Finding Feasible Counter-examples when Model Checking Abstracted Java Programs

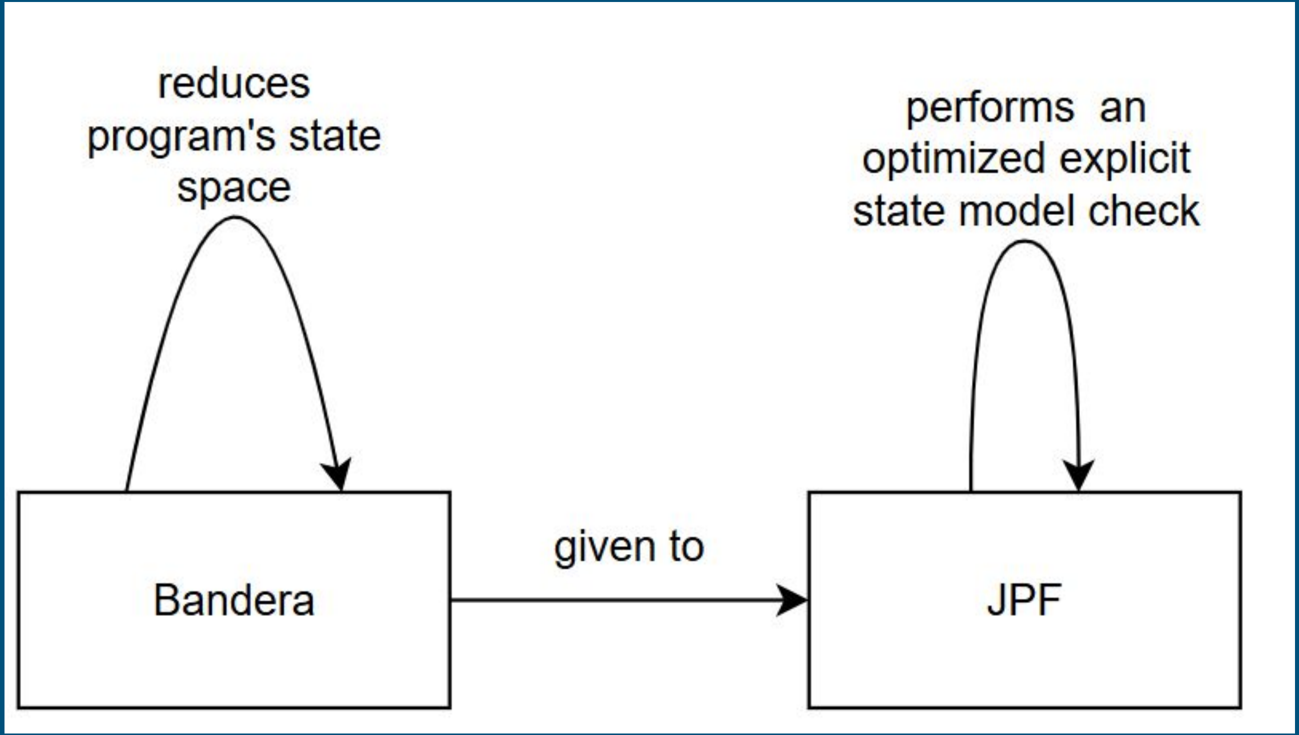
Corina S. Pasareanu1, Matthew B. Dwyer1, and Willem Visser2

Diogo Silva, 95552
Pedro Abreu, 92541
Francisco Santos, 111237

Introduction

- The state explosion problem
- Scaling Model Checking to larger systems techniques
- Problem finding the specification to be false
- JPF to try to to solve it
- Importance of model checking over the years
- Opinion of researchers on property-preserving abstraction
- Challenge addressed by this paper
- Integration of Java, Bandera and JPF described in the paper

Proposed solution


reduces program's state space
performs an optimized explicit state model check
given to
Bandera
JPF

Proposed solution

- Bandera uses abstraction that preserves the ability to prove all paths properties
- The work describes in this paper:
  - treats:
    - abstraction of the program’s data
    - runtime system scheduler
    - property to be checked.
  - evaluates the feasibility against the semantics of a real programming language
  - there are different approaches for different cost profiles
- The precision of the abstract model increases when we minimize the use of nondeterminism
- Benefits from using Bandera and JPF

Proposed solution

Steps for verifying a property from a concrete program:

- Abstraction mapping
- Property transformation
- Verification
- Inference

Proposed solution

Abstract Interpretation (AI)

- Data abstraction
- Domain of abstract values
- Abstraction function
- Abstract primitive operations
- Property abstraction
- Scheduler Abstraction

Abstraction implementation is done with Bandera

```
public class Signs {
    public static final int NEG =0;
    public static final int ZERO=1;
    public static final int POS =2;
    public static int abs(int n) {
        if (n< 0) return NEG;
        if (n == 0) return ZERO;
        if (n > 0) return POS;
    }
}
```

Proposed solution

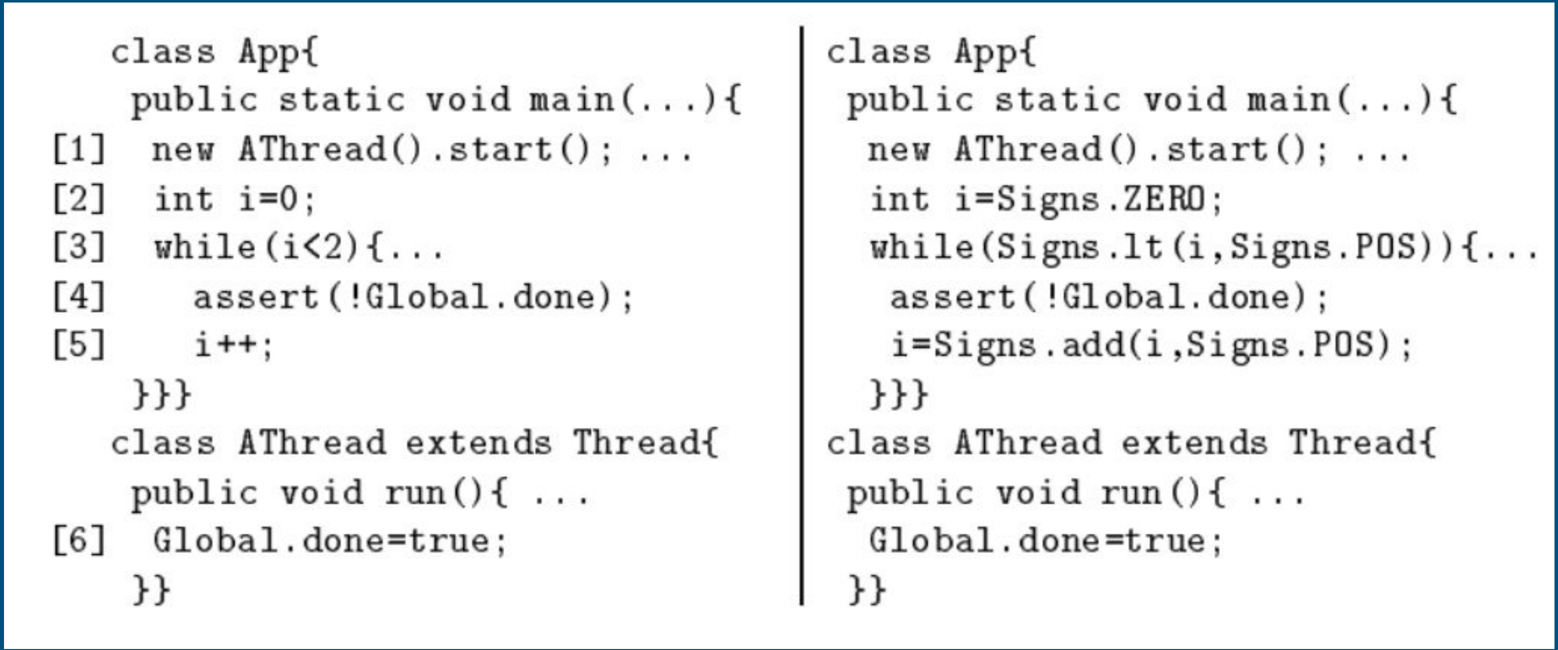
Choose-free State Space Search

- Enhanced JPF model checker to search paths free from nondeterminism
- Search algorithm backtracks on encountering non deterministic choice.

Theorem:

Every deterministic path in the abstracted program corresponds to a path in the concrete program.

Proposed solution



```
    class App{
     public static void main(...){
[1]   new AThread().start(); ...
[2]   int i=0;
[3]   while(i<2){...
[4]     assert(!Global.done);
[5]     i++;
     }}}
    class AThread extends Thread{
     public void run(){ ...
[6]   Global.done=true;
     }}
```

```
class App{
 public static void main(...){
  new AThread().start(); ...
  int i=Signs.ZERO;
  while(Signs.lt(i,Signs.POS)){...
   assert(!Global.done);
   i=Signs.add(i,Signs.POS);
 }}}
class AThread extends Thread{
 public void run(){ ...
  Global.done=true;
 }}
```

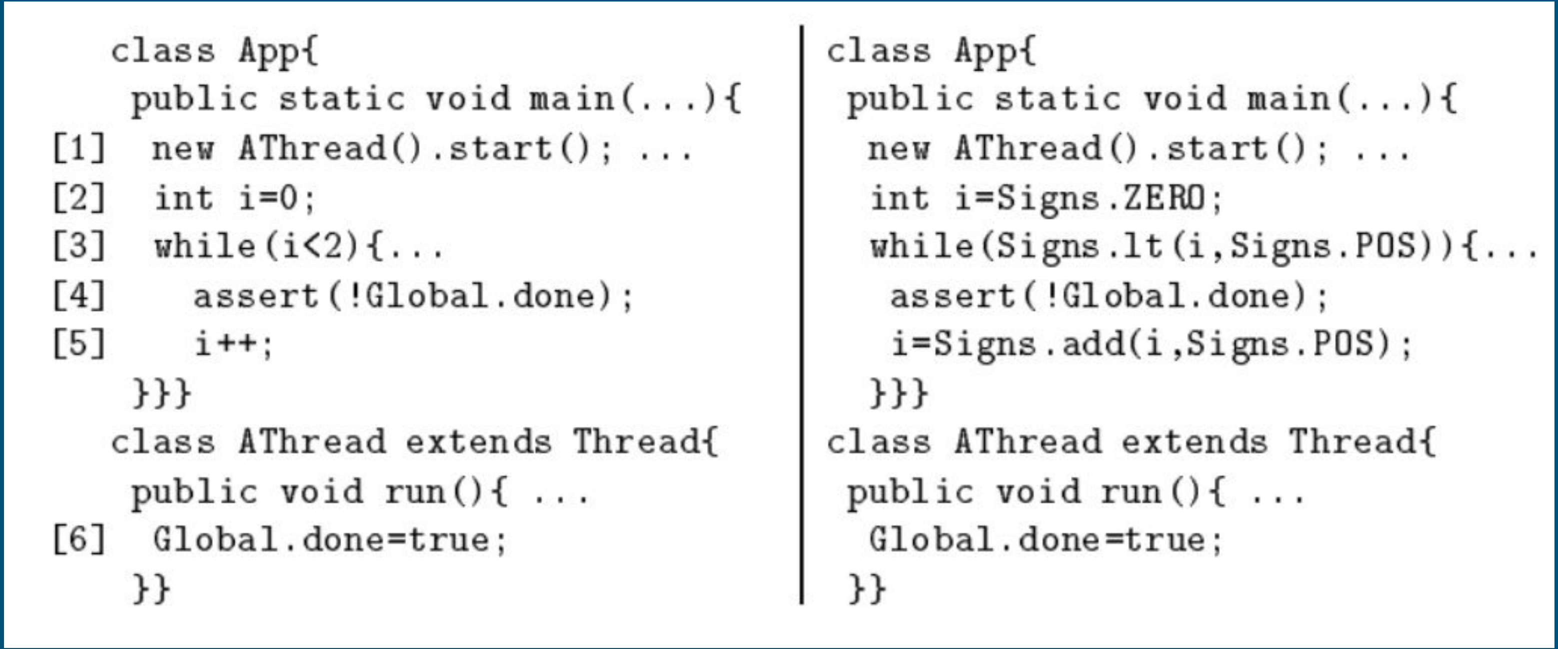
Simple example of concrete (left) and abstracted (right) code.

Proposed solution

Abstract Counter-example Guided Concrete Simulation

- Bandera generates an abstracted program with a clear line-to-line correspondence to the concrete program.
- Each concrete bytecode maps to a set of abstract byte-codes executing atomically in JPF.
- JPF simulates concrete execution using abstract counter-examples.
- Concrete execution line must match the abstract line in the counter-example.

Proposed solution



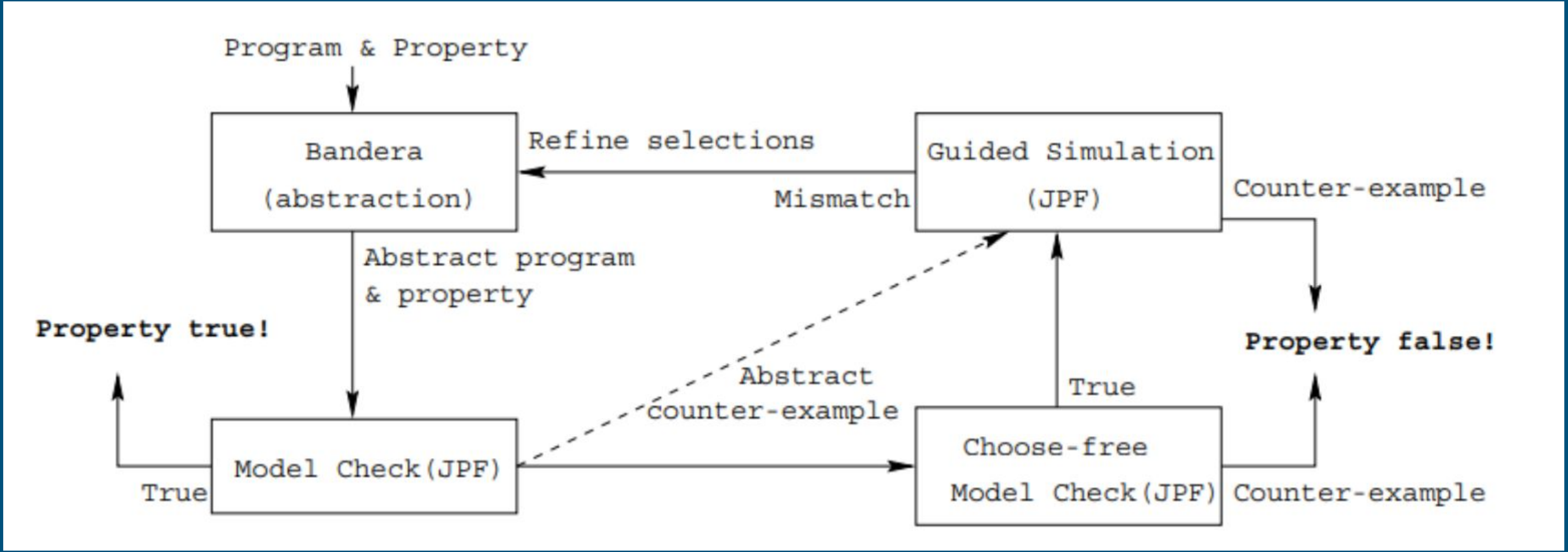
```
class App{
 public static void main(...){
  new AThread().start(); ...
  int i=0;
  while(i<2){...
    assert(!Global.done);
    i++;
 }}}
class AThread extends Thread{
 public void run(){ ...
  Global.done=true;
 }}
```

```
class App{
 public static void main(...){
  new AThread().start(); ...
  int i=Signs.ZERO;
  while(Signs.lt(i,Signs.POS)){...
   assert(!Global.done);
   i=Signs.add(i,Signs.POS);
  }}}
class AThread extends Thread{
 public void run(){ ...
  Global.done=true;
 }}
```

Simple example of concrete (left) and abstracted (right) code.

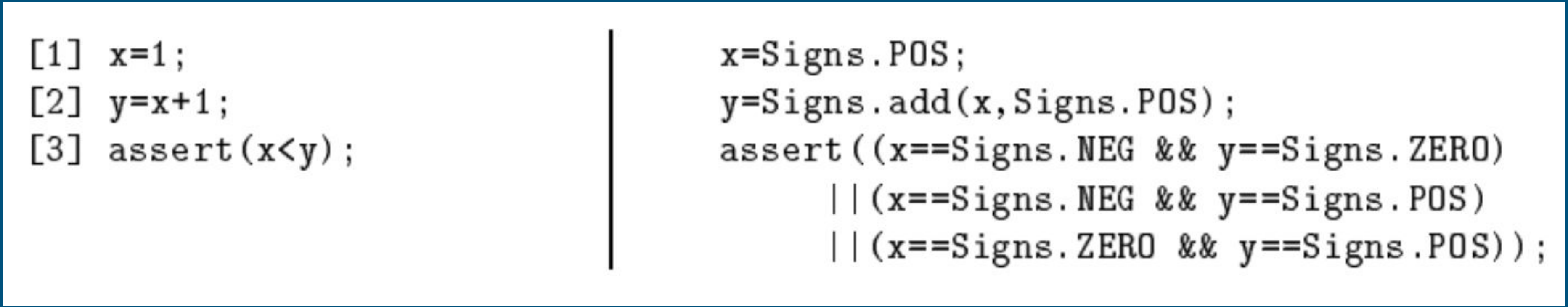
Proposed solution

Methodology



Proposed solution

Discussion



```
x=1;
y=x+1;
assert(x<y);
```

```
x=Signs.POS;
y=Signs.add(x,Signs.POS);
assert((x==Signs.NEG && y==Signs.ZERO)
      ||(x==Signs.NEG && y==Signs.POS)
      ||(x==Signs.ZERO && y==Signs.POS));
```

Defective Programs

To further assess the efficiency of these techniques, a set of multi-threaded concurrency/synchronization-based programs were put to the test:

- **RAX (Remote Agent Experiment);**
- **Pipeline;**
- **RWVSN;**
- **DEOS;**

Discussion

The experiments produce results with significant value that suggest that:

- The proposed techniques can reduce the length of counter-examples;
- The proposed techniques can guarantee feasible counter-examples;
- Choose-free model check is faster than a typical model check;
- Choose-free searches can pave the way for more aggressive abstractions (further optimising checks);

Future Work & Conclusion

In terms of reflecting on the quality of the work carried out and future appliances of the paper, it can be noted that:

- Choose-free search and counter--example guided simulation could be implemented and automated in any explicit-state model checker (i.e. Bandera);
- The techniques suggested yielded positive results and suggest that their use could be scaled up when resorting to model checking in the industry at large;