TracNav

- JPFWiki - Welcome Page

  Introduction...

  Installing JPF...

  User Guide...

  Developer Guide...

  Projects

  - jpf-core
  - jpf-actor
  - jpf-awt
  - jpf-awt-shell
  - jpf-concurrent
  - jpf-cv
  - jpf-delayed
  - jpf-guided-test
  - jpf-mango
  - jpf-racefinder
  - jpf-rtembed
  - jpf-statechart
  - net-iocache
  - jpf-aprop
  - jpf-numeric
  - jpf-symbc
  - jpf-concolic
  - jpf-symbc-load?
  - jpf-extended-test-gen
  - jpf-parallel-spf?
  - eclipse-jpf
  - netbeans-jpf
  - jpf-inspector
  - jpf-shell
  - jpf-template
  - jpf-trace-server
  - standard NB example
- Summer Projects
- External Projects
- Change(B)log

  About...

- Events
- Presentations
- Papers
- FAQ
- History?
- Support
- People?
- Playground
- Table of Context

jpf-delayed

Milos Gligoric and Tihomir Gvero, {milos.gligoric, tihomir.gvero}@gmail.com, January 2010

Repository

The repository for jpf-delayed is http://babelfish.arc.nasa.gov/hg/jpf/jpf-delayed.

Delayed Choice

The basic ***delayed choice*** postpones non-deterministic choice of values until they are used, reducing the size of the search tree. The technique works with both int and boolean, i.e., with Verify.getInt and Verify.getBoolean methods. Additionally, we speed up the basic ***delayed choice*** by introducing copy propagation that keeps non-deterministic values symbolic even if they are copied through memory locations. We also implement a special class for linked structures, called ObjectPool, which has the following methods for non-deterministic assignments of objects:

```
public final class ObjectPool<T> implements Iterable<T> {
 public ObjectPool(Class<?> clz, int size, boolean includeNull) {...}
 public T getAny() {...}
 public T getNew() {...}
 public Iterator<T> iterator() {...}
}
```

The constructor can create finite (if size > 0) and infinite (if size == -1) ***pools*** of a given type (clz), which may (if includeNull == true) or may not (if includeNull == false) include the value "null". The method getAny non-deterministically returns any value from the ***pool*** (including optionally "null"), whereas getNew returns an object that was not returned by previous calls (and never "null"). These methods avoid generating structures that are isomorphic due to the fact that all fresh objects are observationally equivalent. Therefore, getAny returns either one of the previously returned objects or only the first object that was not returned before. Iterator allows a user to iterate over all objects in the ***pool***.

Simple Examples

The following are two very simple examples of using ***delayed choice***. Consider the code shown below, that uses Verify.getInt for non-deterministic assignment:

```
int x = Verify.getInt(a, b); // (1)
... // code that doesn't use variable "x"
int y = x; // (2) "x" is used for the first time
assert(x == 10) // (3)
```

By default, JPF assigns a concrete value at line (1) and creates a non-deterministic choice point. If ***delayed choice*** is used, a concrete value is not assigned until point (2), i.e., when the variable is used for the first time. Moreover, if ***delayed choice*** is used with ***copy propagation***, a concrete value is not assigned until point (3), i.e., the first non-copy use of the variable "x".

The high-level idea of ***delayed choice*** for objects is the same as for getInt, but the implementation for object pools is more complex because getNew is a stateful command. To preserve the set of reachable states of the eager implementation, delayed choice introduces symbolic values at each call to getNew or getAny and also accumulates the constraints imposed by the requirement that getNew returns objects distinct from previously returned objects. Here is a simple example showing the use of getAny and getNew methods:

```
p = new ObjectPool<Node>(Node.class, 3); //default value for "includeNull" is false
n1 = p.getNew();
a1 = p.getAny();
a2 = p.getAny();
a3 = p.getAny();
n2 = p.getNew();
n3 = p.getNew();
use(a1); use(a2); use(a3);
```

The ***delayed choice*** will pick the concrete values of a1, a2, a3 only at their use points. When it picks the values, it must have sufficient information to deduce that all values a1, a2, a3 must be equal; otherwise, it will be impossible, in the pool of size 3 where getAny doesn't return "null", to assign values n2, n3 such that n2 is not in the set {n1, a1,a2, a3} and n3 is not in the set {n1, a1, a2, a3, n2}.

Running

To run JPF in the basic ***delayed choice*** mode, one can provide the following program arguments:

```
+vm.insn_factory.class=gov.nasa.jpf.delayed.ncp.NCPInstructionFactory
+vm.restorer.class=gov.nasa.jpf.jvm.NCPDelayedRestorer
```

To run JPF in the ***copy propagation delayed choice*** mode, one can provide the following program arguments:

```
+vm.insn_factory.class=gov.nasa.jpf.delayed.cp.CPInstructionFactory
+vm.restorer.class=gov.nasa.jpf.jvm.CPDelayedRestorer
```

Acknowledgments

This work was partially done while we were interns at the Swiss Federal Institute of Technology in Lausanne (EPFL), during summer 2008. The project was mentored by Prof. Viktor Kuncak from EPFL, and was joint work with Vilas Jagannath and Prof. Darko Marinov from the University of Illinois at Urbana-Champaign and Prof. Sarfraz Khurshid from the University of Texas at Austin. Technical details of this work are described here.