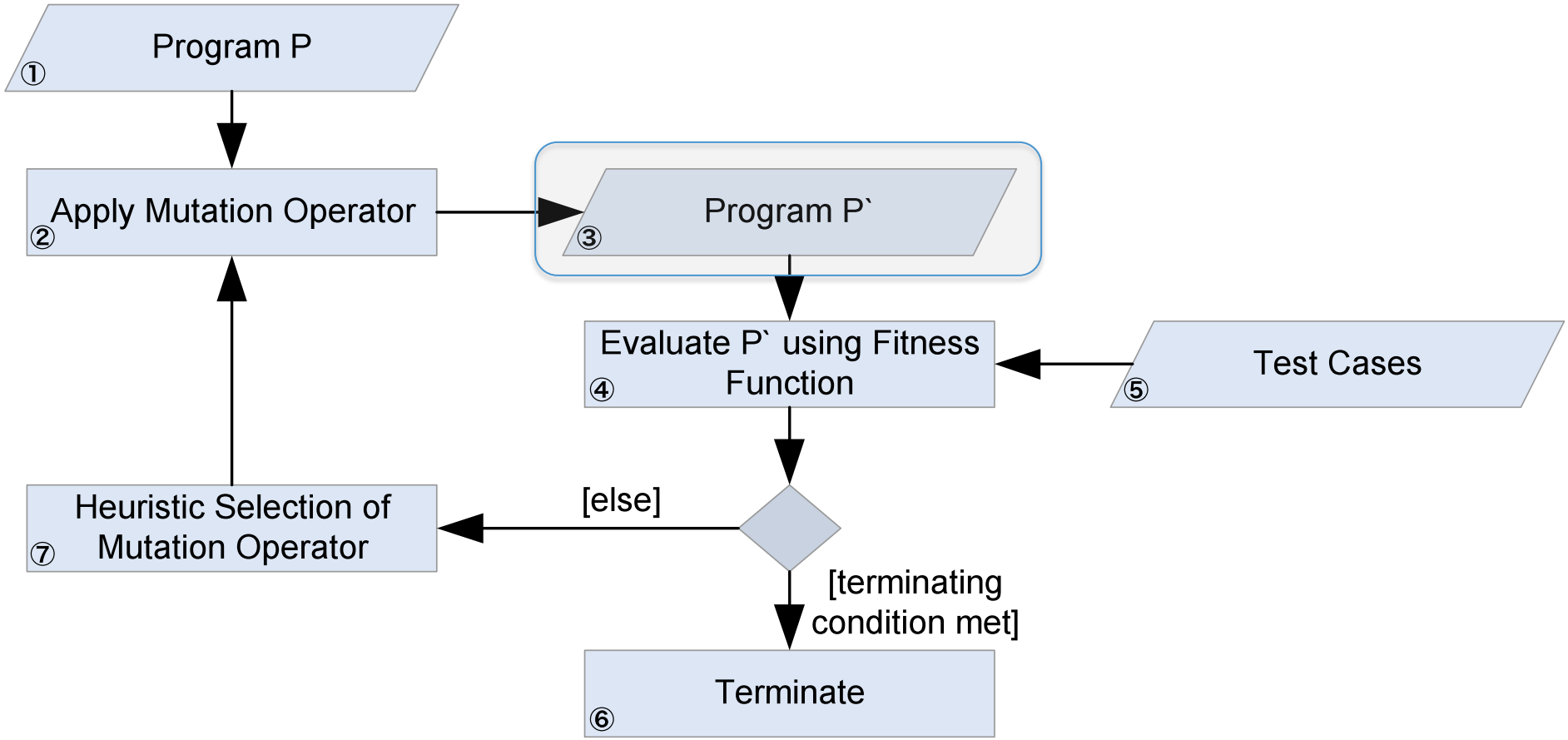
```
synchronized(lock) {
    obj.write(var1);
    obj.write(var2) ;
}
```

Mutation Operator #3

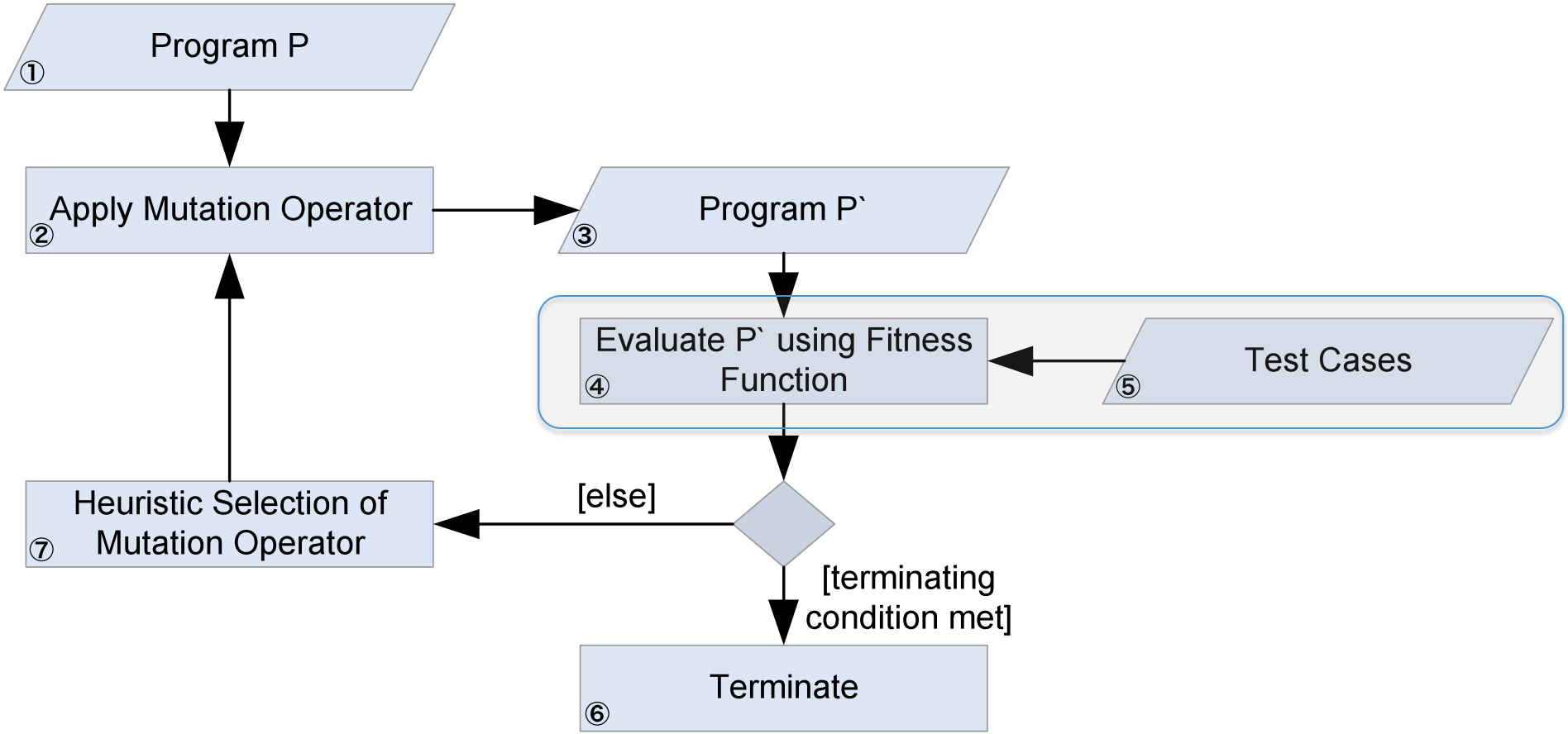
- *Interchange nested lock objects*
  - Targets and fixes Deadlock bugs



Putting it Together – cont.



Putting it Together – cont.



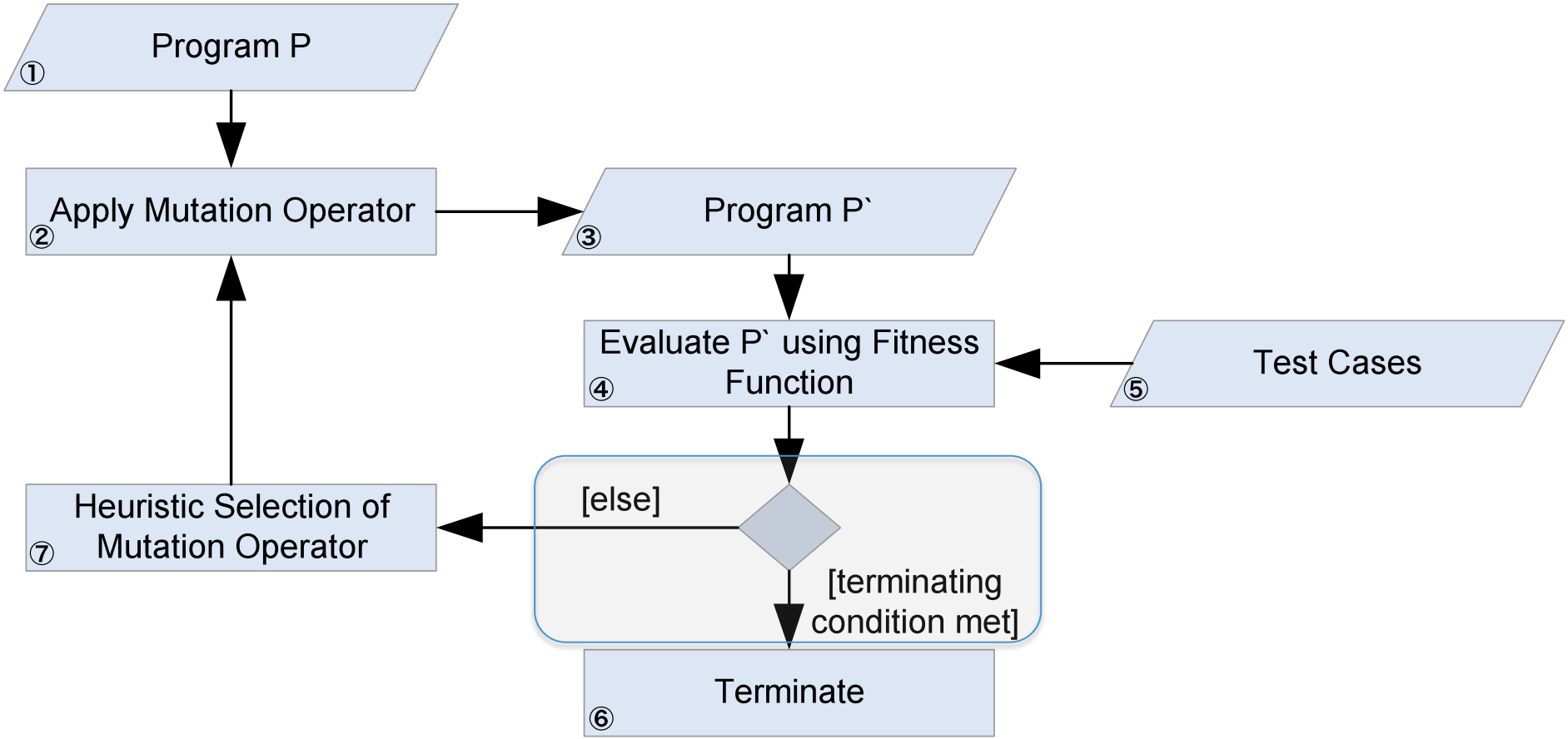
Fitness Function

- Test program using test cases
  - Run test cases multiple times with IBM's ConTest testing tool
  - Want to explore as many thread interleavings

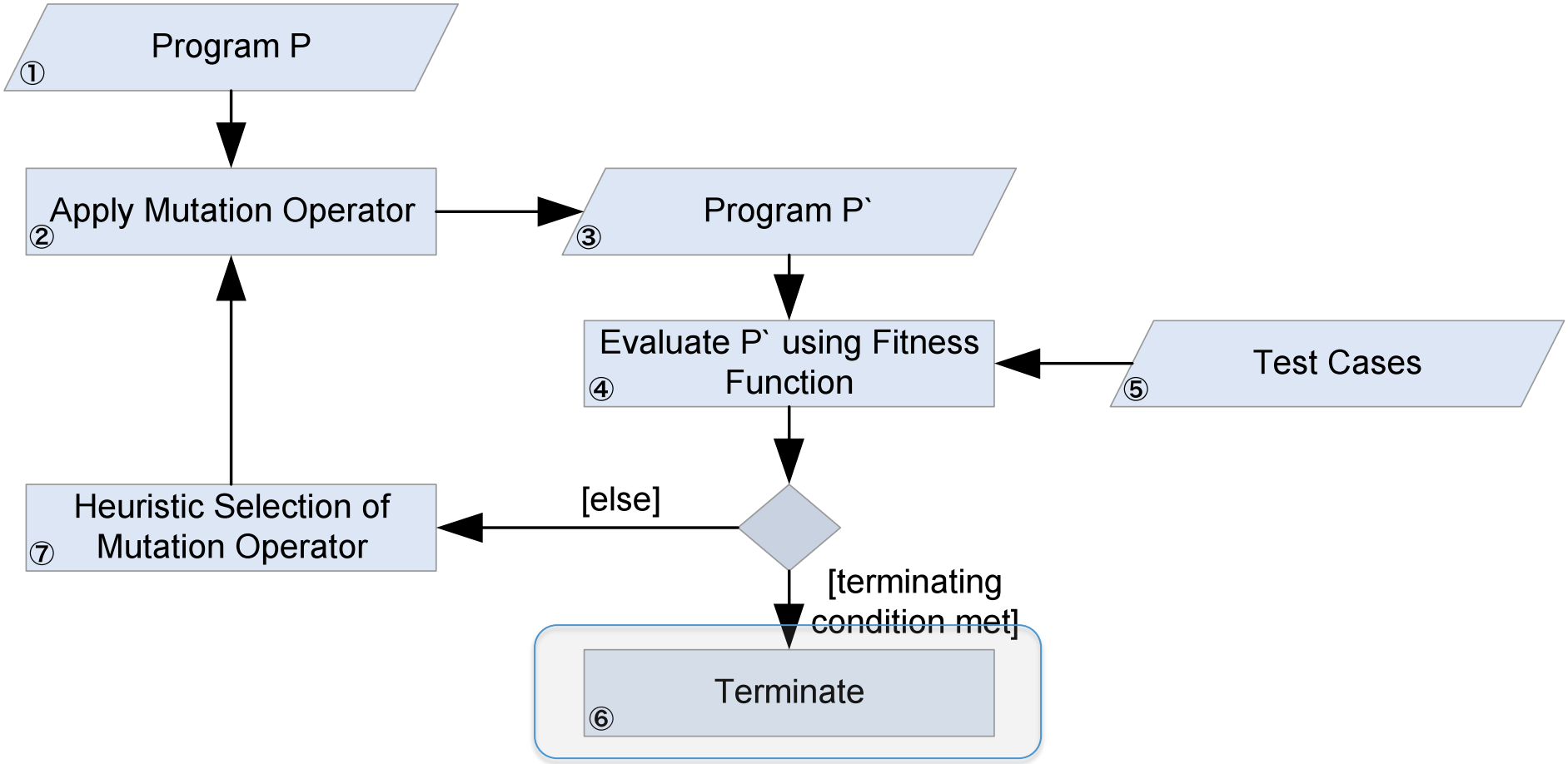
$$fitness(P) = \sum_{i=0}^{n} \frac{interleavings\ without\ a\ bug}{total\ \#\ of\ interleavings\ tested}$$

$$n = \#\ of\ Test\ Cases$$

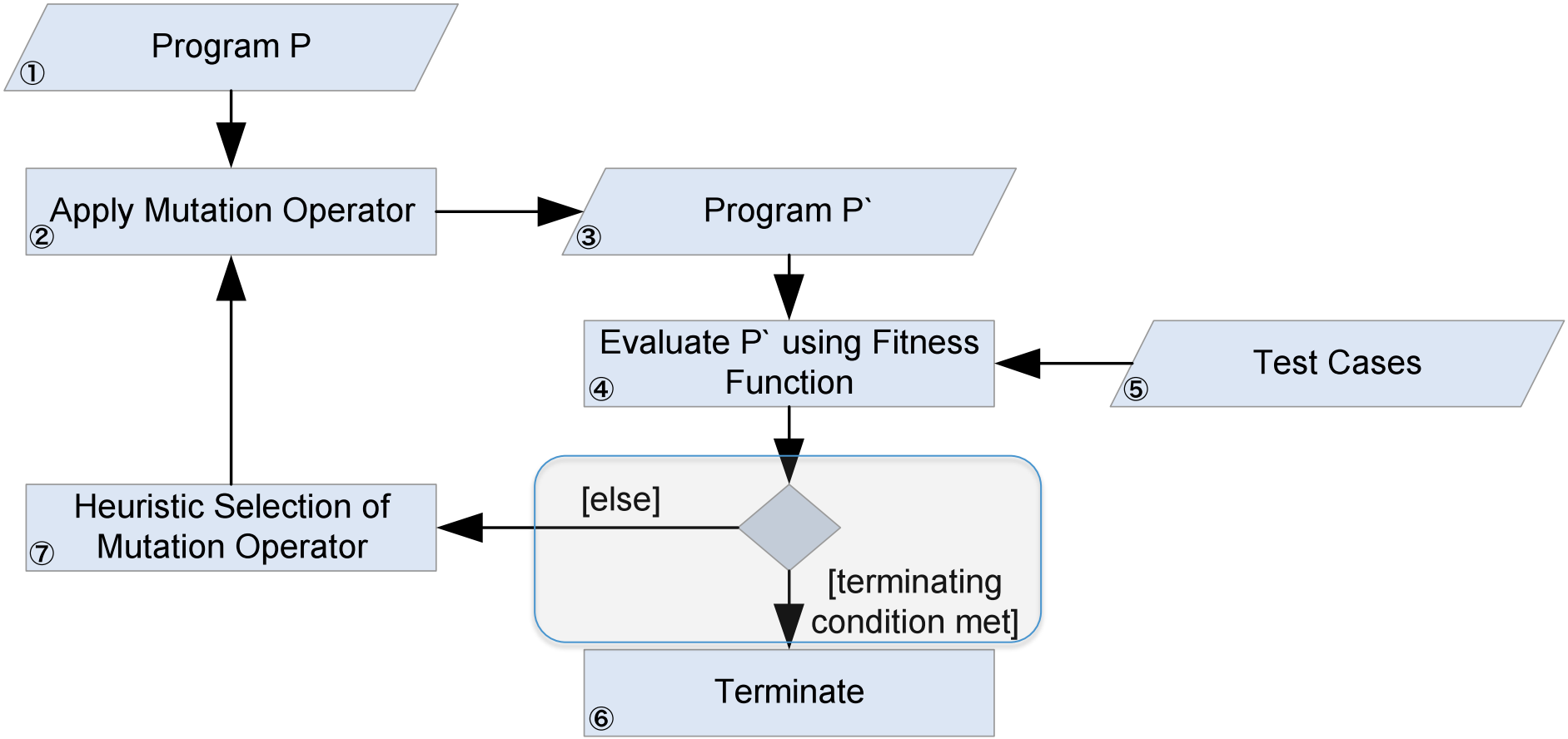
Putting it Together – cont.



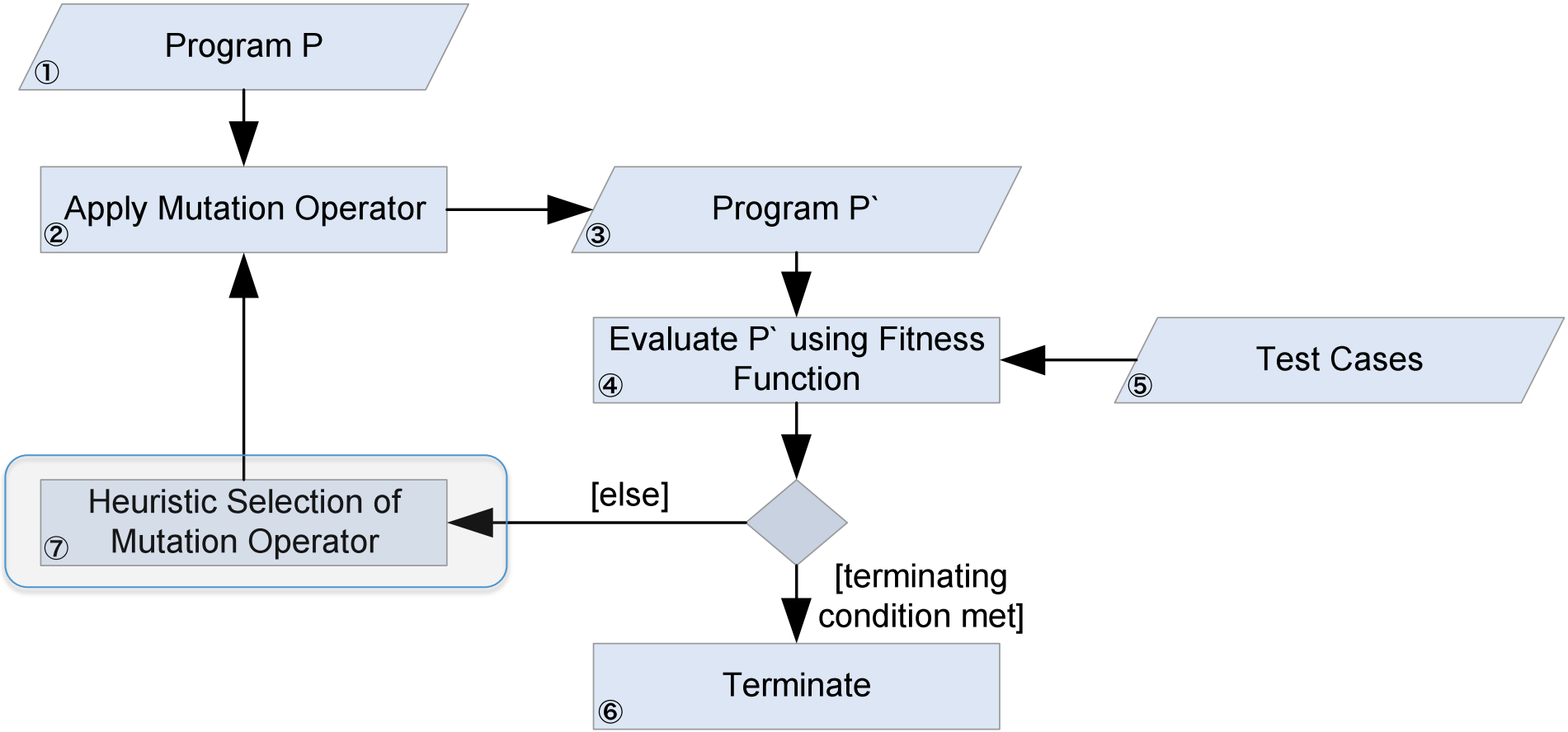
Putting it Together – cont.



Putting it Together – cont.



Putting it Together – cont.



Heuristic Selection of Mutation Operator

- Based on test results, select appropriate mutation operator to fix the bug
- If most test failed due to a deadlock use Mutation Operator #3, otherwise use one of the other two operators

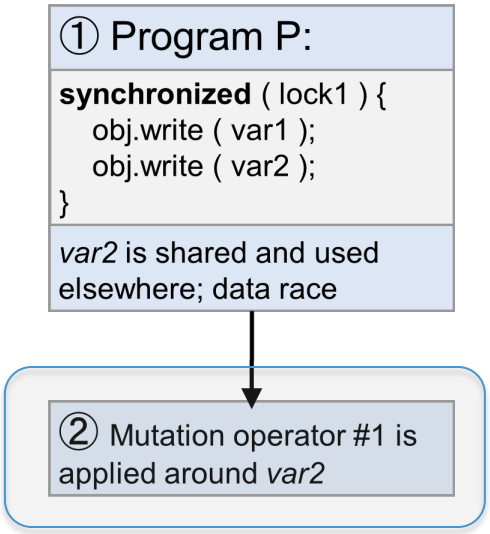
Example Walkthrough

① Program P:

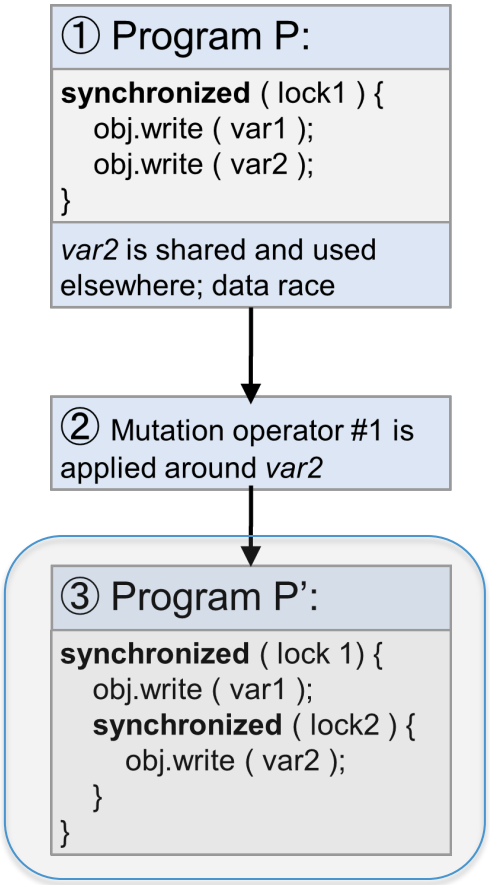
```
synchronized ( lock1 ) {
    obj.write ( var1 );
    obj.write ( var2 );
}
```

var2 is shared and used elsewhere; data race

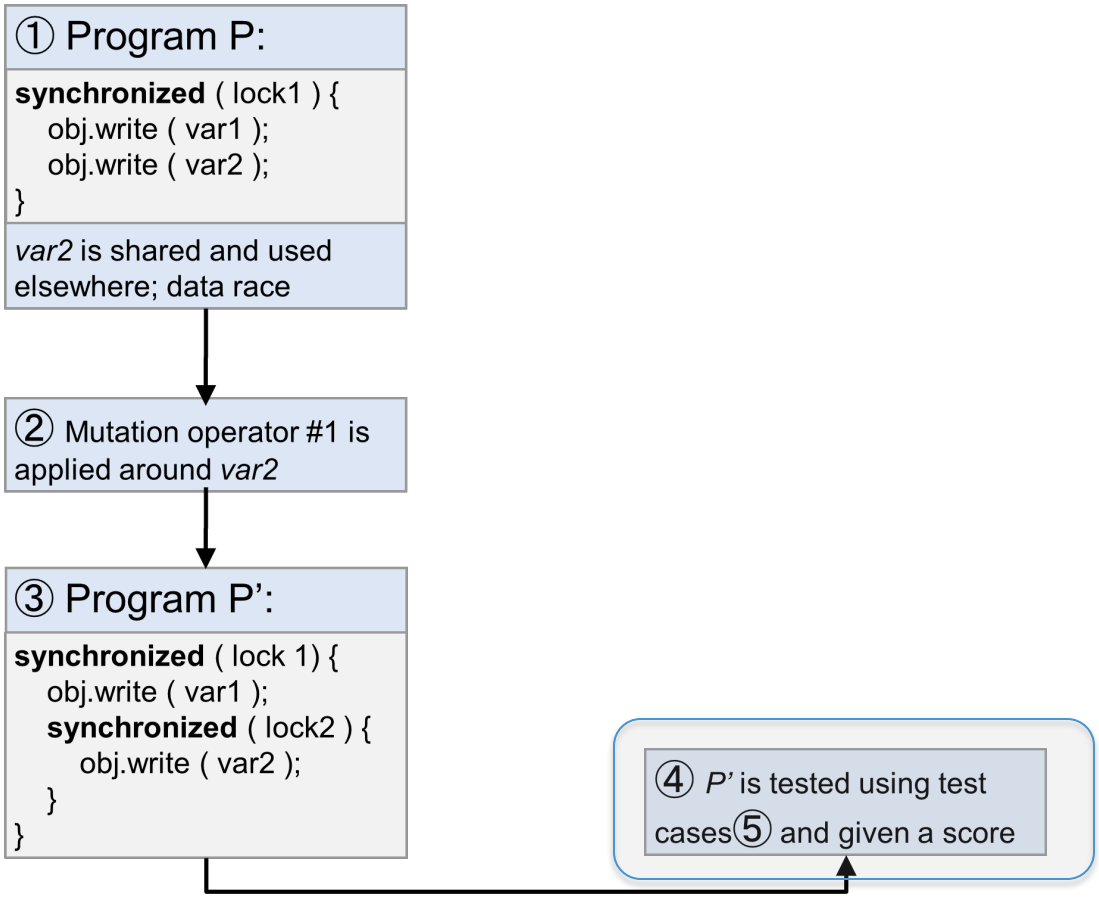
Example Walkthrough – cont.



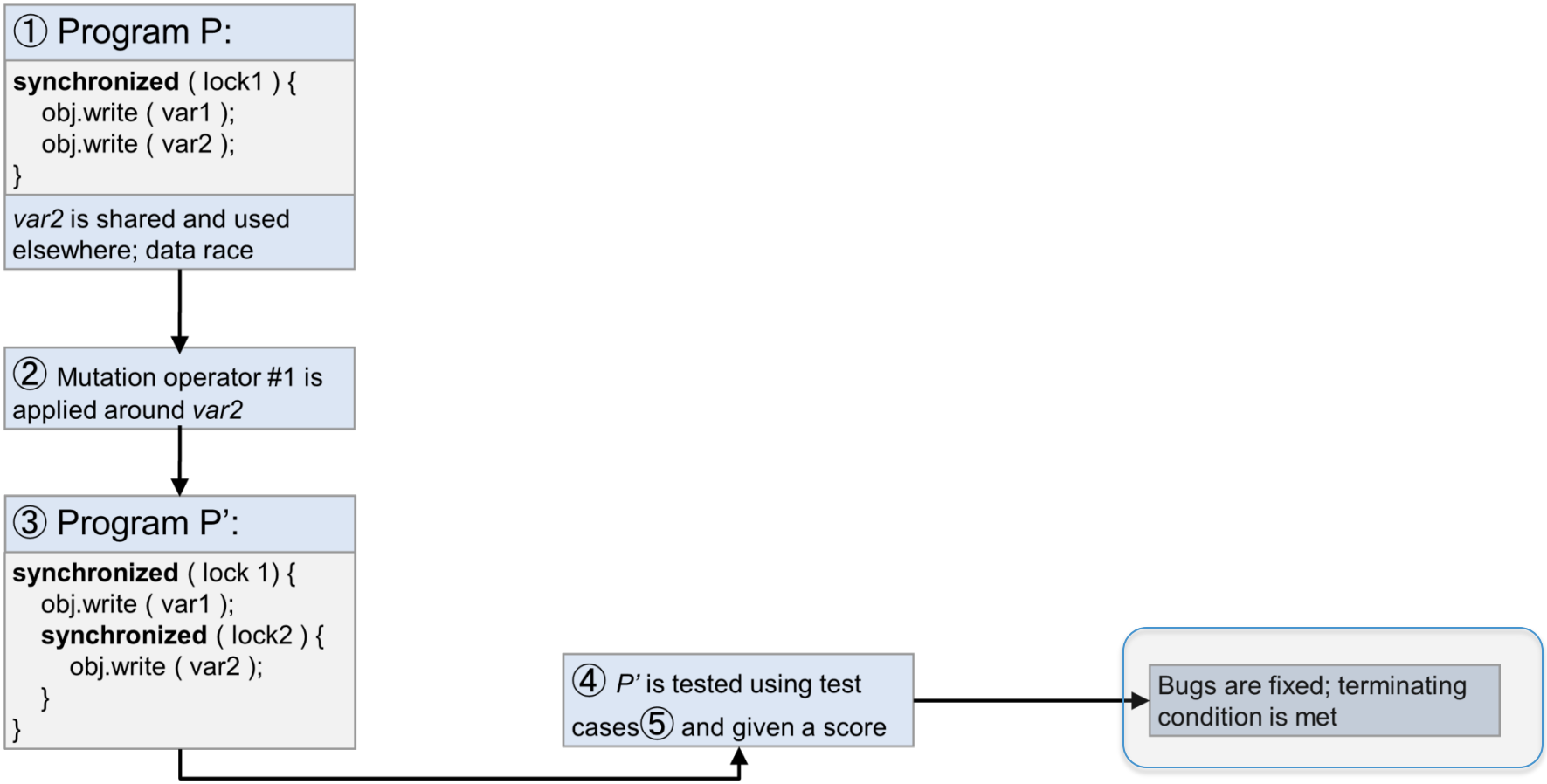
Example Walkthrough – cont.



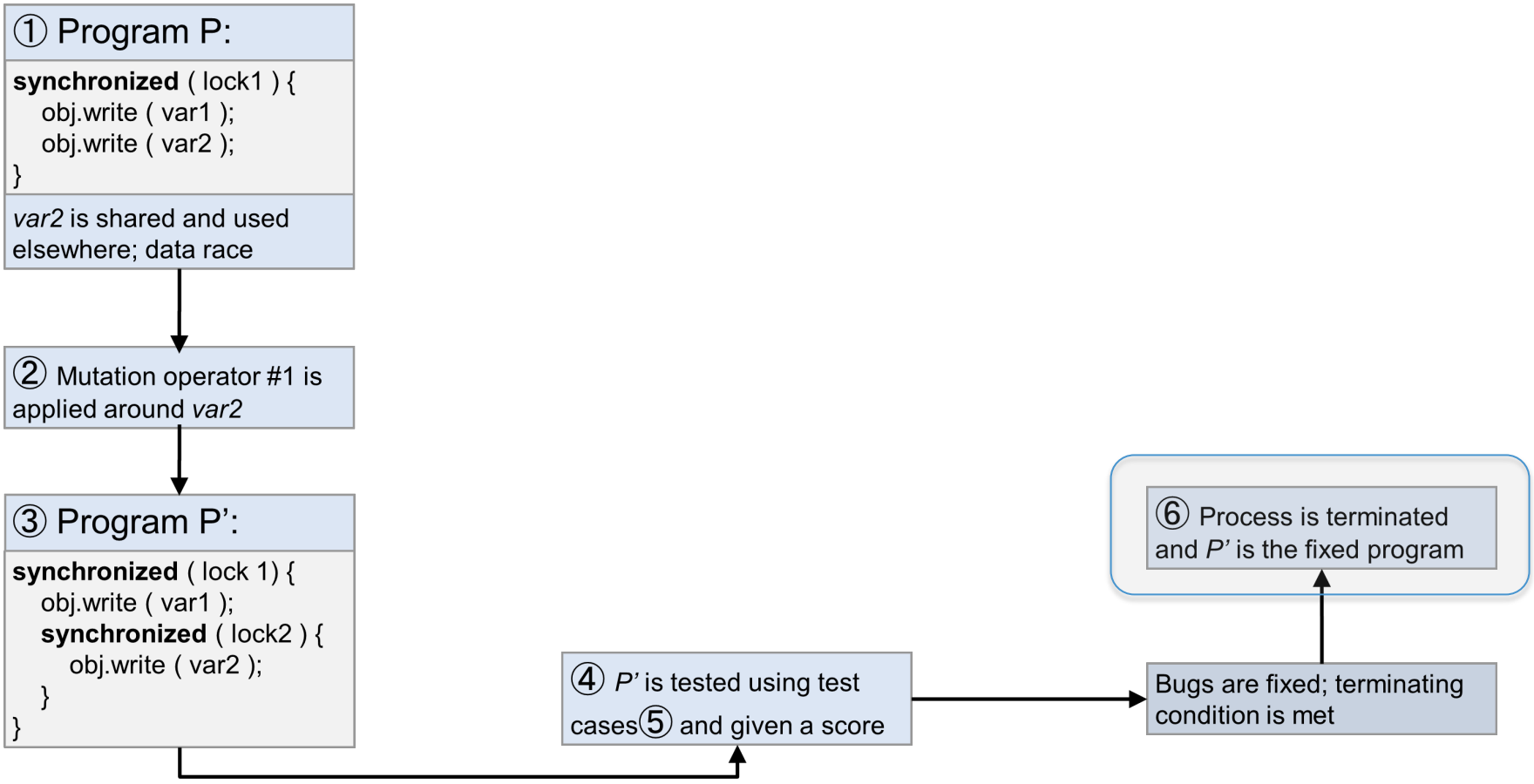
Example Walkthrough – cont.



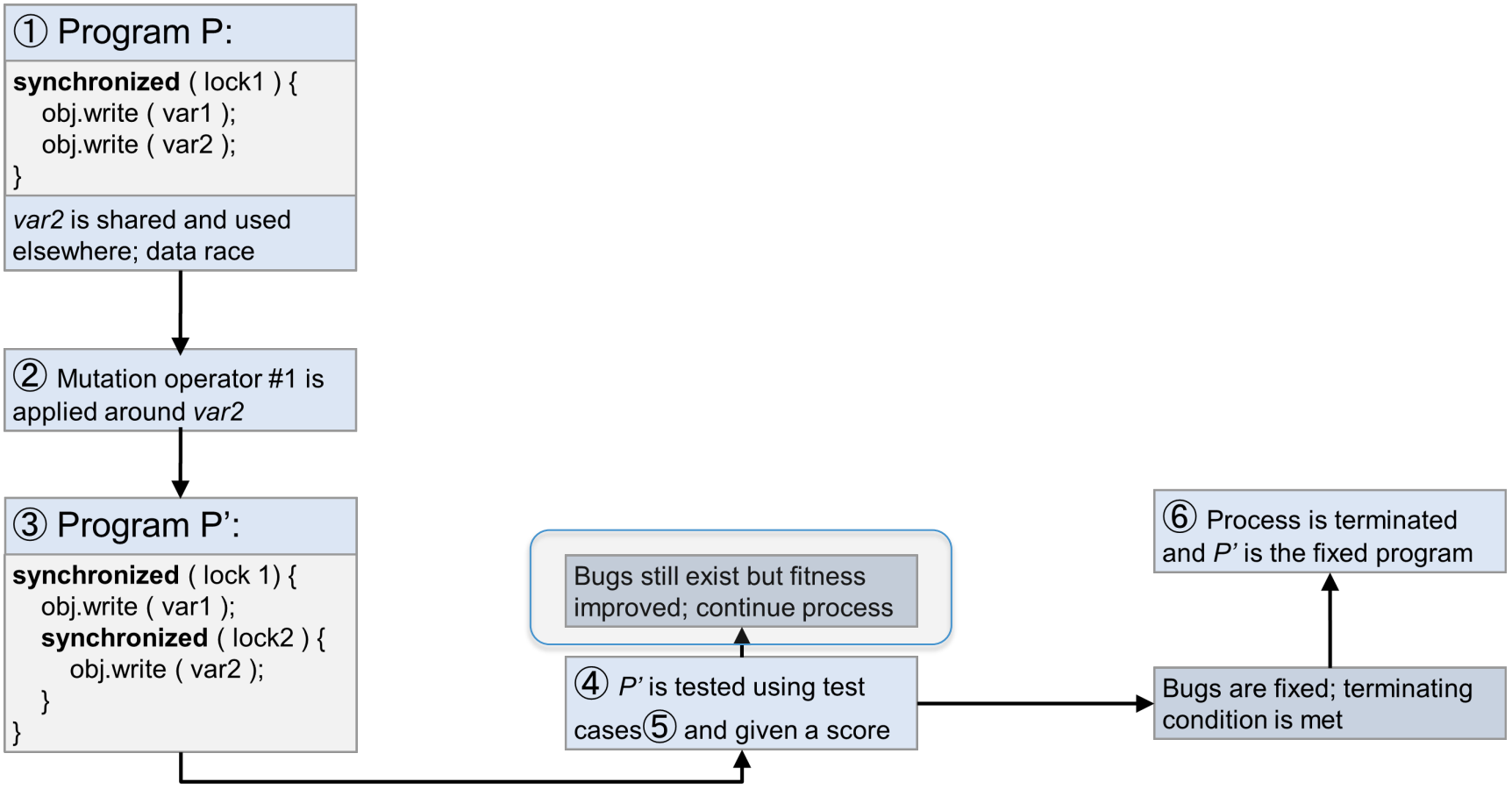
Example Walkthrough – cont.



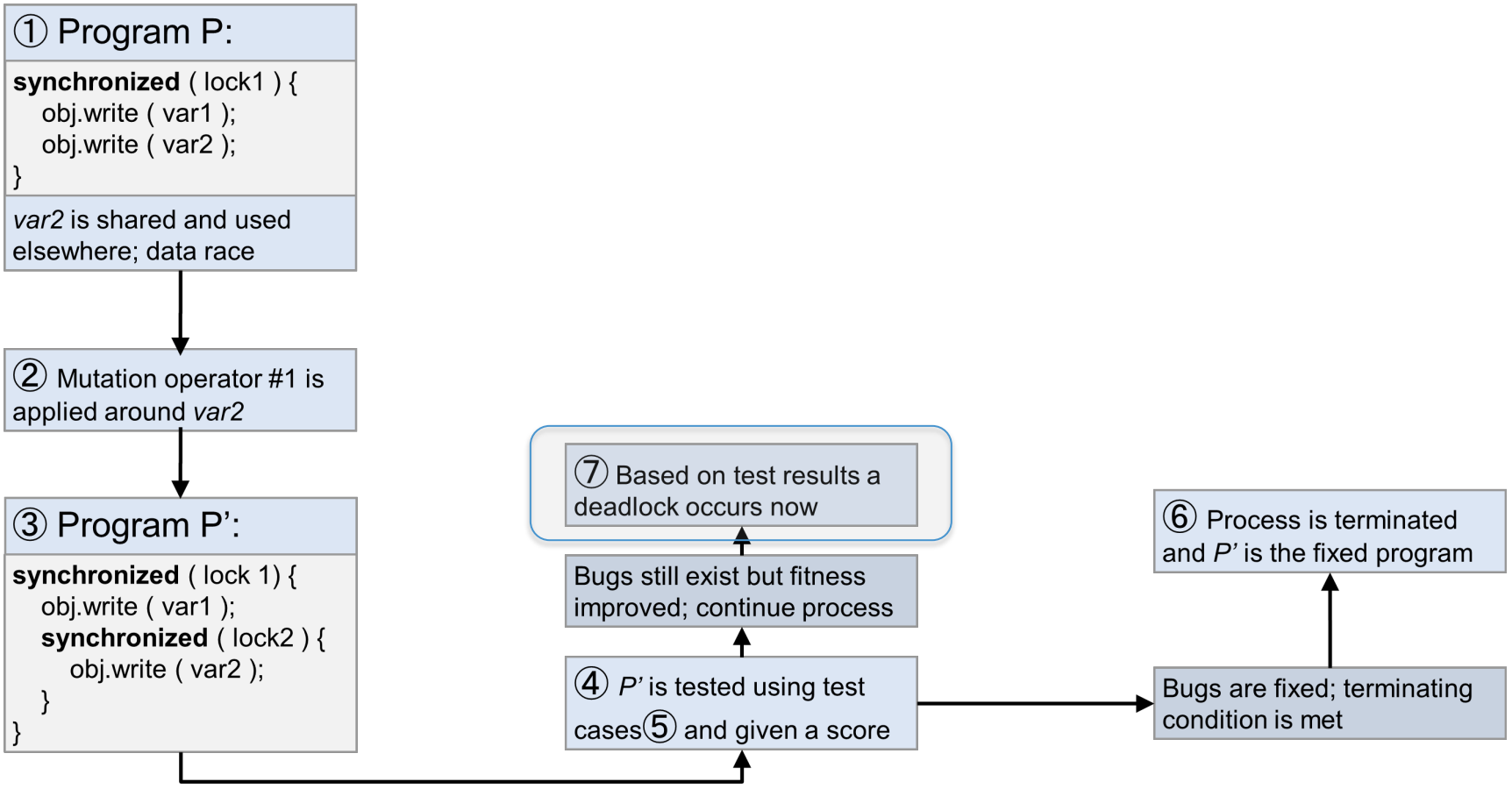
Example Walkthrough – cont.



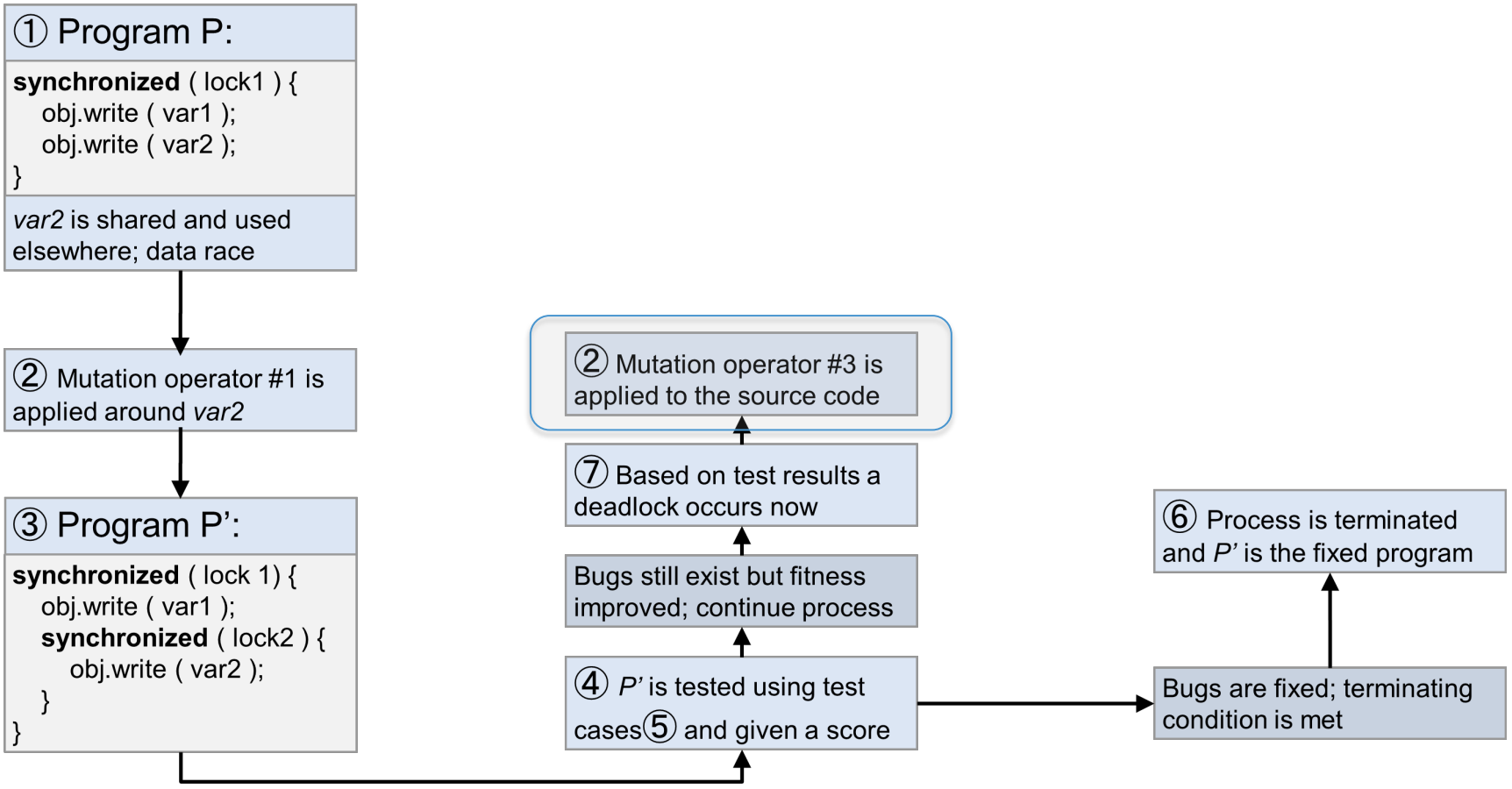
Example Walkthrough – cont.



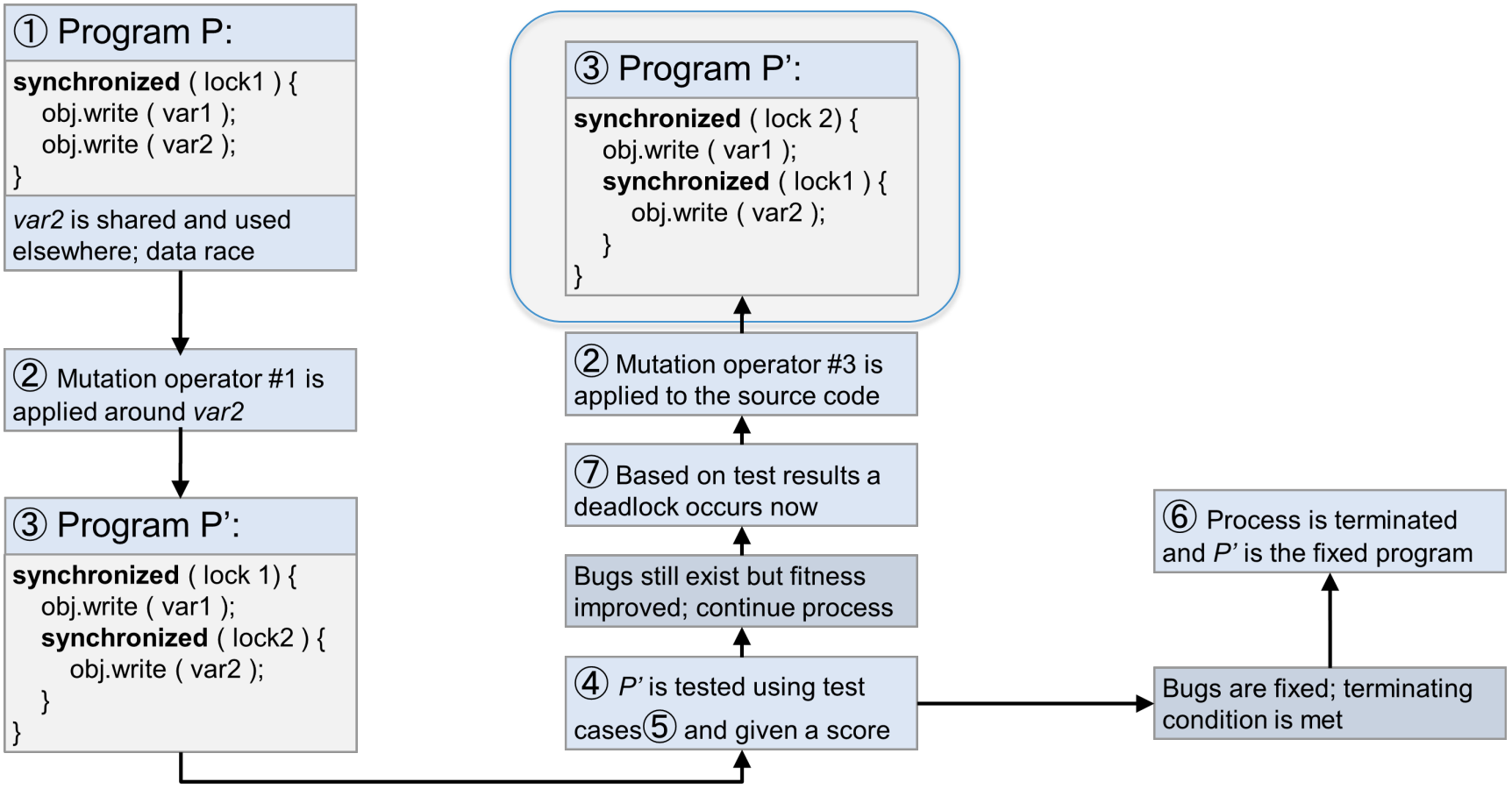
Example Walkthrough – cont.



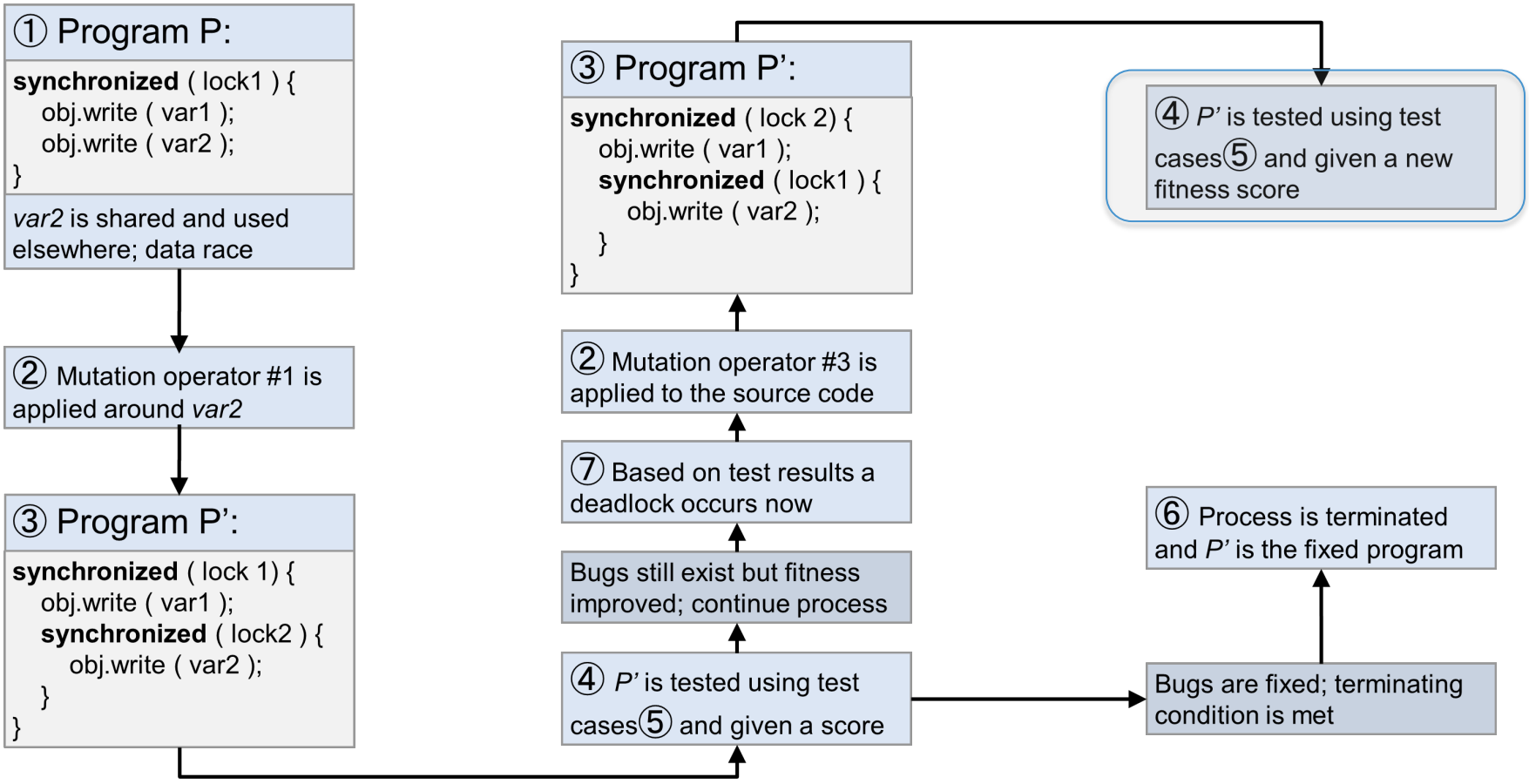
Example Walkthrough – cont.



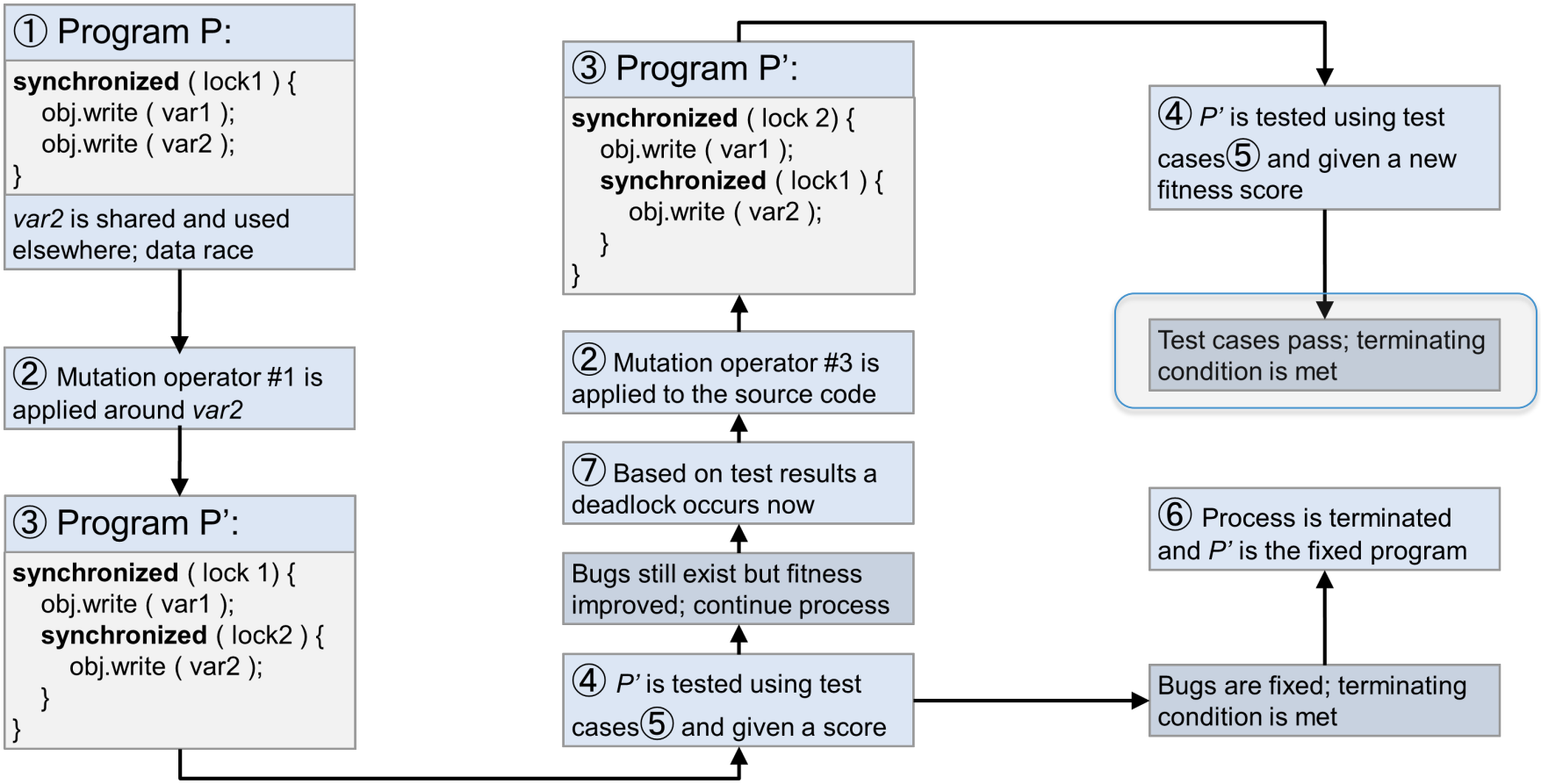
Example Walkthrough – cont.



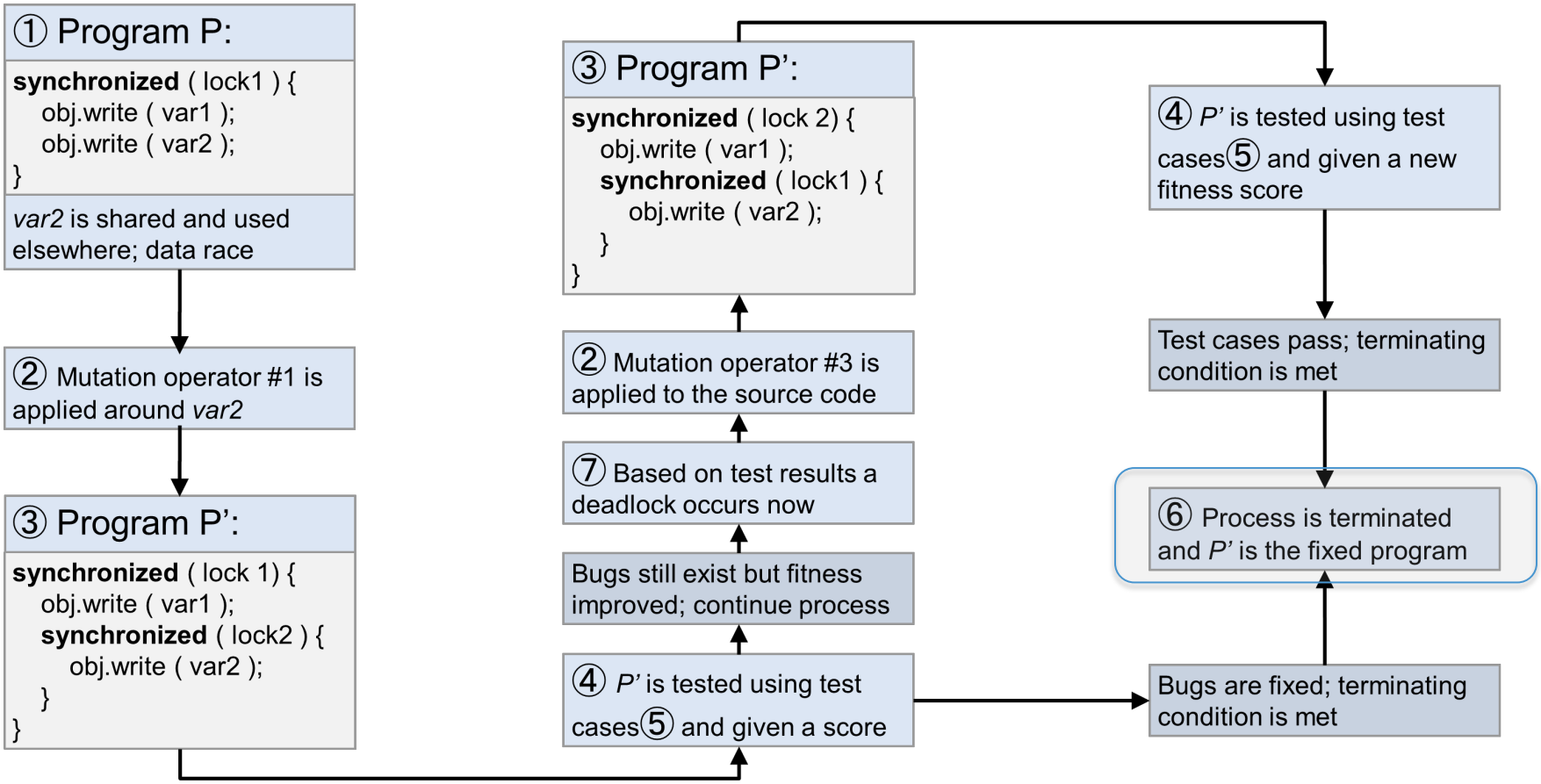
Example Walkthrough – cont.



Example Walkthrough – cont.



Example Walkthrough – cont.



Conclusion and Future Work

- Based on solid foundation from related works
  - Extension of related work to concurrency

- Future Work:
  - Complete implementation
  - Evaluation and optimization:
    - Mutation Operators
    - Fitness Function
    - Overall approach

Automatic Repair of Concurrency Bugs

Jeremy S. Bradbury, **Kevin Jalbert**

Software Quality Research Group
University of Ontario Institute of Technology
Oshawa, Ontario, Canada

{jeremy.bradbury, kevin.jalbert}@uoit.ca

http://faculty.uoit.ca/bradbury/sqrg/

UOIT CHALLENGE INNOVATE CONNECT SQR GROUP

Funding provided by: NSERC CRSNG